ON COMPUTING A LIVENESS ENFORCING SUPERVISORY POLICY FOR A CLASS OF GENERAL PETRI NETS

BY

NISHA SOMNATH

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Industrial Engineering in the Graduate College of the University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Associate Professor Ramavarapu Sreenivas, Chair, Director of Research
Associate Professor Carolyn Beck
Associate Professor Dusan Stipanovic
Professor Petros G. Voulgaris
Discrete-Event/Discrete-State (DEDS) Systems are prone to livelocks. Once a system enters a livelocked-state, there is at least one activity of the modeled system that cannot be executed from all subsequent states of the system. This phenomenon is common to many operating systems where some process enters into a state of suspended animation for perpetuity, and the user is left with no other option than to terminate the process, or reboot the machine. This thesis is about computing Liveness Enforcing Supervisory Policies (LESPs) for Petri net (PN) models of DEDS systems. The existence of an LESP for general PNs is not even semi-decidable.

This thesis identifies two classes of PNs $\mathcal{F}$ and $\mathcal{H}$ for which the existence of a LESP is decidable. It also describes an object-oriented implementation of a procedure for the synthesis of the minimally-restrictive LESP for any instance from these classes. The minimally-restrictive LESP prevents the occurrence of events in a DEDS system only when it is absolutely necessary.

A suite of methods, based on refinement/abstraction concepts, is developed to reduce the complexity of LESP-synthesis. This involves the synthesis of a LESP for a simplified-version of a complex PN structure, which is subsequently refined to serve as a LESP for the original complex PN.

Two PNs are in a simulation relationship if their behaviors are “similar” in a formal sense. The thesis concludes with a result that shows that the above mentioned procedure can be generalized to PNs in simulation relationships. That is, a LESP for a PN can be modified to serve as a LESP for another PN that is “similar.” The implementation of this theoretical observation is suggested as a topic for future work.
I dedicate this thesis to my family: Mom, Dad, Pratyush and Ammuma
ACKNOWLEDGMENTS

I am extremely grateful to my advisor Professor Ramavarapu Sreenivas for giving me the opportunity to work on this research. Without his constant support and guidance this thesis would not have been possible. It was a real privilege to be his student and words alone cannot express my gratitude towards him.

This work would not have been possible if it were not for the constant encouragement and support from my family. I thank my mom for her incredible patience and my dad for never losing his faith in me. I may have been a source for some of their stress and worry and this thesis would not have been accomplished without their endless sacrifices.

I would like to acknowledge my wonderful friends Maria Jones, Spurti Akki and Deepika Sreedhar. They stood by me through some trying times and never let me give up.
# TABLE OF CONTENTS

LIST OF FIGURES ................................................................. vi

CHAPTER 1 INTRODUCTION ....................................................... 1

CHAPTER 2 A REVIEW OF PETRI NETS ......................................... 5
  2.1 Notations, definitions and other preliminary observations .... 5

CHAPTER 3 MARKING-BASED SUPERVISORY CONTROL
  OF PNS .................................................................................. 13
  3.1 Supervisory Policies for Liveness Enforcement ............ 13

CHAPTER 4 RECENT THEORETICAL RESULTS AND OBSERVATIONS ON LESP-SYNTHESIS ................................................................. 22
  4.1 $\mathcal{F}$ class PNs [1] ....................................................... 22
  4.2 $\mathcal{H}$ class PNs [2] ....................................................... 28
  4.3 On the role of Choice/Non-Choice Transitions in Supervisory Control of PNs ......................................................... 30
  4.4 LESPs for Petri Nets that are Similar [3] ......................... 32

CHAPTER 5 LESP-SYNTHESIS ALGORITHM AND OBJECT-ORIENTED IMPLEMENTATION ................................................................. 38

CHAPTER 6 REDUCTION RULES AND OBJECT ORIENTED IMPLEMENTATION OF REDUCTION ALGORITHM ......................................................... 42
  6.1 Reduction Rules ............................................................... 42
  6.2 Implementation of Reduction Techniques .................. 47
  6.3 Deducing Minimal Elements of the Original PN ........ 59
  6.4 Examples .......................................................... 66

CHAPTER 7 CONCLUSIONS AND FUTURE WORK ............................ 94

REFERENCES ............................................................................ 97
LIST OF FIGURES

1.1 Livelocked Mozilla Firefox .......................... 2
1.2 Error Reporting System used by Windows ........... 3

2.1 A general FCPN structure $N_1 = (\Pi_1, T_1, \Phi_1, \Gamma_1)$ .................. 8
2.2 A general FCPN structure $N_2 = (\Pi_2, T_2, \Phi_2, \Gamma_2)$ ................. 8
2.3 The procedure for the construction of the Reachability Tree of a PN $N(m^0)$, where $N = (\Pi, T, \Phi, \Gamma)$ ............................... 10
2.4 A PN $N_3(m^0_3)$ that is not bounded and not live (cf. section V.C, [4]) .... 11
2.5 Coverability graph for the PN $N_3(m^0_3)$ in 2.4. It can be seen that the transition $t_1$ is not fired even once ............... 11
2.6 A PN $N_4(m^0_4)$ with infinitely large set of markings .................. 12
2.7 Finite coverability graph for $N_4(m^0_4)$ of figure 2.6 with infinitely large set of markings .................. 12

3.1 The procedure for testing the existence of an LESP for a PN $N(m^0)$, where $N = (\Pi, T, \Phi, \Gamma)$, assuming $\Delta(N)$ is right-closed .................. 17
3.2 An FCPN $N_5(m^0_5)$ where $N_5 = (\Pi_5, T_5, \Phi_5, \Gamma_5)$ .............. 18
3.3 The output file generated by the software described in reference [5] for the FCPN shown in figure 3.2 .................. 18

4.1 General FCPN structure $N_6$ .......................... 23
4.2 General FCPN structure $N_7$ .......................... 24
4.3 General FCPN structure $N_8$ .......................... 25
4.4 The minimally-restrictive LESP that ensures all reachable markings of the plant FCPN PN $N_6(m^0_6)$ are in the right-closed set $\Delta(N_6)$ identified by the minimal elements $\{(2 0)^T, (0 3)^T\}$. This policy is a minimally restrictive LESP for any $m^0_6 \in \Delta(N_6)$. There is no LESP for $N_6(m^0_6)$ if $m^0_6 \notin \Delta(N_6)$ .............. 27
4.5 The minimally restrictive LESP for the general plant PN $N_7(m^0_7)$ . This LESP ensures the markings reachable under its supervision from any $m^0_7 \in \Delta(N_7)$ remain within the right-closed set $\Delta(N_7)$, where $\text{min}(\Delta(N_7)) = \{(0 0 1 0)^T, (2 0 0 0)^T, (1 0 0 1)^T, (0 2 0 0)^T, (0 0 0 2)^T\}$ .............. 28
4.6 (a) The PN structure $N_6$ is a member of $\mathcal{H}$ as it is an FCPN structure. (b) The PN structure $N_7$ is also a member of $\mathcal{H}$. The non-choice transition $t_5$ is controllable (uncontrollable) in $N_6$ ($N_7$).

4.7 If $\hat{\mathcal{P}}$ is a policy that permits the firing of $\hat{t}_6$ if and only if $(\hat{m}_1 + \hat{m}_2 + \hat{m}_3 + \hat{m}_4 + \hat{m}_6 + \hat{m}_7) \geq 2$; and, permits the firing of $\hat{t}_1(=\alpha(t_0))$ at every marking in $\beta^{-1}(m)$ if and only if $\mathcal{P}$ permits the firing of $t_0$ at marking $m$, then $\hat{N}(\hat{m}_0)$ under $\hat{\mathcal{P}}$ simulates $N(m_0)$ under $\mathcal{P}$. From Theorem 4.4.2 we infer that the policy $\mathcal{P}$ is an LESP for $N(m_0)$ if and only if every transition in $\alpha(T) = \{\hat{t}_1, \hat{t}_8, \hat{t}_9, \hat{t}_{10}, \hat{t}_{11}, \hat{t}_{12}, \hat{t}_{13}\}$ is live under $\hat{\mathcal{P}}$ in $\hat{N}$.

5.1 Class Diagram.

5.2 $N_7(m_0^7)$

5.3 Input file for $N_7(m_0^7)$. The first line shows that there are four places and six transitions in the PN structure. This is followed by the $(4 \times 6)$ IN and OUT matrices, which accounts for the eight lines that follow the first line. The penultimate line identifies the initial marking $m_0^7$ that places two tokens in place $p_1$, and the last line identifies the controllable (uncontrollable) transitions with a 1(0). Since this line is all zeros, but for the fifth position, it follows that the only controllable transition in this structure is $t_5$.

5.4 The output file generated from the input file of Figure 5.3.

6.1 Rule #1.

6.2 Rule #2.

6.3 Rule #3.

6.4 Class Diagram for Reduction Techniques.

6.5 Class structure of MarkingVector.

6.6 Class structure - variables of class Reduction.

6.7 Class structure - methods of class Reduction.

6.8 Flowchart of Reduc() method.

6.9 Flowchart of reduction_path() method.

6.10 Flowchart of reduction_matrix() method.

6.11 Flowchart of reduction_matrix.t() method.

6.12 Flowchart of member functions of class Reduction.

6.13 Class Diagram for Deducing Minimal Elements.

6.14 Class structure of PetriNet.

6.15 Class structure of Minele.

6.16 Flow of Implementation.

6.17 Example-1.

6.18 Input file for Example-1.
6.19 Output for Reduction part of the algorithm for Example-1 69
6.20 Reduced Example-1 ........................................... 69
6.21 Minimal Elements of the Reduced net for Example-1 .... 70
6.22 Deducing minimal elements for Example-1 .................. 70
6.23 Example-2 ....................................................... 71
6.24 Input file for Example-2 ......................................... 71
6.25 Output for Reduction part of the algorithm for Example-2 71
6.26 Reduced Example-2 ............................................. 72
6.27 Minimal Elements of the Reduced net for Example-2 ...... 72
6.28 Deducing minimal elements for Example-2 ................... 73
6.29 Example-3 ....................................................... 73
6.30 Input file for Example-3 ......................................... 74
6.31 Output for Reduction part of the algorithm for Example-3 74
6.32 Reduced Example-3 ............................................. 75
6.33 Minimal Elements of the Reduced net for Example-3 ...... 75
6.34 Deducing minimal elements for Example-3 ................... 76
6.35 PN-9 (cf. [5]). .................................................... 77
6.36 Input file for PN-9 (cf. [5]). .................................... 77
6.37 Output for Reduction part of the algorithm for PN-9 (cf. [5]) 78
6.38 Reduced PN-9 (cf. [5]). .......................................... 79
6.39 Minimal Elements of the Reduced net for PN-9 (cf. [5]) .. 80
6.40 Deducing minimal elements for PN-9 (cf. [5]). ............ 81
6.41 PN-13 (cf. [5]). .................................................. 82
6.42 Input file for PN-13 (cf. [5]). ................................... 82
6.43 Output for Reduction part of the algorithm for PN-13 (cf. [5]) 83
6.44 Reduced PN-13 (cf. [5]). ....................................... 84
6.45 Minimal Elements of the Reduced net PN-13 (cf. [5]) ... 85
6.46 Deducing minimal elements for PN-13 (cf. [5]). .......... 86
6.47 PN-11 (cf. [5]). .................................................. 87
6.48 Input file for PN-11 (cf. [5]). ................................... 87
6.49 Output for Reduction part of the algorithm for PN-11 (cf. [5]) 88
6.50 Reduced PN-11 (cf. [5]). ..................................... 89
6.51 Minimal Elements of the Reduced net for PN-11 (cf. [5]) . 90
6.52 Deducing minimal elements for PN-11 (cf. [5]). .......... 91
6.53 Example-5 ........................................................ 93

7.1 A PN \( N_{10}^{m_0_{10}} \) with (a) an event-based LESP (that is not \textit{minimally restrictive}), and (b) A static-map based LESP that is minimally restrictive. ......................... 96
CHAPTER 1

INTRODUCTION

Every windows user is familiar with the screen-shot as shown in Figure 1.1. In this scenario Mozilla Firefox is an unresponsive program that remains in a state of suspended animation for perpetuity. This task is *livelocked*. The only recourse is to terminate the unresponsive program using Windows Task Manager or to reboot the system. This step is followed by sending an error report to the developers which is used to generate next-generation of fixes through a service like Windows update (cf. figure 1.2). Clear understanding of the concept of livelock-avoidance could help avoid such instances.

Forced-shutdown could be a major concern in critical software applications such as health care and avionics. A livelocked application in these areas could have disastrous consequences. Restarting a livelocked task after forcibly terminating it could have dire implications in service systems, where complex service-level-agreements (SLAs) between the service-provider and the clients can result in large compensations owed to a client that is terminated prejudicially in the middle of service. Livelock-avoidance in these systems are clearly a pressing issue, and based on the frequent occurrences of instances like what is shown in figure 1.1, we can say with confidence that to date there is no cogent theory for livelock avoidance.

It is important to work on design principles for livelock-avoidance where some tasks could enter into a state of suspended animation as mentioned above. Currently, these systems are over-designed and scheduled inefficiently which leads to higher costs in *Discrete-Event/Discrete-State* (DEDS) systems. Our research focuses on implementation of a *supervisory policy* for the avoidance of livelocks in *Petri nets* (PN) models of DEDS systems. In this paradigm, a DEDS system is modeled as a PN, where all activities that can occur at a discrete-state are permitted to occur in the PN model. The PN model is not *live*, if it can enter into a livelocked state where some activities can never proceed to completion. The objective is to synthesize a *supervi-
sory policy that determines the set of events that are to be prevented from occurring at each discrete state of the DEDS system such that the resulting supervised PN is live. This is a liveness enforcing supervisory policy (LESP) for the DEDS system.

In the most general setting, the supervisory policy cannot prevent the occurrence of certain events of the DEDS system. For instance, a failure-event, or an event that is an exigency, cannot be prevented from occurring. These external, uncontrollable events are beyond the control of the supervisory policy. The synthesis procedure for an LESP has to hedge against the pernicious influences of these external events in the general setting. An LESP \( \mathcal{P} \) is said to be minimally restrictive if the fact that it prevents the occurrence of an event at a discrete state implies that all other LESPs will also prevent the occurrence of the same event at that discrete state. The existence of an LESP for a PN model of a DEDS system implies the existence of a unique, minimally restrictive LESP.

There are necessary and sufficient conditions for the existence of an LESP for an arbitrary PN model of a DEDS system. Unfortunately, testing the existence of a LESP for DEDS systems that are modeled by arbitrary PN models is undecidable. In fact, neither the existence, nor the non-existence
of a LESP for an arbitrary PN model is even semi-decidable. This means any heuristic procedure for the synthesis of an LESP for arbitrary PN models will hang indefinitely for at least one instance where there is an LESP, and another instance for which there is no LESP. Consequently, we need to restrict attention to specific classes of PN models for DEDS systems for which the existence (and synthesis) of a LESP is decidable. If every activity of the DEDS system can be prevented from occurring by the LESP (i.e. there are no uncontrollable events), or if the PN model that represents the DEDS system is an Ordinary Free Choice PN, there is a synthesis procedure for the minimally restrictive LESP. The relevant details are presented in subsequent chapters.

This thesis covers the identification of two classes of general PN structures, $\mathcal{F}$ and $\mathcal{H}$ described in (cf. chapter 4), for which the existence (and synthesis) of a LESP is decidable (cf. [1, 2]). This is shown by establishing the property that for any PN structure $N$ that belongs to these classes, the existence of a LESP when $N$ is initialized with the marking (i.e. state) $m^0$ implies the existence of a LESP when $N$ is initialized with any (term-wise) larger initial marking.

The software described in reference [5] can be used to synthesize the minimally restrictive LESP for any member of the classes of PNs that meet the monotonicity property referred to in (cf. chapter 3). We have encountered examples where the software of reference [5] takes an unusually long time to compute the minimally restrictive LESP for specific problem instances. We use reduction methods described in (cf. chapter 6) to reduce the computational time for deducing LESP for PNs. The object-oriented implementa-
tion of these techniques is described in (cf. sections 6.1, 6.2 and 6.3). We use illustrative examples to prove the utility of various results that have been obtained. These examples are interspersed in the subsequent chapters of this thesis.
CHAPTER 2
A REVIEW OF PETRI NETS

In this chapter we formally define PN concepts that are pertinent to the development of the results in this thesis. A detailed treatment can be found in Murata’s review article [4], or Peterson’s book [6].

2.1 Notations, definitions and other preliminary observations

The set of non-negative (positive) integers is denoted by $\mathcal{N}$ ($\mathcal{N}^+$). The cardinality of a set $A$ is represented as $\text{card}(A)$. A Petri net structure $N = (\Pi, T, \Phi, \Gamma)$ is an ordered 4-tuple, where $\Pi = \{p_1, \ldots, p_n\}$ is a set of $n$ places, $T = \{t_1, \ldots, t_m\}$ is a collection of $m$ transitions, $\Phi \subseteq (\Pi \times T) \cup (T \times \Pi)$ is a set of arcs, and $\Gamma : \Phi \to \mathcal{N}^+$ is the weight associated with each arc.

In graphical representation of PNs places (resp. transitions) are represented by circles (resp. boxes), and each member of $\phi \in \Phi$ is denoted by a directed arc. If $\phi = (p, t)$ (resp. $(t, p)$) the arc is directed from $p$ (resp. $t$) to $t$ (resp. $p$). The initial marking is represented by an appropriate integer, $m^0(p)$, within each place $p \in \Pi$. The weight of an arc is represented by an integer that is placed alongside the arc. If an arc has a unitary weight, it is not represented in its graphical representation.

The set of all finite-length strings of transitions is represented by $T^*$. For a string of transitions $\sigma \in T^*$, we use $x(\sigma)$ to denote the Parikh vector of $\sigma$. That is, the $i$-th entry, $x_i(\sigma)$, corresponds to the number of occurrences of transition $t_i$ in $\sigma$.

If all arcs of a PN are unitary, it is said to be an ordinary PN, otherwise it is a general PN. The initial marking of a PN structure $N$ is a function $m^0 : \Pi \to \mathcal{N}$, which identifies the number of tokens in each place. A marking $m : \Pi \to \mathcal{N}$ is sometimes represented by an integer-valued vector $m \in \mathcal{N}^n$. 

5
where the $i$-th component $m_i$ represents the token load ($m(p_i)$) of the $i$-th place. The function- and vector-interpretation of the marking is used interchangeably. A Petri net (PN), $N(m^0)$, is a PN structure $N$ together with its initial marking $m^0$.

Let $\bullet x := \{y | (y,x) \in \Phi\}$ and $x^* := \{y | (x,y) \in \Phi\}$. If $\forall p \in \bullet t, m^i(p) \geq \Gamma((p,t))$ for some $t \in T$ and some marking $m^i$, then $t \in T$ is said to be enabled at marking $m^i$. The set of enabled transitions at marking $m^i$ is denoted by the symbol $T_e(N,m^i)$. An enabled transition $t \in T_e(N,m^i)$ can fire, which changes the marking $m^i$ to $m^{i+1}$ according to $m^{i+1}(p) = m^i(p) - \Gamma(p,t) + \Gamma(t,p)$.

A string of transitions $\sigma = t_1 \cdots t_k$, where $t_j \in T(j \in \{1, \ldots, k\})$ is said to be a valid firing string starting from the marking $m^i$, if, (1) the transition $t_1 \in T_e(N,m^i)$, and (2) for $j \in \{1, \ldots, k - 1\}$ the firing of the transition $t_j$ produces a marking $m^{i+j}$ and $t_{j+1} \in T_e(N,m^{i+j})$ is enabled. If $m^{i+k}$ results from the firing of $\sigma \in T^*$ starting from the initial marking $m^i$, we represent it symbolically as $m^i \xrightarrow{\sigma} m^{i+k}$. Given an initial marking $m^0$ the set of reachable markings for $m^0$ denoted by $\mathcal{R}(N,m^0)$, is defined as the set of markings generated by all valid firing strings starting with marking $m^0$ in the PN $N$. A PN $N(m^0)$ is said to be live if

$$\forall t \in T, \forall m^i \in \mathcal{R}(N,m^0), \exists m^j \in \mathcal{R}(N,m^i) \text{ such that } t \in T_e(N,m^j).$$

In the context of a marking being represented as nonnegative integer-valued vector, it is useful to define input matrix $IN$ and output matrix $OUT$ as two $m \times n$ matrices, where

$$IN_{i,j} = \begin{cases} 1 & \text{if } p_i \in \bullet t_j \\ 0 & \text{otherwise} \end{cases} \quad \text{and} \quad OUT_{i,j} = \begin{cases} 1 & \text{if } p_i \in t_j^* \\ 0 & \text{otherwise} \end{cases}$$

The incidence matrix $C$ of the PN $N$ is an $n \times m$ matrix, where $C = OUT - IN$. If $x(\sigma)$ is an $m$-dimensional vector whose $k$-th component corresponds to the number of occurrences of $t_k$ in a valid string $\sigma \in T^*$, and if $m^i \xrightarrow{\sigma} m^{i+j}$, then $m^{i+j} = m^i + Cx(\sigma)$.

A set of markings $\mathcal{M} \subseteq N^n$ is said to be right-closed [7] if $(m^1 \in \mathcal{M}) \land ...
\((m^2 \geq m^1) \Rightarrow (m^2 \in \mathcal{M})\), and is uniquely defined by its finite set of minimal-elements.

A collection of places \(P \subseteq \Pi\) is said to be a siphon (resp. trap) if \((\bullet P) \subseteq (P^\bullet)\) (resp. \((P^\bullet) \subseteq (\bullet P)\)), where \((\bullet P) := \bigcup_{p \in P} (\bullet p)\) and \((P^\bullet) := \bigcup_{p \in P} (p^\bullet)\). A trap (resp. siphon) \(P\), is said to be minimal if \(\nexists \tilde{P} \subset P\), such that \((\bullet \tilde{P}) \subseteq (\bullet P)\) (resp. \((\tilde{P}^\bullet) \subseteq (P^\bullet)\)).

A PN structure \(N = (\Pi, T, \Phi, \Gamma)\) is Free-Choice (FC) if \(\forall p \in \Pi, (\text{card}(p^\bullet) > 1 \Rightarrow \bullet(p^\bullet) = \{p\})\), where \(\text{card}(\bullet)\) denotes the cardinality of the set argument. A PN \(N(m^0)\) where \(N\) is FC, is a Free-Choice Petri net (FCPN). In other words, a PN structure is Free-Choice if and only if an arc from a place to a transition is either the unique output arc from that place, or, is the unique input arc to the transition. Commoner’s Liveness Theorem (cf. chapter 4, [8]; [9]) states an ordinary FCPN \(N(m^0)\) is live if and only if every minimal siphon in \(N\) contains a minimal trap that has a non-empty token load at the initial marking \(m^0\).

Testing the liveness of an ordinary FCPN is NP-hard. Under appropriate conditions, an ordinary FCPN that violates Commoner’s Liveness Theorem can be made live by supervision. If an ordinary FCPN \(N(m^0)\) is live for an initial marking \(m^0\), the ordinary FCPN \(N(\hat{m}^0)\) is also live for any \(\hat{m}^0 \geq m^0\). That is, the class of ordinary FCPNs exhibit liveness monotonicity.

The liveness monotonicity property, and Commoner’s Liveness Theorem, are not satisfied by general FCPNs. As an illustration consider the FCPN structure \(N_1 = (\Pi_1, T_1, \Phi_1, \Gamma_1)\) shown in figure 2.1. Since \(\Gamma_1((p_5, t_6)) = 2\), \(N_1\) is a general FCPN. \(N_1\) has no traps. Since \(\bullet \Pi_1 \subset \Pi_1^\bullet\), \(N_1\) has a siphon that contains no traps. Prima facie, this general FCPN violates Commoner’s Liveness Theorem. However, \(N_1(m_1^0)\) is live if and only if the sum of tokens assigned to all places by the initial marking \(m_1^0\) is an odd number. Neither does \(N_1\) posses the property of liveness monotonicity that is true of ordinary FCPNs.

A PN structure \(N = (\Pi, T, \Phi, \Gamma)\) is said to be a Simple Petri Net (SPN) if and only if

\[\forall t \in T, \text{card}(\{p \in t \mid \text{card}(p^\bullet) > 1\}) \leq 1.\]

The family of FCPN structures is strictly contained in the family of SPN structures. Barkaoui et al. [10] present a sufficient condition for the liveness
of a general SPN $N(m^0)$. They note that if all siphons $P \subseteq \Pi$ of the SPN $N(m^0)$ satisfy the requirement $\forall m \in \mathbb{R}(N, m^0), \exists p \in P$ such that $m(p)$ is greater than or equal to the largest weight among the arcs that originate from $p$, then $N(m^0)$ is live. This sufficient condition is not necessary for liveness. Since the class of SPN structure strictly includes the class of general FCPNs, the general FCPN structure $N_2 = (\Pi_2, T_2, \Phi_2, \Gamma_2)$ shown in figure 2.2 is also an SPN, and $\Pi_2$ is a siphon (and a trap). $N_2(m^0_2)$ is live for any $m^0_2 \neq 0$. Consequently, $N_2(m^0_2)$ is live for $m^0_2 = (1 0 0 0 0)^T$, but $\forall p \in \Pi_2, m^0_2(p)$ is strictly less than the largest weight among all arcs that originate from $p$.

These examples illustrate that (1) Liveness Monotonicity and Commoner’s Liveness Theorem are inapplicable to the class of general FCPNs, and (2) the sufficient condition of Barkaoui et al. [10] are not necessary for liveness of general FCPNs.

We present an important analytical tool for PNs in the following subsection, which can be automatically generated by software tools described in subsequent chapters, additionally this construction plays a crucial role in
several theoretical results in the literature and in this thesis.

2.1.1 Coverability graph

*Reachability/Coverability graph* consists of all possible markings that can be reached when the transitions in a net are fired. When the transitions from an initial marking $m^0$ are fired it gives rise to new markings. From these new markings further markings are reached as transitions are enabled. This leads to a tree structure that could be infinitely large. The procedure listed below can be interpreted as a finite-characterization of this tree structure, which is known as the *reachability tree* (cf. section 4.2.1, [6]). The vertex set of this tree is $V$, and each vertex $v \in V$ has an (extended) marking of the PN, $\mu(v)$, associated with it. An extended marking can be thought of as markings where some places can have infinite tokens. The symbol $\omega$ is used to represent the presence of infinite tokens. Each edge of this tree has a transition associated with it. The tree is constructed using the procedure of figure 2.3.

If the duplicate nodes are merged with the parent node in a reachability graph, we get the *coverability graph*. A PN is unbounded if and only if there are $\omega$ symbols in its coverability graph. The coverability graph is finite for any PN.

Figure 2.4 represents a PN $N_3(m^0_3)$ that is not bounded and not live. The reason $N_3(m^0_3)$ is not bounded is because the number of tokens in $p_1$ can grow without bound with repeated firings of $t_3t_2t_4$. This can also be inferred from the fact that there is at least one vertex $v$ of the coverability graph in Figure 2.5 where $\mu(v)$ assigns the $\omega$-symbol to place $p_1$. This PN is not live since the transition $t_1$ is not fired even once. In general, liveness (resp. boundedness) cannot (resp. can) be inferred from the coverability graph of a PN.
1: The root vertex is $v_0$. $V \leftarrow \{v_0\}$, and $\mu(v_0) = m^0$.

2: \textbf{for} $v_i \in V$ \textbf{do}
3: \hspace{1em} if $\mu(v_i)$ is identical to $\mu(v_j)$ for some $v_j \in V$ \textbf{then}
4: \hspace{2em} $v_i$ has no children, and is marked as the \textit{duplicate} of $v_j$.
5: \hspace{1em} \textbf{end if}
6: \hspace{1em} if no transition is enabled under the marking $\mu(v_i)$ \textbf{then}
7: \hspace{2em} $v_i$ has no children, and is marked as a \textit{terminal} vertex.
8: \hspace{1em} \textbf{end if}
9: \hspace{1em} if $v_i$ is not a duplicate-vertex \textbf{then}
10: \hspace{2em} \textbf{for} $t_j$ that is enabled under $\mu(v_i)$ \textbf{do}
11: \hspace{3em} Create a new vertex $v_k$. $V \leftarrow V \cup \{v_k\}$.
12: \hspace{3em} Create a new directed edge starting from $v_i$ and ending at $v_k$. Label this edge with the transition $t_j$.
13: \hspace{3em} if The number of tokens in $p$ is $\omega$ under $\mu(v_i)$, for some $p \in \Pi$ \textbf{then}
14: \hspace{4em} The number of tokens in $p$ is $\omega$ under $\mu(v_k)$ too.
15: \hspace{3em} \textbf{else}
16: \hspace{4em} The number of tokens in $p$ under $\mu(v_k)$ is what results when $t_j$ is fired under $\mu(v_k)$.
17: \hspace{3em} \textbf{end if}
18: \hspace{3em} if ($\exists v_q \in V$ on the directed path from $v_0$ to $v_k$ such that $\mu(v_q) \leq \mu(v_k)$) \textbf{then}
19: \hspace{4em} \textbf{for} ($p \in \Pi$) \textbf{do}
20: \hspace{5em} if $p$ has fewer tokens under $\mu(v_q)$ than under $\mu(v_k)$ \textbf{then}
21: \hspace{6em} The number of tokens in $p$ is $\omega$ under $\mu(v_k)$.
22: \hspace{5em} \textbf{end if}
23: \hspace{4em} \textbf{end for}
24: \hspace{3em} \textbf{end if}
25: \hspace{3em} \textbf{end for}
26: \hspace{1em} \textbf{end if}
27: \textbf{end for}

Figure 2.3: The procedure for the construction of the \textit{Reachability Tree} of a PN $N(m^0)$, where $N = (\Pi, T, \Phi, \Gamma)$.
Figure 2.5: Coverability graph for the PN $N_3(m_3^0)$ that is not bounded and not live (cf. section V.C, [4]).

Figure 2.4: A PN $N_3(m_3^0)$ that is not bounded and not live (cf. section V.C, [4]).

To reiterate, while the reachable set of markings of a PN can be infinitely large, its coverability graph is always finite. The PN $N_4(m_4^0)$ in Figure 2.6 has an infinite set of markings that can be reached from the initial marking of $(1\ 0\ 0\ 0)^T$. From its finite coverability graph, shown in Figure 2.7, we can infer that the token load of every place in this PN can grow without bound.

It should be noted that the number of vertices in the coverability graph of a PN can be prohibitively large. Oftentimes, this is the reason behind the ineffectiveness of coverability graph based methods in the analysis of PNs. However, coverability graphs can be very effective in establishing novel decidability results. Alternately, any effective method for reducing the size of the coverability graph, while retaining features that are pertinent to a specific problem, can improve the viability of coverability graph based methods for the analysis of PNs.
however, they do not contain any traps. Hence at some point this petri net will cease to be live.

4. Right closed set - A set of markings $\Omega$ is right-closed if $m_1 \in \Omega \Rightarrow m_2 \in \Omega$ for all $m_2 \geq m_1$. That is, if a marking is in the set, then all larger markings are also in the set. Right-closed sets are uniquely defined by its finite set of minimal elements. For controllable petri nets, a supervisory policy that enforces livelock freedom (if it exists) is characterized by an appropriately selected right-closed set \[3\]. The policy prevents the occurrence of any transition at a marking if its firing will result in a new marking that is not in the right-closed set.

For Figure 1.4 the set of minimal elements are \[
\{(0 0 0 1), (0 0 1 0), (0 1 0 0), (1 0 0 0)\}.
\]

The supervisory policy of this petri net would prevent the firing of the transition $t_5$ at the marking $(0 0 0 1)$. This is because the firing $t_5$ at $(0 0 0 1)$ would result in the marking $(0 0 0 0)$, which is not in the right-closed set defined by the minimal elements $(0 0 0 1), (0 0 1 0), (0 1 0 0), (1 0 0 0)$.

Figure 2.6: A PN $N_4(m_4^0)$ with infinitely large set of markings

Figure 2.7: Finite coverability graph for $N_4(m_4^0)$ of figure 2.6 with infinitely large set of markings
In this chapter we present the paradigm of supervisory control of PNs, and review the results that are relevant to the topics covered in this thesis.

3.1 Supervisory Policies for Liveness Enforcement

This paradigm of marking-based supervisory control assumes a subset of controllable transitions, denoted by $T_c \subseteq T$, can be prevented from firing by an external agent called the supervisor. The set of uncontrollable transitions, denoted by $T_u \subseteq T$, is given by $T_u = T - T_c$. The controllable (resp. uncontrollable) transitions are represented as filled (resp. unfilled) boxes in graphical representation of PNs.

A supervisory policy $\mathcal{P} : \mathcal{N}^n \times T \rightarrow \{0, 1\}$, is a function that returns a 0 or 1 for each transition and each reachable marking. The supervisory policy $\mathcal{P}$ permits the firing of transition $t_j$ at marking $m_i$, only if $\mathcal{P}(m_i, t_j) = 1$. A policy $\mathcal{P}$ is marking monotone if $\forall t_i \in T, \forall m^2 \geq m^1, (\mathcal{P}(m^1, t_i) = 1) \Rightarrow (\mathcal{P}(m^2, t_i) = 1)$.

If $t_j \in T_e(N, m^i)$ for some marking $m^i$, we say the transition $t_j$ is state-enabled at $m^i$. If $\mathcal{P}(m^i, t_j) = 1$, we say the transition $t_j$ is control-enabled at $m^i$. A transition has to be state- and control-enabled before it can fire. The fact that uncontrollable transitions cannot be prevented from firing by the supervisory policy is captured by the requirement that $\forall m^i \in \mathcal{N}^n, \mathcal{P}(m^i, t_j) = 1$, if $t_j \in T_u$. This is implicitly assumed of any supervisory policy in this paper.

A string of transitions $\sigma = t_1 \cdots t_k$, where $t_j \in T (j \in \{1, \ldots, k\})$ is said to be a valid firing string starting from the marking $m^i$, if, (1) $t_1 \in T_e(N, m^i), \mathcal{P}(m^i, t_1) = 1$, and (2) for $j \in \{1, \ldots, k-1\}$ the firing of the transition $t_j$ produces a marking $m^{i+j}$ and $t_{j+1} \in T_e(N, m^{i+j})$ and $\mathcal{P}(m^{i+j}, t_{j+1}) = 1$. 
The set of reachable markings under the supervision of $P$ in $N$ from the initial marking $m^0$ is denoted by $\mathcal{R}(N, m^0, P)$. We use the symbol $\mathcal{R}(N, m^0)$ to denote the set of reachable markings when the PN $N(m^0)$ is unsupervised, or when the supervisory policy $P$ is a trivial policy that permits every transition at all markings (as would be the case when $T = T_u$, for instance).

If $m^i \xrightarrow{\sigma} m^j$, for some $\sigma \in T^*$, we have $m^j = m^i + Cx(\sigma)$, where $C$ is the incidence matrix of $N$, and $x(\sigma)$ is the Parikh mapping of $\sigma$. We say $m^i$ is potentially reachable from $m^0$ if $\exists y \in N^m$ that satisfies the equation $Cy = (m^i - m^0)$. If $m^i$ is not potentially reachable from $m^0$, we can conclude that $m^i \notin \mathcal{R}(N, m^0, P)$ for any $P$.

For a marking monotone supervisory policy $P$, the construction procedure for the coverability graph of a PN (cf. section 2.1.1) can be extended to accommodate the supervisory policy $P$, which results in the coverability graph $G(N(m^0, P))$ (cf. figure 1, [11]). We use the symbol $v_1 \xrightarrow{\sigma} v_2$ to denote the path labeled by $\sigma \in T^*$ from vertex $v_1$ to vertex $v_2$ in $G(N(m^0, P))$.

A transition $t_k$ is live under the supervision of $P$ if $\forall m^i \in \mathcal{R}(N, m^0, P)$,

$$\exists m^j \in \mathcal{R}(N, m^i, P) \text{ such that } t_k \in T_e(N, m^j) \text{ and } P(m^j, t_k) = 1.$$  

A policy $P$ is a liveness enforcing supervisory policy (LESP) for $N(m^0)$ if all transitions in $N(m^0)$ are live under $P$. The policy $P$ is said to be minimally restrictive if for every LESP $\hat{P} : N^m \times T \rightarrow \{0, 1\}$ for $N(m^0)$, the following condition holds

$$\forall m^i \in N^m, \forall t \in T, P(m^i, t) \geq \hat{P}(m^i, t).$$

The existence of an LESP for an arbitrary PN is undecidable (cf. Corollary 5.2, [12]). Additionally, neither the existence nor the non-existence of an LESP for an arbitrary PN is semi-decidable (cf. Theorems 3.1 and 3.2, [11]). Therefore, any heuristic procedure that attempts to find an LESP for an arbitrary PN will hang indefinitely for at least one instance where there is an LESP, and another instance for which there is no LESP.

If there is an LESP for some $N(m^0)$, then there is a unique minimally restrictive LESP for PN $N(m^0)$ (cf. theorem 6.1, [12]). The minimally restrictive LESP can be synthesized for:
1. The class of arbitrary PNs where \( T = T_c \), that is, all transitions in the PN are controllable (cf. Corollary 5.1, [12]);

2. The class of Ordinary FCPNs (cf. Theorem 5.16, [11]);

3. The class of general FCPNs denoted by \( \mathcal{F} \), which strictly includes the class of ordinary FCPNs (Theorem 3.5, [1]);

4. The class of ordinary PNs denoted by \( \mathcal{G} \), which also strictly includes the class of ordinary FCPNs, but is incomparable to the class \( \mathcal{F} \) referred to above (cf. Theorem 3.5, [13]); and,

5. The class of general PNs denoted by \( \mathcal{H} \), which strictly includes classes \( \mathcal{G} \) and \( \mathcal{F} \) introduced earlier (cf. Section 3, [2]).

The minimally restrictive LESP, when it exists for any instance of these classes, is marking monotone.

The set of initial markings, \( \Delta(N) \), for which there is a supervisory policy that enforces liveness for a PN structure \( N \), is defined as

\[
\Delta(N) = \{ m^0 \mid \exists \text{ an liveness enforcing supervisory policy for } N(m^0) \}.
\]

For any PN structure \( N \) that belongs to the five classes identified above, the set \( \Delta(N) \) is right-closed, and is characterized by its minimal elements \( \min(\Delta(N)) \).

A set of markings \( \mathcal{M} \subseteq \mathcal{N}^n \) is said to be control-invariant with respect to a partially controlled PN structure \( N = (\Pi, T, \Phi, \Gamma) \), if \( \mathcal{M} = \Gamma(\mathcal{M}) \), where \( \Gamma(\mathcal{M}) = \{ m^i \in \mathcal{N}^n \mid \exists \sigma \in T_u, \exists m^j \in \mathcal{M}, \text{ such that } m^j \xrightarrow{\sigma} m^i \} \). Note, \( \mathcal{M} \subseteq \Gamma(\mathcal{M}) \) in general. Alternately, if \( \mathcal{M} \) is control-invariant with respect to \( N, m^i \in \mathcal{M}, m^i \xrightarrow{\sigma} m^j \) in \( N \), and \( m^j \notin \mathcal{M} \), then there must be at least one controllable transition in the firing string \( \sigma \in T^* \). There is a procedure to test the control-invariance of a right-closed set of markings \( \mathcal{M} \) with respect to a PN structure \( N \) (cf. Lemma 5.10, [11]). If \( \mathcal{M} \) does not pass this test, then it is possible to find the largest subset of \( \mathcal{M} \) that is control invariant with respect to \( N \).

The set \( \Delta(N) \) is control invariant with respect to the PN structure \( N \). That is, if \( m^1 \in \Delta(N), t_u \in T_e(N, m^1) \cap T_u \) and \( m^1 \xrightarrow{t_u} m^2 \) in \( N \), then \( m^2 \in \Delta(N) \). Alternately, only the firing of a controllable transition at a marking in \( \Delta(N) \) can result in a new marking that is not in \( \Delta(N) \).
Suppose \( m^0 \in \Delta(N) \), then the supervisory policy that control-disables any (controllable) transition at a marking in \( \Delta(N) \) if its firing would result in a new marking that is not in \( \Delta(N) \), is the minimally restrictive LESP for \( N(m^0) \). This lends itself to an effective procedure for the synthesis of a minimally restrictive LESP when it exists, which is elaborated below.

Suppose, (1) \( N \) is a PN structure where \( \Delta(N) \) is known to be right-closed, (2) \( \Psi \) is a right-closed set of markings that is control invariant with respect to \( N \), (3) \( \mathcal{P}_\Psi \) is a supervisory policy that control-disables any (controllable) transition at a marking in \( \Psi \) if its firing would result in a new marking that is not in \( \Psi \), and (4) \( m^0 \in \Psi \), we can construct the coverability graph, \( G(N(m^0), \mathcal{P}_\Psi) \), of \( N(m^0) \) under the supervision of \( \mathcal{P}_\Psi \), along the same lines as the coverability graph of a PN (cf. section 4.2.1, [6]). The policy \( \mathcal{P}_\Psi \) enforces liveness in \( N(m^0) \) if and only if

1. \( m^0 \in \Psi \), and

2. (Path-requirement) there is a vertex \( v \), and a closed-path \( v \xrightarrow{\sigma} v \) in \( G(N(m^0), \mathcal{P}_\Psi) \) \((\sigma \in T^*)\), for each \( m^i \in \min(\Psi) \) where

   (a) all transitions appear at least once in \( \sigma \) (i.e. \( x(\sigma) \geq 1 \)), and

   (b) the net-change in the token-load in each place after the firing of \( \sigma \) is non-negative (i.e. \( Cx(\sigma) \geq 0 \)).

The above test can be represented as a feasibility problem for an appropriately posed instance of an Integer Linear Program (ILP) on the coverability graph \( G(N(m^0), \mathcal{P}_\Psi) \) (cf. appendix, [12]).

The algorithm for the synthesis of a liveness enforcing supervisory policy for a PN structure \( N \) that belongs to a class where \( \Delta(N) \) is known to be right-closed essentially involves a search for a right-closed set of markings \( \Psi \) that is control invariant with respect to \( N \), where each member of \( \min(\Psi) \) meets the path-requirement on its coverability graph described above. This is done in an iterative manner starting with an initial set

\[
\Psi_0 = \{ m^0 \mid \exists \text{ an LESP for } N(m^0) \text{ if all transitions in } N \text{ are controllable} \}
\]

which is known to be right-closed (cf. corollary 5.1, [12]; equation 3, [11]). If \( \Psi \) is the largest subset that meets the aforementioned requirements, and \( m^0 \in \)}
Ψ, then the minimal elements, \( \text{min}(\Psi) \), effectively represent the minimally restrictive LESP for the PN \( N(m^0) \).

The LESP synthesis procedure is described in figure 3.1.

1: \textbf{if} \( m^0 \not\in \Psi_i \) \textbf{then}
2: The procedure terminates with the conclusion that there is no LESP for \( N(m^0) \).
3: \textbf{else if} \( m^0 \in \Psi_i \), and \( \Psi_i \) is not control invariant with respect to \( N \) \textbf{then}
4: \( \Psi_i \) is replaced by its largest control invariant subset, \( \Psi_{i+1} \) where \( \Psi_{i+1} \subset \Psi_i \). Following this, the process is repeated with \( \Psi_i \leftarrow \Psi_{i+1} \) (i.e. go to step 1).
5: \textbf{else}
6: Each minimal element of the control invariant, right-closed set \( \Psi_i \) is tested for the path-requirement on its coverability graph described above.
7: \textbf{if} If all minimal elements satisfy this requirement \textbf{then}
8: The members of \( \text{min}(\Psi_i) \) are presented as a description of the LESP for \( N(m^0) \).
9: \textbf{else}
10: Each minimal element \( m^i \) that fails the requirement is “elevated” by \( \text{card}(\Pi) \)-many unit-vectors as follows
   \[
   m^i \leftarrow \{m^i + 1_i \mid i \in \{1, 2, \ldots, \text{card}(\Pi)\}\}
   \]
   where \( 1_i \) is the \( i \)-th unit-vector. That is, the above process replaces the minimal element \( m^i \) with \( \text{card}(\Pi) \)-many minimal elements, which in turn defines a right-closed set \( \Psi_{i+1} \subset \Psi_i \).
11: \( \Psi_i \leftarrow \Psi_{i+1} \), and go to step 1.
12: \textbf{end if}
13: \textbf{end if}

Figure 3.1: The procedure for testing the existence of an LESP for a PN \( N(m^0) \), where \( N = (\Pi, T, \Phi, \Gamma) \), assuming \( \Delta(N) \) is right-closed.

This procedure forms the corpus of the algorithm used to synthesize the minimally restrictive liveness enforcing supervisory policy for \( N(m^0) \), when it exists, for a structure \( N \) for which it is known that \( \Delta(N) \) is right-closed. This procedure has been implemented in C/C++ on Mac (Windows) platforms using the Xcode (Visual Studio 2012) compiler [14, 15, 16].
3.1.1 LESP Synthesis via an Illustrative Example

The FCPN $N_5(m_5^0)$ shown in figure 3.2 is used to illustrate the LESP synthesis procedure of figure 3.1 of the previous subsection. The output generated by the software of reference [5] is shown in figure 3.3.

Figure 3.2: An FCPN $N_5(m_5^0)$ where $N_5 = (\Pi_5, T_5, \Phi_5, \Gamma_5)$.

Figure 3.3: The output file generated by the software described in reference [5] for the FCPN shown in figure 3.2.

The iteration starts with $\Psi_0$, the largest controllable, right-closed subset of the set of initial markings for which there is an LESP for the fully-controlled
version of $N_5$. In the context of this example, eight minimal elements identify the right-closed of initial markings for which there is an LESP for the fully-controllable version of $N_5$ shown in figure ??.

The second and third among this list of eight minimal elements are not control invariant as $t_5, t_7 \in T_u$ and $(0 0 1 0 0 0 0 0 0)^T \xrightarrow{t_5} (0 0 0 1 0 0 0 0 0)^T \xrightarrow{t_7} (0 0 0 0 0 0 0 0 0)^T$. The largest controllable subset of this right-closed set is $\Psi_0$, which is identified by the six minimal elements shown immediately afterwards in the same figure. That is, $\min(\Psi_0)$ are the six vectors listed below.

1 : $(1 0 0 0 0 0 0 0 0)^T$
2 : $(0 0 0 0 0 0 1 0 0)^T$
3 : $(0 0 0 0 0 0 0 1 0)^T$
4 : $(0 0 0 0 0 0 0 0 1)^T$
5 : $(0 0 0 1 1 0 0 0 0)^T$
6 : $(0 1 0 1 0 0 0 0 0)^T$,

and $m_3^0 \in \Psi_0$.

Each of these six minimal elements are tested for the path-requirement on its coverability graph in line 6 of the procedure. Four minimal elements,

$(1 0 0 0 0 0 0 0 0)^T, (0 0 0 0 0 0 1 0 0)^T, (0 0 0 0 0 0 0 1 0)^T, \text{ and } (0 0 0 0 0 0 0 0 1)^T$,

fail this test. The path-requirement is violated for the first minimal element $(1 0 0 0 0 0 0 0 0)^T \in \min(\Psi_0)$, as $T_e(N_5, (1 0 0 0 0 0 0 0 0)^T) = \{t_1, t_3\} (\subseteq T_c)$.

But,

$(1 0 0 0 0 0 0 0 0)^T \xrightarrow{t_3} (0 1 0 0 0 0 0 0 0)^T$,

and

$(1 0 0 0 0 0 0 0 0)^T \xrightarrow{t_5} (0 0 1 1 0 0 0 0 0)^T$.

Since, $(0 1 0 0 0 0 0 0 0)^T, (0 0 1 1 0 0 0 0 0)^T \notin \Psi_0$, the supervisory policy $P_{\Psi_0}$ would disable these transitions at the marking $(1 0 0 0 0 0 0 0 0)^T$, which effectively creates a policy-induced deadlock state. The requirement is violated for the second, third and fourth minimal elements

$(0 0 0 0 0 0 1 0 0)^T, (0 0 0 0 0 0 0 1 0)^T, (0 0 0 0 0 0 0 0 1)^T \in \min(\Psi_0)$,
as the marking $(1 0 0 0 0 0 0 0 0)^T$ is inevitably reached after the firing of an appropriate set of transitions. Specifically,

$$(0 0 0 0 0 1 0 0)^T \xrightarrow{t_9 t_{11}} (1 0 0 0 0 0 0 0 0)^T,$$

$$(0 0 0 0 0 0 1 0)^T \xrightarrow{t_{10} t_{11}} (1 0 0 0 0 0 0 0 0)^T,$$

and

$$(0 0 0 0 0 0 0 1)^T \xrightarrow{t_{11}} (1 0 0 0 0 0 0 0 0)^T.$$

Since the marking $(1 0 0 0 0 0 0 0 0)^T$ failed the path-requirement, it follows that these three marking would fail the requirement, as well.

The four minimal elements, that failed the path-requirement, are elevated by nine unit vectors, and the largest controllable, right-closed set of this newly constructed set is identified by the twenty-four minimal elements shown in figure 3.3, which identifies the next iterate $\Psi_1$. Each of these twenty-four minimal elements pass the loop-test referred to earlier, implying that $\Delta(N_5) = \Psi_1$. This is effectively identifies the minimally restrictive LESP for $N_5(m_3^6)$.

3.1.2 Review of Relevant Prior Work

We present a brief review of results that are pertinent to the approach used in this paper. Giua [17] introduced monitors into supervisory control of PNs. Monitors are external places added to an existing PN structure whose token load at any instant indicates the amount of a particular resource that is available for consumption. Moody and Antsaklis [18] used monitors to enforce liveness in certain classes of PNs, this work was extended by Iordache and Antsaklis [19] to include a sufficient condition for the existence of policies that enforce liveness in a class of PNs called Asymmetric Choice Petri nets. Reveliotis et al. used the theory of regions to identify policies that enforce liveness in Resource Allocation Systems [20]. Ghaffari, Rezg and Xie [21] also used the theory of regions to obtain a minimally restrictive supervisory policy that enforces liveness for a class of PNs. Liu et al. [22] characterized the set of live initial markings of a class of general PN structures known as $WS^3PR$, which was used to construct monitors that enforce liveness in a class of $WS^3PR$. Marchetti and Munier-Kordon [23] presented a sufficient condition for liveness, that can be tested in polynomial time, for a class of general PNs known as Unitary Weighted Event Graphs. Basile et al. [24] presented
sufficient conditions for minimally-restrictive, closed-loop liveness of a class of Marked Graph PNs supervised by monitors that enforce Generalized Mutual Exclusion Constraints (GMECs).
In this chapter we present a review of the theoretical results in references [1, 2]. Some of the results in these references are about identifying families of PN structures for which the $\Delta(N)$-set is right-closed. These results increase the purview of the software tool described in reference [5]. That is, this software tool can be used, with no modifications, to any instance of each of these classes. This chapter also highlights other theoretical results that improve the running-time of the software tool.

4.1 $\mathcal{F}$ class PNs [1]

The right-closed nature of the $\Delta(N)$-set is crucial to deciding the existence of a LESP for a PN $N(m_0)$. That is, if there is a LESP for an initial marking then there is a liveness enforcing supervisory policy for the same PN structure for any larger initial marking. This property is established by construction.

The $\Delta(N)$-set for a general PN is not necessarily right-closed. Consider the PN structure $N_1$ in Figure 2.1. All the transitions of this PN structure are uncontrollable and the arc $(p_5, t_6)$ has weight 2. There is an LESP for $N_1(m_0^1)$ if and only if it is live, and $N_1(m_0^1)$ is live if and only if the sum of the tokens assigned to the places by $m_0^1$ is odd. Therefore, $\Delta(N_1)$ is the set of initial markings whose sum is an odd number. This set of initial markings is not a right-closed set.

In reference [1] we identified a class $\mathcal{F}$, of general FCPN structures, for which the set of initial markings that enforce liveness is right-closed. Consequently, the existence of supervisory policy in $\mathcal{F}$ class nets is decidable. The class of Ordinary FCPNs is strictly contained in the $\mathcal{F}$ class. In a broad sense, the results of reference [11] stated originally in the context of Ordinary FCPNs, apply with appropriate modifications, to the class $\mathcal{F}$, as well. This
would mean that the software of reference [5] can be used for any instance of $\mathcal{F}$.

Each member of $\mathcal{F}$ is identified by the following property –

if a place has multiple output transitions, at least one of which is uncontrollable, then the weight(s) associated with the arc(s) that originate from the place at hand, to each uncontrollable transition, must be the smallest of all outgoing arc weights from the place.

More specifically, a general PN structure $N = (\Pi, T, \Phi, \Gamma)$ is said to belong to the class $\mathcal{F}$ if,

1. $N$ is an FCPN structure, and
2. $\forall p \in \Pi$,

$$((p^\bullet \cap T_u \neq \emptyset) \land (\text{card}(p^\bullet) > 1)) \Rightarrow \left( \forall t_u \in p^\bullet \cap T_u, \Gamma(p, t_u) = \min_{t \in T_p^\bullet} \Gamma((p, t)) \right).$$

Consider Figure 4.1, the PN $N_6$ has two outgoing arcs from $p_2$ with weight 2 each. The uncontrollable transition has the smallest outgoing arc weight as a result of which this PN belongs to $\mathcal{F}$ class. The supervisory policy that enforces liveness in $N_6$ is the right-closed set with minimal elements $\{(2 \ 0)^T, (0 \ 2)^T\}$. The minimally restrictive supervisory policy that enforces liveness prevents the firing of transition $t_3$ at a marking, if the new marking that would result from its firing is not in the right-closed set with minimal elements $\{(2 \ 0)^T, (0 \ 2)^T\}$.

![Figure 4.1: General FCPN structure $N_6$](image)

The general FCPN structure $N_7$ in Figure 4.2 and $N_8$ in Figure 4.3 belong to $\mathcal{F}$ class.
\( N_7 \) belongs to \( \mathcal{F} \) as every outgoing arc from place \( p_1 \) or \( p_2 \) has a weight of two. There is a liveness enforcing policy for any initial marking of this structure that belongs to the right-closed set whose minimal elements are \( \{(0010)^T,(2000)^T,(1001)^T,(0200)^T,(0002)^T\} \). The minimally restrictive supervisory policy that enforces liveness will disable the controllable transition \( t_5 \) at a marking if its firing would result in a new marking that is not in this right-closed set.

\( N_8 \) belongs to the class \( \mathcal{F} \) as the outgoing arcs from \( p_3 \) to uncontrollable transitions \( t_5 \) and \( t_6 \) have a weight of unity; and the weight of the outgoing arc from \( p_1 \) to the uncontrollable transition \( t_2 \) is the smallest of all weights associated with arcs that originate from \( p_1 \). There is a liveness enforcing policy for any initial marking of this structure that belongs to the right-closed set whose minimal elements are \( \{(10000)^T,(00011)^T\} \).

The following result, which parallels lemma 5.1 in reference [11], notes that if \( N \in \mathcal{F} \), and if a few extra uncontrollable transitions were to fire in \( N(\hat{m}^0) \) compared to \( N(m^0) \) where \( \hat{m}^0 \geq m^0 \), then it is always possible to extend the firing strings in \( N(m^0) \) and \( N(\hat{m}^0) \) in such a way the Parikh vectors of the extended firing strings are identical, provided there is an LESP for \( N(m^0) \).

**Lemma 4.1.1.** Let \( \mathcal{P} : \mathcal{N}^n \times T \to \{0,1\} \) be an LESP for a general FCPN \( N(m^0) \), where \( N = (\Pi,T,\Phi,\Gamma) \), \( T = T_c \cup T_u \), and \( N \in \mathcal{F} \). Suppose \( m^0 \overset{\sigma}{\rightarrow} m^i \) under the supervision of \( \mathcal{P} \) in \( N \), and \( \hat{m}^0 \overset{\hat{\sigma}}{\rightarrow} \hat{m}^j \) in the absence of any supervision in \( N \) for some \( \hat{m}^0 \geq m^0 \). Further, let us suppose that the number of
occurrences of each controllable transition in $\hat{\sigma}$ and $\sigma$ are identical; however, the string $\hat{\sigma}$ has a few more uncontrollable transitions than the string $\sigma$. That is, $\{t_j \in T | x(\hat{\sigma})_j > x(\sigma)_j\} \subseteq T_u$. Then $\exists \tilde{\sigma}_1, \tilde{\sigma}_2 \in T^*$, such that

1. $\hat{m}^j \xrightarrow{\tilde{\sigma}_1} \hat{m}^k$ in $N$ in the absence of any supervision,

2. $m^i \xrightarrow{\tilde{\sigma}_2} m^l$ under the supervision of $P$ in $N$, and

3. $x(\tilde{\sigma}\tilde{\sigma}_1) = x(\sigma\tilde{\sigma}_2) \Rightarrow \hat{m}^k \geq m^l$).

That is, $m^0 \xrightarrow{\sigma_2} m^l$ under the supervision of $P$, and $\hat{m}^0 \xrightarrow{\tilde{\sigma}_3} \hat{m}^k$ in the absence of any supervision in $N$. If $\hat{m}^0 \geq m^0$ and $x(\sigma\tilde{\sigma}_2) = x(\tilde{\sigma}\tilde{\sigma}_1)$, then $\hat{m}^k \geq m^l$.

Proof. Since $P$ enforces liveness in $N(m^0)$, we can pick a string $\sigma_1 \in T^*$ such that

1. $m^i \xrightarrow{\sigma_1} m^{i+1}$ under $P$ in $N$,

2. $\forall \sigma_1 \in pr(\sigma_1) - \{\sigma_1\}$, if $m^i \xrightarrow{\sigma_1} m^{i+1}$, then $T_e(N, m^{i+1}) \cap \{t_j \in T | x(\hat{\sigma})_j > x(\sigma)_j\} = \emptyset$, where $pr(\bullet)$ is the prefix-set of the string argument, and

3. $\{t_j \in T | x(\hat{\sigma})_j > x(\sigma)_j\} \cap T_e(N, m^{i+1}) \neq \emptyset$.

That is, none of the transitions in the set $\{t_j \in T | x(\hat{\sigma})_j > x(\sigma)_j\} \subseteq T_u$ are state-enabled (and trivially control-enabled) following the firing of any proper prefix of the firing string $\sigma_1$. Additionally, at least one member of the
set \( \{t_j \in T \mid \mathbf{x}({\hat{\sigma}})_j > \mathbf{x}(\sigma)_j \} \) is state-enabled (and trivially control-enabled) at the marking \( \mathbf{m}^i+1 \) that results from the firing of the string \( \sigma_t \) at \( \mathbf{m}^i \).

It follows that \( \mathbf{m}^i \xrightarrow{\sigma} \mathbf{m}^{i+1} \) in the absence of any supervision in \( N \), which can be established by contradiction. Suppose \( \sigma_1 = \tilde{t}_1 \cdots \tilde{t}_i \tilde{t}_{i+1} \cdots \), and \( \mathbf{m}^i \xrightarrow{\tilde{t}_1 \cdots \tilde{t}_i} \mathbf{m}^{i+2} \), but \( \tilde{t}_{i+1} \notin T_e(N, \mathbf{m}^{i+2}) \). This must be due to the reduction in the number of tokens in an input place of \( \tilde{t}_{i+1} \), as a result of the firing of some transition \( t_u \in \{t_j \in T \mid \mathbf{x}({\hat{\sigma}})_j > \mathbf{x}(\sigma)_j \} \subseteq T_u \). Since \( N \) is an FCPN structure, transitions \( t_u \) and \( \tilde{t}_{i+1} \) must share a unique input place (i.e. \( t_u \cap t_{i+1} = \{p\} \) for some \( p \in \Pi \)). Since \( N \in \mathcal{F} \), whenever \( p \) has sufficient tokens to state-enable \( \tilde{t}_{i+1} \), it follows that \( t_u \) is also state-enabled at the same marking. This contradicts the second of three conditions required of \( \sigma_1 \).

If \( t_u \in \{t_j \in T \mid \mathbf{x}({\hat{\sigma}})_j > \mathbf{x}(\sigma)_j \} \cap T_u(N, \mathbf{m}^{i+1}) \), then \( \mathbf{m}^0 \xrightarrow{\sigma_1} \mathbf{m}^{i+1} \xrightarrow{t_u} \mathbf{m}^{i+2} \) under \( \mathcal{P} \) in \( N \). As noted above, \( \mathbf{m}^0 \xrightarrow{\sigma_1} \mathbf{m}^{i+1} \) in the absence of any supervision in \( N \). Additionally, \( \{t_j \in T \mid \mathbf{x}({\hat{\sigma}})_{\sigma_1t_u} > \mathbf{x}(\sigma)_{\sigma_1t_u} \} \subset \{t_j \in T \mid \mathbf{x}({\hat{\sigma}})_j > \mathbf{x}(\sigma)_j \} \). The claim is established by replacing \( \sigma \) with \( \sigma_1t_u \), and \( \hat{\sigma} \) with \( \hat{\sigma}_1 \) in the above argument as often as necessary. Since the cardinality of the set \( \{t_j \in T \mid \mathbf{x}({\hat{\sigma}})_j > \mathbf{x}(\sigma)_j \} \) decreases with each repetition, the process is guaranteed to terminate, which establishes the result. \( \square \)

Following reference [11], we construct a supervisory policy \( \hat{\mathcal{P}} : \mathcal{N}^{\text{card}(\Pi)} \times T \rightarrow \{0, 1\} \) from the supervisory policy \( \mathcal{P} \) for \( N(\mathbf{m}^0) \) as follows –

1. \( \forall t \in T, \hat{\mathcal{P}}(\hat{\mathbf{m}}^0, t) = \mathcal{P}(\mathbf{m}^0, t) \).

2. Suppose \( \hat{\mathbf{m}}^0 \xrightarrow{\hat{\sigma}} \hat{\mathbf{m}}^i \) in \( N \) under the supervision of \( \hat{\mathcal{P}} \),
   (a) \( \forall t_i \in T_u, \hat{\mathcal{P}}(\hat{\mathbf{m}}^i, t_i) = 1 \).
   (b) \( \forall t_i \in T_c, (\hat{\mathcal{P}}(\hat{\mathbf{m}}^i, t_i) = 1) \iff \)
   i. \( t_i \in T_u(N, \hat{\mathbf{m}}^i) \), and
   ii. \( \exists \sigma \in T^*, \) such that
      A. \( \mathbf{m}^0 \xrightarrow{\gamma} \mathbf{m}^k \) under the supervision of \( \mathcal{P} \) in \( N \),
      B. \( \forall k \in \{1, 2, \ldots, m\}, \mathbf{x}(\hat{\sigma}t_i)_k \geq \mathbf{x}(\sigma)_k \), and
      C. \( \{t_j \in T \mid \mathbf{x}({\hat{\sigma}})_{t_i} > \mathbf{x}(\sigma)_{t_i} \} \subseteq T_u \).

That is, the policy \( \hat{\mathcal{P}} \) control-enables a controllable transition at a marking only if its firing is essential to achieving condition 3 articulated in the statement of lemma 4.1.1. Consequently, the supervisory policy \( \hat{\mathcal{P}} \) is an LESP for
Theorem 4.1.2. Let $N \in \mathcal{F}$, where $N = (\Pi, T, \Phi, \Gamma)$ is a general FCPN structure, then the set $\Delta(N)$ is right-closed.

Consequently, the software described in references [14, 15, 16, 5] can be used to compute the minimal element of $\Delta(N)$ for any $N \in \mathcal{F}$. This in turn describes the minimally restrictive LESP for $N(m^0)$ for any $m^0 \in \Delta(N)$. As an illustration, the minimally restrictive LESP for $N_6(m^0_6)$ for any $m^0_6 \in \Delta(N_6)$, where $N_6$ is the PN structure of figure 4.1, is shown in figure 4.4. Similarly, figure 4.5 shows the minimally restrictive LESP for $N_7(m^0_7)$, where $N_7$ is the PN structure of figure 4.2, and $m^0_7 \in \Delta(N_7)$. Likewise, the minimally restrictive LESP for $N_8(m^0_8)$, where $N_8$ is the PN structure of figure 4.3, and $m^0_8 \in \Delta(N_8)$, permits the firing of $t_1$ only when there is at least one token in places $p_4$ and $p_5$, or there are at least two tokens in $p_1$.

![Supervisory Policy Diagram](image-url)

Figure 4.4: The minimally-restrictive LESP that ensures all reachable markings of the plant FCPN $N_6(m^0_6)$ are in the right-closed set $\Delta(N_6)$ identified by the minimal elements $\{(2,0)^T, (0,3)^T\}$. This policy is a minimally restrictive LESP for $N_6(m^0_6)$ if $m^0_6 \notin \Delta(N_6)$. There is no LESP for $N_6(m^0_6)$ if $m^0_6 \notin \Delta(N_6)$.

The following section describes the class of PN structures $\mathcal{H}$, where $\mathcal{F} \subset \mathcal{H}$, where the set $\Delta(N)$ is right-closed for any $N \in \mathcal{H}$.
Permit $t_5$ if the marking that would result from its firing is greater than or equal to $(0 0 1 0)^T$, or $(2 0 0 0)^T$, or $(1 0 0 1)^T$, or $(0 2 0 0)^T$, or $(0 0 0 2)^T$

Supervisory Policy

Figure 4.5: The minimally restrictive LESP for the general plant PN $N_7(m^0)$. This LESP ensures the markings reachable under its supervision from any $m^0 \in \Delta(N_7)$ remain within the right-closed set $\Delta(N_7)$, where $\min(\Delta(N_7)) = \{(0 0 1 0)^T, (2 0 0 0)^T, (1 0 0 1)^T, (0 2 0 0)^T, (0 0 0 2)^T\}$.

4.2 $\mathcal{H}$ class PNs [2]

$\mathcal{H}$ class PNs are an extension of $\mathcal{F}$ class PNs, that is $\mathcal{F} \subset \mathcal{H}$. The existence of a liveness enforcing supervisory policy (LESP) for an instance of $\mathcal{H}$ class PNs, initialized at a marking, is sufficient to infer the existence of an LESP when the same instance is initialized at a larger marking. As a consequence, the existence of an LESP for the PN that results when a member of this family is initialized with a marking, is decidable. The class $\mathcal{H} \subseteq \tilde{\mathcal{H}}$, where $\tilde{\mathcal{H}}$ is defined in the following paragraphs.

Let, $\Omega(t) = \{\hat{t} \in T \mid t \cap \hat{t} \neq \emptyset\}$, denote the set of transitions that share a common input place with $t \in T$ for a PN structure $N = (\Pi, T, \Phi, \Gamma)$. Consequently, $(t_1 \in \Omega(t_2)) \Rightarrow (t_2 \in \Omega(t_1))$. Let $\tilde{\mathcal{H}}$ denote a class of PN structures where the following property is true:

$$\forall \mathbf{m} \in \Delta(N), \forall t_u \in T_u, \forall t \in \Omega(t_u), (t \in T_e(N, \mathbf{m})) \Rightarrow (t_u \in T_e(N, \mathbf{m})). \quad (4.1)$$

That is, $\tilde{\mathcal{H}}$ is a class of PN structures where, if a transition $t$ is state-enabled,
then all uncontrollable transitions that share a common input place with $t$ are also state-enabled at any marking in $\Delta(N)$. When the proofs of Lemma 5.1, Observations 5.2, 5.3, 5.4, and Lemma 5.5 of reference [11] are applied to the case when $N \in \tilde{H}$, we get the result shown below.

**Theorem 4.2.1.** $\Delta(N)$ is right-closed if $N \in \tilde{H}$

However, right-closure of $\Delta(N)$ does not necessarily imply membership in $\tilde{H}$. $\Delta(N_2)$ is right-closed for the general PN structure $N_2$ in Figure 2.2. $\Delta(N_2)$ is identified by the inequality $(1 1 1 1 1)^T \geq 1$, and $m = (0 0 1 0)^T \in \Delta(N_2)$. $t_2$ and $t_3$ are uncontrollable transitions that have a common input place. While $t_3 \in T_e(N_2, m)$, $t_2 /\in T_e(N_2, m)$.

There is an LESP for the PN $N(m^0)$ if and only if $m^0 \in \Delta(N)$, and the existence of an LESP is undecidable for a general PN (cf. corollary 5.2, [12]). This would mean that the set $\Delta(N)$ cannot be computed for an arbitrary PN structure $N$. To overcome this limitation, we modify the requirement of equation 6.1 as

$$\forall m \in \mathcal{N}^n, \forall t_u \in T_u, \forall t \in \Omega(t_u), (t \in T_e(N, m)) \Rightarrow (t_u \in T_e(N, m)). \quad (4.2)$$

This requirement defines a class of PNs, which we denote as $\mathcal{H}(\subseteq \tilde{H})$, and from theorem 1, we conclude $\Delta(N)$ is right-closed for any $N \in \mathcal{H}$.

The following result characterizes the class $\mathcal{H}$.

**Theorem 4.2.2.** A PN structure $N = (\Pi, T, \Phi, \Gamma)$ belongs to the class $\mathcal{H}$ if and only if $\forall p \in \Pi, \forall t_u \in p^* \cap T_u$,

$$(\Gamma(p, t_u) = \min_{t \in p^*} \Gamma(p, t)) \land (\forall t \in \Omega(t_u), \bullet t_u \subseteq t).$$

**Proof.** (If) Suppose, $t \in T_e(N, m)$ for $m \in \mathcal{N}^n$, and $\exists t_u \in \Omega(t) \cap T_u(\Rightarrow t \in \Omega(t_u))$. Since $\bullet t_u \subseteq \bullet t$ and $\forall p \in p^*, \Gamma(p, t_u) = \min_{t \in p^*} \Gamma(p, t)$, it follows that $t_u \in T_e(N, m)$.

(Only If) We will show that the violation of requirement in the statement of the theorem for a PN structure $N$ would imply that $N \notin \mathcal{H}$.

Suppose $\exists p \in \Pi, \exists t_u \in p^* \cap T_u$ such that either

1. $\Gamma(p, t_u) > \min_{t \in p^*} \Gamma(p, t)$, or
2. $\exists t \in \Omega(t_u), \bullet t_u \subseteq \bullet t \neq \emptyset$. 
In each of these cases we construct a marking \( m \in N^n \) such that \( \exists t \in \Omega(t_u), t \in T_e(N, m) \) and \( t_u \notin T_e(N, m) \), which leads to the conclusion that \( N \notin \mathcal{H} \).

For the first-case, the marking \( m \) places exactly \( (\min_{p \in P} \Gamma(p, t)) \)-many tokens in \( p \), and sufficient tokens in the input places of any transition \( \hat{t} \in \Omega(t_u) \) such that \( \Gamma(p, \hat{t}) = \min_{p \in P} \Gamma(p, t) \) that will result in \( \hat{t} \in T_e(N, m) \) and \( t_u \notin T_e(N, m) \).

Similarly, for the second-case, the marking \( m \) places sufficient tokens in the input places of \( t \) such that \( t \in T_e(N, m) \), while ensuring that the places in \( (\cdot t_u - \cdot t) \) remain empty. Consequently, \( t \in T_e(N, m) \) and \( t_u \notin T_e(N, m) \). \( \square \)

There is an \( O(n^2m^2) \) algorithm that decides if an arbitrary PN structure belongs to the class \( \mathcal{H} \), where \( n = \text{card}(\Pi) \) and \( m = \text{card}(T) \). The right-closure of \( \Delta(N) \) for any \( N \in \mathcal{H} \), along with the results in reference [11] implies that the existence of an LESP for \( N(m^0) \) is decidable. Furthermore, the software package described in references [14, 15, 16, 5] can be used to compute the minimally restrictive LESP for \( N(m^0) \), when it exists. Additionally, \( \mathcal{F} \subset \mathcal{H} \) and \( \mathcal{G} \subset \mathcal{F} \).

4.3 On the role of Choice/Non-Choice Transitions in Supervisory Control of PNs

A transition \( t \in T \) is said to be a choice-transition (non-choice transition) if \( (\cdot t) \neq \{t\} \Rightarrow (\cdot t) = \{t\} \). The minimally restrictive LESP for an ordinary FCPN \( N(m^0) \) does not control-disable a non-choice (controllable) transition [25]. The following result shows that a similar observation holds for any minimally restrictive LESP for \( N(m^0) \) where \( N \in \mathcal{H} \).

**Theorem 4.3.1.** Suppose \( m^0 \in \Delta(N) \) for a PN \( N(m^0) \), where \( N \in \mathcal{H} \), then the minimally restrictive LESP for \( N(m^0) \) does not disable any controllable transition \( t_c \in T_c \) that satisfies the requirement \( (\cdot t_c) = \{t_c\} \)

The proof of observation 3 in reference [25], originally stated in the context of ordinary FCPNs, *mutatis mutandis*, serves as a proof of the above claim. It is not repeated here in the interest of space. This observation does not hold for general PN structures.
As a consequence of this observation, without loss of generality, we can assume all non-choice transitions are uncontrollable, even when they are not. This is critical to the execution of the software package described in references [14, 15, 16, 5], which is illustrated by example. The PN structures $N_9$ and $N_5$ (cf. figure 3.2) shown in figure 4.6(a) and 4.6(b) are FCPN structures, and consequently they belong to the class $\mathcal{H}$. The only difference between them is that the non-choice transition $t_5$ is controllable (resp. uncontrollable) in $N_9$ ($N_5$).

The sets $\Delta(N_9)$ and $\Delta(N_5)$ are identical, and are identified by the twenty-four minimal elements shown in figure 3.3, which shows the output generated by the above mentioned software for $N_7$.

We turn our attention to the iteration scheme for $N_9$ where $t_5$ is left as a controllable transition. The right-closed set of initial markings for which there is an LESP for the fully-controlled version of $N_9$ is identified by the same set of eight minimal elements shown in the initial part of the output of figure 3.3. The largest controllable subset of this set ($\Psi_0$) is identified by the six minimal elements of figure 3.3 along with the vector $(001000000)^T$. This extra minimal element is due to the fact that $t_5$ is controllable in $N_9$, which fails the loop-test along with the four that failed the test in figure 3.3. After the elevation by unit-vectors as described above, the next iterate $\Psi_1$ has the twenty-four minimal elements shown in figure 3.3 together with eight new elements

\[
\{(101000000)^T, (011000000)^T, (002000000)^T, \\
(001100000)^T, (001010000)^T, (001001000)^T, \\
(001000100)^T, (001000001)^T\}.
\]

That is, the minimal element $(001000000)^T$ of $\Psi_0$ is replaced by these eight elements, which defines $\Psi_1$. These eight elements fail the loop-test, and are replaced with more elevated vectors, and so on. The right-closed set that is defined by this iteration scheme in the limit is the set $\Delta(N_7)$ described earlier. But, as a computation scheme this procedure will not terminate. This issue is mitigated by ensuring that all controllable, non-choice transitions are interpreted as being a part of the set of uncontrollable transitions.

As this example illustrates, the process of relabeling each non-choice transition as members of the set of uncontrollable transitions improves the running-
Figure 4.6: (a) The PN structure $N_6$ is a member of $\mathcal{H}$ as it is an FCPN structure. (b) The PN structure $N_7$ is also a member of $\mathcal{H}$. The non-choice transition $t_5$ is controllable (uncontrollable) in $N_6$ ($N_7$).

time of the software described in reference [5], which is described at length in the next chapter.

The next section addresses the issue of synthesizing LESP’s for a PN from that of another PN that is “similar” to it.

4.4 LESP’s for Petri Nets that are Similar [3]

In this section we review the results in reference [3]. The notion of simulation was introduced in the PN literature to formalize the concept of “similarity” among two PNs [26]. If a PN can simulate another PN, one could say that they are similar in some sense. This concept is generalized to include controlled PNs that are under the influence of a supervisory policy. We derive a necessary and sufficient condition for the existence of LESP’s in PNs in a simulation relationship. The LESP for the simulated PN, along with the results on this paper and additional observations can oftentimes provide the LESP for the other PN.

We first extend Best’s definition of simulation [26] to controlled PNs. Let $N = (\Pi, T, \Phi, \Gamma)$ and $\hat{N} = (\hat{\Pi}, \hat{T}, \hat{\Phi}, \hat{\Gamma})$ be two PN structures. Suppose
$\alpha : T \to \hat{T}$ is an injection (i.e. every member of $T$ has an image in $\hat{T}$; but, the converse is not necessarily true). This injective function can be extended to include sets of transitions $T_1 \subseteq T$ as

$$\alpha(T_1) = \bigcup_{t \in T_1} \alpha(t).$$

With a slight abuse of notation, we extend the above function to strings of transitions $\alpha : T^* \to \hat{T}^*$ for $\sigma \in T^*$ and $t \in T$, as

$$\alpha(\sigma t) = \alpha(\sigma)\alpha(t)$$

where $\alpha(\epsilon) = \epsilon$, and $\epsilon$ is the empty-string. The inverse-function $\alpha^{-1} : \hat{T}^* \to T^*$ is defined for $\hat{\sigma} \in \hat{T}^*$, $\hat{t} \in \hat{T}$, as

$$\begin{align*}
\alpha^{-1}(\hat{\sigma}t) &= \alpha^{-1}(\hat{\sigma}), & \text{when } \hat{t} \notin \alpha(T), \\
\alpha^{-1}(\hat{\sigma}t) &= \alpha^{-1}(\hat{\sigma})\alpha^{-1}(\hat{t}), & \text{when } \hat{t} \in \alpha(T),
\end{align*}$$

and $\alpha^{-1}(\epsilon) = \epsilon$.

**Definition 4.4.1.** Let $N = (\Pi, T, \Phi, \Gamma)$ and $\hat{N} = (\hat{\Pi}, \hat{T}, \hat{\Phi}, \hat{\Gamma})$ be two PN structures, and $\alpha : T \to \hat{T}$ be an injective function. Suppose $N(m^0)$ (resp. $\hat{N}(\hat{m}^0)$) is supervised by policy $P : N^{\text{card}(\Pi)} \times T \to \{0, 1\}$ (resp. $\hat{P} : \hat{N}^{\text{card}(\hat{\Pi})} \times \hat{T} \to \{0, 1\}$).

We say that $\hat{m}^0$ under the supervision of $\hat{P}$ simulates $N(m^0)$ under the supervision of $P$ if and only if there is a surjection $\beta : N^{\text{card}(\Pi)} \to \hat{N}^{\text{card}(\hat{\Pi})}$ such that:

1. $m^0 = \beta(\hat{m}^0)$,

2. Suppose $m^1 = \beta(\hat{m}^1), \hat{m}^1 \in \hat{R}(\hat{N}, \hat{m}^0, \hat{P})$ and $m^1 \in R(N, m^0, P)$, then
   
   (a) whenever $m^1 \xrightarrow{t} m^2$ in $N$ under $P$, then $\exists \hat{m}^2 \in \beta^{-1}(m^2), \exists \hat{\sigma} \in \hat{T}^*$ such that $m^1 \xrightarrow{\hat{\sigma}} \hat{m}^2$ under $\hat{P}$ in $\hat{N}$, and $\alpha^{-1}(\hat{\sigma}) = t$.

   (b) whenever $\hat{m}^1 \xrightarrow{\hat{\sigma}} \hat{m}^2$ under $\hat{P}$ in $\hat{N}$ for some $\hat{\sigma} \in \hat{T}^*$, then $m^1 \xrightarrow{\alpha^{-1}(\hat{\sigma})} \beta(\hat{m}^2)$ under $P$ in $N$.

As noted in reference [26], the fact that $\alpha : T \to \hat{T}$ is an injective function would mean that $\text{card}(\hat{T}) \geq \text{card}(T)$. The transitions in $\alpha(T)$ simulate the
Theorem 4.4.2. Suppose \( \alpha \) \( PN \) structures, and string under the mapping to be viewed as an internal to \( \hat{T} \) every firing string in \( N \) of transitions in \( T \). A marking \( \hat{m} \in N_{\text{card}(\Pi)} \) of \( \hat{N} \) represents the marking \( \beta(\hat{m}) \) in \( N \). There could be many markings of \( \hat{N} \) can represent the a single marking of \( N \). But the surjective nature of \( \beta(\bullet) \) guarantees that every marking of \( N \) is represented by some marking of \( \hat{N} \).

Item 1 of definition 4.4.1 requires that the initial marking of \( \hat{N} \) must represent the initial marking of \( N \). Item 2a requires that the firing of any transition \( t \) in \( N \) under the supervision of \( \mathcal{P} \) must be simulated by the firing of a string of transitions in \( \hat{N} \) under the supervision of \( \hat{\mathcal{P}} \), while item 2b requires that every firing string in \( \hat{N} \) under the supervision of \( \hat{\mathcal{P}} \) has a corresponding firing string under the mapping \( \alpha^{-1}(\bullet) \) that is permitted under \( \mathcal{P} \) in \( N \).

We now state and prove the main result of this paper.

**Theorem 4.4.2.** Suppose \( N = (\Pi, T, \Phi, \Gamma) \) and \( \hat{N} = (\hat{\Pi}, \hat{T}, \hat{\Phi}, \hat{\Gamma}) \) are two \( PN \) structures, and \( \alpha: T \to \hat{T} \) is an injective function. Let \( N(m^0) \) (resp. \( \hat{N}(\hat{m}^0) \)) be supervised by policy \( \mathcal{P} : N_{\text{card}(\Pi)} \times T \to \{0,1\} \) (resp. \( \hat{\mathcal{P}} : N_{\text{card}(\Pi)} \times \hat{T} \to \{0,1\} \)), and \( \hat{N}(\hat{m}^0) \) under the supervision of \( \hat{\mathcal{P}} \) simulates \( N(m^0) \) under \( \mathcal{P} \) with respect to \( \alpha(\bullet) \), then \( \mathcal{P} \) is an LESP for \( N(m^0) \) if and only if \( \forall \hat{t} \in \alpha(T), \hat{t} \) is live under \( \hat{\mathcal{P}} \).

**Proof.** (Only If) Let \( \mathcal{P} \) be an LESP for \( N(m^0) \), and \( \hat{m}^0 \overset{\hat{\sigma}_1}{\to} \hat{m}^1 \) under \( \hat{\mathcal{P}} \) in \( \hat{N}(\hat{m}^0) \). From item 1 of definition 4.4.1 we have \( m^0 = \beta(\hat{m}^0) \). From item 2b, \( m^0 \overset{\alpha^{-1}(\hat{\sigma}_1)}{\to} \beta(\hat{m}^1) \) under \( \mathcal{P} \) in \( N \).

Since \( \mathcal{P} \) is an LESP for \( N \), \( \forall t \in T, \exists \sigma_2 \in T^*, \exists m^2 \in N_{\text{card}(\Pi)} \) such that \( \beta(\hat{m}^1) \overset{\sigma_2}{\to} m^2 \) in \( N \) under \( \mathcal{P} \), where \( t \) occurs once in \( \sigma_2 \). From item 2a of definition 4.4.1, \( \exists \hat{\sigma}_2 \in \hat{T}^*, \exists \hat{m}^2 \in R(\hat{N}, \hat{m}^0, \hat{\mathcal{P}}) \) such that \( \hat{m}^1 \overset{\hat{\sigma}_2}{\to} \hat{m}^2 \) and \( \alpha(t) \) occurs in \( \hat{\sigma}_2 \). Therefore, all \( \hat{t} \in \alpha(T) \) is live under \( \hat{\mathcal{P}} \) in \( \hat{N}(\hat{m}^0) \).

(If) Suppose, \( \forall \hat{t} \in \alpha(T), \hat{t} \) is live in \( \hat{N} \) under \( \hat{\mathcal{P}} \). Let \( \hat{m}^0 \overset{\hat{\sigma}_1}{\to} \hat{m}^1 \) under \( \mathcal{P} \) in \( N \), where \( \sigma_1 = t_1 \cdots t_m \). From items 1 and 2a of definition 4.4.1, \( \exists \hat{\sigma}_1, \ldots, \hat{\sigma}_m \in \hat{T}^* \) such that \( \hat{m}^0 \overset{\hat{\sigma}_1 \cdots \hat{\sigma}_m}{\to} \hat{m}^1 \) in \( \hat{N} \) under the supervision of \( \hat{\mathcal{P}} \), and \( \hat{m}^1 = \beta(\hat{m}^1) \).

Since all transitions in \( \alpha(T) \) are live under \( \hat{\mathcal{P}} \) in \( \hat{N} \), \( \exists \hat{\sigma}_2 \in \hat{T}^*, \exists \hat{m}^2 \in N_{\text{card}(\Pi)} \) such that \( \hat{m}^1 \overset{\hat{\sigma}_2}{\to} \hat{m}^2 \) in \( \hat{N} \) under \( \hat{\mathcal{P}} \), and \( \alpha(t) \) occurs in \( \hat{\sigma}_2 \), for any \( t \in T \). By item 2b of definition 4.4.1, \( \hat{m}^1 \overset{\alpha^{-1}(\hat{\sigma}_2)}{\to} \hat{m}^2 \) under \( \mathcal{P} \) in \( N \), where \( m^2 = \beta(\hat{m}^2) \) and \( t \in T \) occurs in \( \alpha^{-1}(\hat{\sigma}_2) \). Therefore, every transition in \( T \) is live under the supervision of \( \mathcal{P} \) in \( N(m^0) \).

\( \square \)
Table 4.1: The injective function $\alpha : T \rightarrow \hat{T}$ for the PNs shown in figure 1.1(a) and 1.1(b).

<table>
<thead>
<tr>
<th>$t \in T$</th>
<th>$\alpha(t) \in \hat{T}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$\hat{t}_1$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$t_8$</td>
</tr>
<tr>
<td>$t_2$</td>
<td>$t_9$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>$t_{10}$</td>
</tr>
<tr>
<td>$t_4$</td>
<td>$t_{11}$</td>
</tr>
<tr>
<td>$t_5$</td>
<td>$t_{12}$</td>
</tr>
<tr>
<td>$t_6$</td>
<td>$t_{13}$</td>
</tr>
</tbody>
</table>

The above result notes that the liveness of the transitions in the set $T$ of the PN $N$ under $\mathcal{P}$ is equivalent to the liveness of the transitions $\alpha(T)$ in the PN $\hat{N}$ under $\hat{\mathcal{P}}$. It is possible that some of the transitions in the set $\hat{T} - \alpha(T)$ are not live under $\hat{\mathcal{P}}$ in $\hat{N}$. However, if $\hat{N}$ has a structure that permits us to infer the liveness of the set $\hat{T} - \alpha(T)$ from the liveness of $\alpha(T) \subseteq \hat{T}$, then theorem 4.4.2 can be enhanced to a result that notes $\mathcal{P}$ is an LESP for $N(m^0)$ if and only if $\hat{\mathcal{P}}$ is an LESP for $\hat{N}$.

Consider the PN structures $N = (\Pi, T, \Phi, \Gamma)$ and $\hat{N} = (\hat{\Pi}, \hat{T}, \hat{\Phi}, \hat{\Gamma})$ shown in figure 1.1(a) and (b) respectively. The injective function $\alpha : T \rightarrow \hat{T}$ is defined in table 4.1.

The surjective function $\beta : N^{11} \rightarrow N^5$ for $N$ and $\hat{N}$ of figure 4.7(a) and (b) is given by the function $\beta(m) = (m_1, m_8, m_9, m_{10}, m_{11})^T$. The token load in places $\hat{p}_1, \hat{p}_8, \hat{p}_9, \hat{p}_{10}$ and $\hat{p}_{11}$ correspond to the token load in places $p_1, p_2, p_3, p_4$ and $p_5$ respectively.

Let $\hat{\mathcal{P}}$ be any policy that permits the firing of $\hat{t}_6$ if and only if $((\hat{m}_1 + \hat{m}_2 + \hat{m}_3 + \hat{m}_4 + \hat{m}_6 + \hat{m}_7) \geq 2)$. Let us also suppose the $\mathcal{P}$ permits the firing of $t_0$ at marking $m \in N^5$ if and only if $\hat{\mathcal{P}}$ permits the firing of $\hat{t}_1(= \alpha(t_0))$ at all markings in $\beta^{-1}(m)$. Then, $N(\hat{m}^0)$ under $\hat{\mathcal{P}}$ simulates $N(m^0)$ under $\mathcal{P}$. Consequently, by Theorem 4.4.2, $\mathcal{P}$ is an LESP for $N(m^0)$ if and only if every transition in $\alpha(T) = \{\hat{t}_1, \hat{t}_8, \hat{t}_9, \hat{t}_{10}, \hat{t}_{11}, \hat{t}_{12}, \hat{t}_{13}\}$ is live under $\hat{\mathcal{P}}$ in $\hat{N}$.

From the structure of $\hat{N}$ we can infer that the liveness of $\hat{t}_1$ implies the liveness of the transitions in the set $\hat{T} - \alpha(T) = \{\hat{t}_2, \hat{t}_3, \hat{t}_4, \hat{t}_5, \hat{t}_6, \hat{t}_7\}$. Consequently, $\mathcal{P}$ is an LESP for $N(m^0)$ if and only if $\hat{\mathcal{P}}$ is an LESP for $\hat{N}(\hat{m}^0)$.

From method of references [27, 28], we know that the supervisory policy $\mathcal{P}$
Figure 4.7: If $\hat{\mathcal{P}}$ is a policy that permits the firing of $\hat{t}_6$ if and only if $((\hat{m}_1 + \hat{m}_2 + \hat{m}_3 + \hat{m}_4 + \hat{m}_6 + \hat{m}_7) \geq 2)$; and, permits the firing of $\hat{t}_1 (= \alpha(t_0))$ at every marking in $\beta^{-1}(\mathbf{m})$ if and only if $\mathcal{P}$ permits the firing of $t_0$ at marking $\mathbf{m}$, then $\hat{N}(\hat{\mathbf{m}}^0)$ under $\hat{\mathcal{P}}$ simulates $N(\mathbf{m}^0)$ under $\mathcal{P}$. From Theorem 4.4.2 we infer that the policy $\mathcal{P}$ is an LESP for $N(\mathbf{m}^0)$ if and only if every transition in $\alpha(T) = \{\hat{t}_1, \hat{t}_8, \hat{t}_9, \hat{t}_{10}, \hat{t}_{11}, \hat{t}_{12}, \hat{t}_{13}\}$ is live under $\hat{\mathcal{P}}$ in $\hat{N}$. 
that permits the firing of $t_0$ if and only if $((m_1 \geq 2) \lor (m_4 \geq 1) \land (m_5 \geq 2))$ is an LESP. As a consequence of the above observation, we can conclude that the supervisory policy $\widehat{P}$ that permits the firing of $\widehat{t}_1$ if and only if $((\hat{m}_1 \geq 2) \lor (\hat{m}_{10} \geq 1) \land (\hat{m}_{11} \geq 2))$; and permits the firing of $\widehat{t}_6$ if and only if $((\hat{m}_1 + \hat{m}_2 + \hat{m}_3 + \hat{m}_4 + \hat{m}_6 + \hat{m}_7) \geq 2)$ is an LESP for $\widehat{N}(m^0)$. That is, the LESP for the larger PN $\widehat{N}(m^0)$ was synthesized from the LESP for the smaller PN $N(m^0)$ with the help of the results in this paper.

The above observation used the structure of $\widehat{N}$ to conclude that the liveness of the set of transitions $\alpha(T)$ under a supervisory policy implies the liveness of the transitions in $\widehat{T} - \alpha(T)$ as well. We suggest the identification of general conditions that are sufficient to make this inference on a wider class of PNs as a future research topic.
CHAPTER 5

LESP-SYNTHESIS ALGORITHM AND OBJECT-ORIENTED IMPLEMENTATION

The theoretical underpinnings of the procedure for the synthesis of the minimal elements of $\Delta(N)$ has been covered in earlier chapters. In this section we review the implementation details of the procedure that computes the members of $\text{min}(\Delta(N))$.

Reference [5] discusses the object-oriented implementation of the algorithms to obtain LESP for the class of PNs for which the set of marking $\Delta(N)$, introduced earlier, is right-closed. The implementation was done in C++ using Microsoft Visual C++ compiler as a command-line application. The implementation primarily uses STL containers viz. std::vector, a sequence container for object collections. To enhance the performance and efficiency, the implementation also uses features like Boost C++ libraries. Below are some illustrative examples.

Figure 5.1 shows the Object oriented representation of a minimally restrictive LESP. The implementation is done within four major classes called PetriNet, NodeTable, MinimalElementsManager and MarkingVector that are described in great detail in reference [5]. We refrain from presenting there functionalities in this document in the interest of space.

The general FCPN structure $N_6$ shown in Figure 4.1 is a member of class $\mathcal{F}$. Consequently, we know $\Delta(N_6)$ is right-closed. Figure 5.2 shows the FCPN structure of figure 4.5 initialized with two tokens in $p_1$. We will refer to this PN as $N_7(m^0_7)$. The input file for $N_7(m^0_7)$ is shown in Figure 5.3. Figure 5.4 shows the output generated by the software, which lists five minimal elements of $\text{min}(\Delta(N_7))$. The LESP that disables $t_5$ at any marking in $\Delta(N_7)$ if its firing would result in a new marking that is not in $\Delta(N_7)$, is the minimally restrictive LESP for $N_7(m^0_7)$ for any $m^0_7 \in \Delta(N_7)$.

On executing the algorithm to obtain the coverability graph for this net, millions of nodes are generated and owing to the resource constraint of the computing device this algorithm takes really long to run. With increase in
The controllable (uncontrollable) transitions are identified by a 1(0).

Several automated manufacturing systems and service enterprise systems can be mod-
elled using one of the known classes of general

The complexity of the system, i.e. with a large number of places and transitions the computations can become tedious resulting in large computational time.

Consequently, we worked on changing the structure of the code to reduce the computational time. Some of the actions that were taken:

1. Executing the code without Boost C++ libraries

2. Incorporating the algorithm for coverability graph used in [11] in C

We implemented the code without using Boost C++ libraries, i.e. using iterators instead of BOOST_FOREACH. We did not use unordered maps and shared pointers instead the node table was declared as a vector.
Figure 5.3: Input file for $N_{6}(m_{0}^{6})$. The first line shows that there are four places and six transitions in the PN structure. This is followed by the $(4 \times 6)$ IN and OUT matrices, which accounts for the eight lines that follow the first line. The penultimate line identifies the initial marking $m_{0}^{6}$ that places two tokens in place $p_{1}$, and the last line identifies the controllable (uncontrollable) transitions with a 1(0). Since this line is all zeros, but for the fifth position, it follows that the only controllable transition in this structure is $t_{5}$.

computational time using this implementation is about the same as the computational time using Boost libraries. To validate this result we implemented several examples. When $N_{3}$ was executed without Boost libraries it took 0.26 seconds which is about the same as using Boost libraries. However, for PNs with complex coverability graph with millions of nodes this implementation takes longer than the implementation using Boost libraries. Hence, PN $N_{8}$ shown in Figure ?? takes a lot longer without the libraries than it would using them. On further investigation we found that the function `processNode()` in the class `NodeTable.cpp` takes up a major chunk of the computation time.

`processNode()`, a recursive method is the primary method of `NodeTable` class which in turn invokes the other member functions to compute the vertex and edge parameters of the reachability graph. This method is initialized with the initial marking $m^{0}$. Each vertex together with all its connecting edges forms a node in the `NodeTable` and is characterized by the members `fromNode`, `marking`, `byTransition`, `enabledTransitions`, `nodeType`, `concurrent`, `conflicting`, `duplicateNode` and index ([5]).

To reduce the computation time taken to calculate the coverability graph as a next step we decided to incorporate the algorithm for coverability graph in C. For this we had to change the `processNode()` and incorporate the algorithm in C under this method. `processNode()` calls `ClassifyNode()`,

```
4 6
2 0 2 0 0 0
0 2 0 0 2 0
0 0 0 1 0 0
0 0 0 0 0 1
0 3 0 0 0 1
3 0 0 2 0 0
0 0 1 0 0 0
0 0 0 0 1 0
2 0 0 0
0 0 0 0 1 0
```
Input File = "pn1"

Incidence Matrix:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-2</td>
<td>3</td>
<td>-2</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-2</td>
<td>.</td>
<td>-2</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>-1</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Initial Marking : ( 2 0 0 0 )

There is an LESP for this (fully controlled) PN

---

Minimal Elements of the fully controlled Net
--------------------------------------------

1: ( 0 0 1 0 )
2: ( 1 0 0 1 )
3: ( 0 0 0 2 )
4: ( 0 2 0 0 )
5: ( 2 0 0 0 )

List of Controllable Transitions
--------------------------------

15

(Final) Minimal Elements of the control-invariant set
-----------------------------------------------------

1: ( 0 0 1 0 )
2: ( 1 0 0 1 )
3: ( 0 0 0 2 )
4: ( 0 2 0 0 )
5: ( 2 0 0 0 )

This is An LESP

Elapsed Time : 0.029148 secs

---

Figure 5.4: The output file generated from the input file of Figure 5.3

isNodeDuplicate( ), firetransition( ) and findOmegaPlaces( ). ClassifyNode( ) further calls identifyEnabledTransitions( ), doTransitionsOverlap( ) and areEnabledTransitionsConcurrent( ). Thus, changing the method processNode( ) would require a complete change in the structure of NodeTable.cpp.

Class NodeTable and PetriNet are tightly coupled with each other and hence are declared as friends of each other, allowing both the classes to access each others private members. PetriNet contains objects of NodeTable and hence is marked by a “Has-a” relationship while NodeTable holds a pointer to Class PetriNet ([5]). Consequently, changing processNode( ) would require a change in the entire structure of the existing code. This process was hence stalled and we began to look at reduction/abstraction techniques to reduce computation time. This is described, after a fashion, in the next chapter.
CHAPTER 6

REDUCTION RULES AND OBJECT ORIENTED IMPLEMENTATION OF REDUCTION ALGORITHM

This chapter discusses the reduction rules and the object-oriented implementation of the reduction algorithm. The reduction rules simplify the PN structure. The minimally restrictive LESP for the reduced PN can be computed using the existing software [14]. This is followed by an abstraction step that yield the LESP for the original PN model. The limitation of this technique is that while the technique guarantees a LESP for the original PN model the LESP might not necessarily be minimally restrictive. The object-oriented implementation uses similar structure and some of the variables used in reference [14] under the assumption that an integration to the existing code in the future would be faster. The implementation has been done in C++ using Mac OS Xcode version 4.2.1 as a command line application. This implementation primarily uses STL Containers viz. \textit{std:vector}, a sequence container representing arrays that can change in size. There are two main parts to this implementation

- The first part takes the original PN as an input. Using reduction techniques the PN is reduced. The output of this code is the incidence matrix of the reduced PN.

- The second code works on the minimal elements of the reduced PN. The minimal elements of the original PN is deduced using this code. The final result is not always minimally restrictive.

6.1 Reduction Rules

This section describes the rules that are used to simplify the PN model. We use three techniques to compute the reduced PN. The argument behind these techniques and the conditions under which they are applicable are discussed in this section.
6.1.1 Rule 1

Rule 1 is pictorially depicted in Figure 6.1. It is imperative that, in the original PN,
\[ p_a \cap T_u - \{ t_m \} = \emptyset. \]

That is, none of the output transitions to \( p_a \) (with the exception of \( t_m \), possibly) are uncontrollable.

A path \( p_a \xrightarrow{w_3} t_m \xrightarrow{w_2} p_b \xrightarrow{w_3} t_n \) in the original PN is simplified as \( \tilde{p}_a \xrightarrow{\frac{w_1}{w_2}} t_n \).

The resulting reduced PN and the original PN have behavioral similarity (cf. section 4.4). The argument for this rule is that - a single firing of \( t_n \) in the original PN will take away \( w_3 \) tokens from \( p_b \). If \( t_m \) has to fire some \( m \)-many times to place \( w_3 \) tokens in \( p_b \), then \( m \times w_2 = w_3 \Rightarrow \frac{w_3}{w_2} = m \). The \( m \)-many firings of \( t_m \) will take away \((m \times w_1)\)-many tokens from \( p_a \). Since \( m = \frac{w_3}{w_2} \), it follows that \( m \)-many firings of \( t_m \) will take away \( \frac{w_1}{w_2} \) tokens from \( p_a \) for each single firing of \( t_n \). Therefore, a single firing of \( t_n \) in the reduced PN structure will result in \( \frac{w_1}{w_2} \) number of tokens being removed from \( \tilde{p}_a \). The rule is applied provided the following two conditions are satisfied:

1. The number of incoming and outgoing arcs to \( t_m \) and \( p_b \) is equal to one.

2. \( \frac{w_1}{w_2} \) is an integer. This can be deduced from the argument above where we proved that for \( m \) many firing of \( t_m \), \( \frac{w_1}{w_2} = m \).

Once the reduced PN is computed using this rule we need to deduce the minimal elements of the original PN from the reduced PN. That is, we need to figure out how to distribute \( b \) tokens in \( \tilde{p}_a \) in the reduced PN to the original PN. If to begin with, say there are \( b \) tokens in \( p_a \) and zero tokens in \( p_b \) and \( k \) firings of \( t_m \) results in \( p \) tokens in \( p_a \) and \( q \) tokens in \( p_b \). This would mean that \((b - p)\) tokens from \( p_a \) were removed by the firing process from \( p_a \). Each firing of \( t_m \) will take away \( w_1 \) tokens from \( p_a \) and place \( w_2 \) tokens in \( p_b \). Repeated applicaiton of this process has now resulted in \( q \) tokens in \( p_b \). This would mean that

\[ q = b - p \Rightarrow w_2 \Rightarrow b = p + \frac{w_1}{w_2} q. \]

Hence, in order to deduce the minimal elements for the original PN we look for all possible integer solutions for the equation,
\[ b = p + \frac{w_1}{w_2} q \] (6.1)

Figure 6.1: Rule #1

6.1.2 Rule 2

Rule 2 is pictorially depicted in Figure 6.2. A path \( t_a \xrightarrow{w_1} p_m \xrightarrow{w_2} t_b \xrightarrow{w_3} p_n \) in the original PN is simplified as \( t_a \xrightarrow{w_3} \tilde{p}_n \). The resulting reduced PN and the original PN have behavioral similarity (cf. section 4.4). The argument for this rule is that -a single firing of \( t_a \) in the original PN will place \( w_1 \) tokens in \( p_m \). If \( t_b \) has to fire \( m \) times to empty \( p_m \), it follows that \( m \times w_2 = w_1 \Rightarrow m = \frac{w_1}{w_2} \).

The process of \( m \)-many firings of \( t_b \) will place \( m \times w_3 \) tokens in \( p_n \). Therefore, single firing of \( t_a \) in the reduced PN will place \( \left( \frac{w_1}{w_2} \right) \)-many tokens in \( \tilde{p}_n \). The rule is applied provided the following two conditions are satisfied:

1. The number of incoming and outgoing arcs to \( p_m \) and \( t_b \) is equal to one.

2. \( \frac{w_1}{w_2} \) is an integer. This can be deduced from the argument above where we proved that if \( m \) many firings of \( t_b \) empties out \( p_m \), \( \frac{w_1}{w_2} = m \).

Once the reduced PN is computed using this rule we need to deduce the minimal elements of the original PN from the reduced PN. That is, we need to figure out how to distribute \( a \) tokens in \( \tilde{p}_n \) in the reduced PN to the original PN. If there are \( x \) tokens in \( p_m \) and \( y \) tokens in \( p_n \). Each firing of \( t_b \)
in the original PN takes $w_2$ tokens out of $p_m$ and places $w_3$ tokens in $p_n$. If we wait for $t_b$ to fire $k$ number of times such that $p_m$ is empty and we end up with $a$-many tokens in $p_n$. This would mean that

$$\frac{x}{w_2}w_3 + y = a$$

Hence, in order to deduce the minimal elements for the original PN we look for all possible integer solutions for the equation,

$$a = y + \frac{w_3}{w_2}x \quad (6.2)$$

This assumes that having $x$ tokens in $p_m$ will make $t_b$ fire as often as necessary, till is emptied. This is the same as saying that $x \geq w_2$. Hence if one of the solutions for 6.2 yields an $x$ value that is less than $w_2$, then we replace that value of $x$ with $w_2$.

The argument behind this is that if $x < w_2$, then $t_b$ cannot fire even once. However, if we were to permit $t_b$ to fire “fractionally” (i.e. $\frac{x}{w_2}$ - th of a single firing of $t_b$), we would place the “appropriate fraction-of $w_3$ -many-tokens” (i.e. $\frac{x}{w_2}w_3$ - many tokens) in $p_n$, which will

1. empty $p_m$ and
2. place $a$ tokens in $p_n$

Replacing $x$ with $w_2$ in such cases will place more than $a$ tokens in $p_n$ which would work too. Thus, while computing all possible integer solutions for $x$ and $y$ we take the following into consideration:

$$\text{if } x \text{ is not equal to } 0, x = \max(x, w_2). \quad (6.3)$$

This step would be unnecessary if the weights of the PN are unitary.

6.1.3 Rule 3

Rule 3 is pictorially depicted in Figure 6.3. This rule is referred to as merging in this document. The argument provided below is for generic weights. However, in the object-oriented implementation for this particular rule the implementation is for unit weights. If the LESP software presented a minimal element for the reduced PN that assigns $a$-many tokens to $\tilde{p}_n$, then we
Figure 6.2: Rule #2

need to figure out how to distribute the \(a\)-many tokens back to the original PN. The claim is that we need to find all possible integer solutions to the equation,

\[
a = z + \frac{w_2}{w_1}x + \frac{w_4}{w_3}y \quad (6.4)
\]

The condition under which this rule can be used is:

1. \(\frac{w_2}{w_3}\) has to be an integer, and

2. \(\frac{w_4}{w_4}\) has to be an integer.

The reduction rule says each firing of \(t_a\) in the reduced PN will place \(((w_2 + w_4) / w_3)\)-many (i.e. integer-many) tokens in \(\tilde{p}_n\). A single firing of \(t_a\) in the original PN will place \(w_1\)-many (respectively \(w_2\)-many) tokens in \(p_1\) (respectively \(p_2\)). If \(t_b\) (respectively \(t_c\)) fires \(m\)-many times (respectively \(n\)-many times) to empty \(p_1\) (respectively \(p_2\)), we have \(m = \frac{w_1}{w_3}\) (respectively \(n = \frac{w_2}{w_4}\)), and the net tokens that would be deposited in \(p_n\) would be \((m \times w_5 + n \times w_6)\) which squares with the reduction rule.

In the present implementation, this rule is applied only for PNs with unitary weights. Consequently, the steps outlined in equation 6.3 are not needed currently. However, a parallel to equation 6.3 should be used when this is rule is applied to general PNs.
6.2 Implementation of Reduction Techniques

This section describes the object-oriented implementation of the reduction techniques. The input to this implementation is the incidence matrix of the original PN. The implementation uses reduction techniques to reduce the PN and the output is the incidence matrix of this reduced PN.

6.2.1 Class Definitions and Diagram

There are two main classes in this implementation Reduction and MarkingVector. Reduction.h is the header file that contains the declaration for the Reduction class and its variables and functions. Reduction.h also includes the header file Marking.h. Class Reduction has one or more objects of the Class MarkingVector and hence is marked by a “Has-a” relationship with MarkingVector. Class Reduction implements the reduction algorithm. Marking.h contains the declaration for the MarkingVector class and its variables and functions. The Figure 6.4 below shows a diagram of the class. The classes and the functions are described in detail in the sections that follow.

6.2.2 Class MarkingVector

The original implementation of the MarkingVector class [14] is retained in this implementation. The MarkingVector class forms the basic building block of the algorithms used to obtain the LESP for a net. The input for computing
an LESP of the net is an incidence matrix which in the original implementation is defined as a `std::vector` of pointers to objects of type `MarkingVector`. Thus, the `MarkingVector` class is the same as the original implementation to provide access to simpler mathematical operations along with the intention of integrating the reduction implementation to the original in the future.

The marking vector \( \mathbf{m} \) represents the number of tokens in each place at any give state of the net. This class contains a public variable `place` which is a vector of integers that stores the token count. The method `initialize()` and other overloaded methods are retained like in the original implementation to provide basic arithmetic operations of multiplication (\( \times \)), addition (+) and subtraction (−) and comparison operations such as <, ≤, == and ≥ on `MarkingVector` objects. The Figure 6.5 shows the structure of this class.
6.2.3 Class Reduction

The Reduction class implements the reduction algorithm to compute the reduced net from the given PN. Every input PN is worked on iteratively to find a path that can be reduced. With every iteration one reduced path and the corresponding inputs for the reduced net are computed. This continues until no reduced path can be found. The final result is then stored in a file and printed. The Reduction class uses objects of the MarkingVector and has methods that solve for the reduced net.

Variables and Initialization

This section describes the variables used in the class and their type and purpose in the implementation.

We begin with variables that were used in the original implementation [14]. To solve for LESP for any PN, the inputs that need to be provided by the user are number of places \( m \), number of transitions \( n \), the input matrix \((\text{IN})\), the output matrix \((\text{OUT})\) and the initial marking \(m^0\). In the original implementation these are stored in the members noOfPlaces, noOfTransitions, inputWeightsToTransition, outputWeightsFromTransition and initialMarking respectively. For a Controlled PetriNet, there is an additional input which corresponds to the transitions that are controllable. The input is given as a switch with 1 and 0 denoting controllable transitions and uncontrollable transitions respectively.

In this implementation, in addition to these members additional global variables noOfPlaces_global, noOfTransitions_global, inputWeightsToTransition_global and OutputWeightsFromTransition_global are provided. The “_global” variables are initialized and store \( m \), \( n \), input matrix and output matrix. The non-“_global” variables are set to the same values as the global variables initially but are subject to change in member functions of the class. With every newly computed PN the variables noOfPlaces, noOfTransitions, inputWeightsToTransition, outputWeightsFromTransition and initialMarking are updated accordingly.

\( \text{noOfPlaces, noOfTransitions, noOfPlaces\_global and noOfTransitions\_global} \) are integer members. initialMarking is of type MarkingVector. The members inputWeightsToTransition, outputWeightsFromTransition, inputWeightsToTran-
position\_global and OutputWeights\_FromTransition\_global are defined as std::vector of pointers to objects of type Marking\_Vector.

To compute the reduced net the algorithm first identifies the places and the transitions within the net that could potentially be removed and then identifies a path to reduce the PetriNet. The members places\_reduced and places\_reduced\_global are defined as std::vector of integers. The value 1 is assigned to the std::vector corresponding to the place that satisfies the condition to be removed or reduced. The concept of “\_global” and non-“\_global” variable is as mentioned earlier in this section. A transition that is identified as a transition that could potentially be reduced or removed is stored by assigning the value 1 to the std::vector, transitions\_reduced and transitions\_reduced\_global. The members places\_to\_be\_reduced and transitions\_to\_be\_reduced are integer counters that increment every time a value 1 is assigned to places\_reduced and transitions\_reduced respectively. These counters are reset during every iteration i.e for every reduced PN. The final output is written into the resultFile.

Table 6.1: Description of variables used in class Reduction

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>incidence_Matrix</td>
<td>vector &lt; Marking_Vector*&gt;</td>
<td>Represents the Incidence Matrix $C$ that is used to characterize any PetriNet.</td>
</tr>
<tr>
<td>initial_Marking</td>
<td>Marking_Vector</td>
<td>Stores the initial marking $m_0$ of a PetriNet.</td>
</tr>
<tr>
<td>input_Weights_To_Transition</td>
<td>vector&lt;Marking_Vector*&gt;</td>
<td>Represents the input matrix of the current PetriNet at any given step.</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>inputWeightsToTransition_global</code></td>
<td><code>vector&lt;MarkingVector*&gt;</code></td>
<td>Represents the input matrix of the original PetriNet.</td>
</tr>
<tr>
<td><code>noOfPlaces</code></td>
<td><code>int</code></td>
<td>Represents the number of places of the current PetriNet at any given step.</td>
</tr>
<tr>
<td><code>noOfPlaces_global</code></td>
<td><code>int</code></td>
<td>Represents the number of places for the original PetriNet provided by the user.</td>
</tr>
<tr>
<td><code>noOfTransitions</code></td>
<td><code>int</code></td>
<td>Represents the number of transitions of the current PetriNet at any given step.</td>
</tr>
<tr>
<td><code>noOfTransitions_global</code></td>
<td><code>int</code></td>
<td>Represents the number of transitions for the original PetriNet provided by the user.</td>
</tr>
<tr>
<td><code>outputWeightsToTransition</code></td>
<td><code>vector&lt;MarkingVector*&gt;</code></td>
<td>Represents the output matrix of the current PetriNet at any given step.</td>
</tr>
<tr>
<td><strong>outputWeightsToTransition_global</strong></td>
<td><em><em>vector&lt;MarkingVector</em>&gt;</em>*</td>
<td>Represents the output of the original PetriNet.</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><strong>place_i</strong></td>
<td><strong>int</strong></td>
<td>Represents one of the variables passed to the function that computes the path for reduction.</td>
</tr>
<tr>
<td><strong>place_reduced</strong></td>
<td><strong>vector&lt;int&gt;</strong></td>
<td>Assigns the value 1 corresponding to a place that satisfies the condition to be removed. This vector is cleared after every reduced net is computed.</td>
</tr>
<tr>
<td><strong>place_reduced_global</strong></td>
<td><strong>vector&lt;int&gt;</strong></td>
<td>Assigns the value 1 corresponding to a place that satisfies the condition to be removed for the original PetriNet.</td>
</tr>
<tr>
<td><strong>placestobereduced</strong></td>
<td><strong>int</strong></td>
<td>Increments every time a value 1 is assigned to <strong>places_reduced</strong>.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>transition_j</strong></td>
<td><strong>int</strong></td>
<td>Represents one of the variables passed to the function that computes the path for reduction.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>transitions_reduced</strong></td>
<td><strong>vector&lt;int&gt;</strong></td>
<td>Assigns the value 1 corresponding to a transition that satisfies the condition to be removed. This vector is cleared after every reduced net is computed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>transitions_reduced_global</strong></td>
<td><strong>vector&lt;int&gt;</strong></td>
<td>Assigns the value 1 corresponding to a transition that satisfies the condition to be removed for the original PetriNet.</td>
</tr>
</tbody>
</table>
Table 6.1 (cont.)

<table>
<thead>
<tr>
<th>transitionstobereduced</th>
<th>int</th>
<th>Increments everytime a value 1 is assigned to transitions_reduced.</th>
</tr>
</thead>
</table>

| controllableTransitions | ✓     |
| ✓ flagControllabilityCheck |
| ✓ incidenceMatrix       |
| ✓ initialMarking        |
| ✓ inputWeightsToTransition |
| ✓ inputWeightsToTransition_global |
| ✓ multiple_transition   |
| ✓ noOfPlaces            |
| ✓ noOfPlaces_global     |
| ✓ noOfTransitions       |
| ✓ noOfTransitions_global|
| ✓ outputWeightsFromTransition |
| ✓ outputWeightsFromTransition_global |
| ✓ place_i               |
| ✓ places_reduced        |
| ✓ places_reduced_global |
| ✓ placesstobereduced    |
| ✓ resultFile            |
| ✓ transition_j          |
| ✓ transitionFireCount   |
| ✓ transitions_reduced   |
| ✓ transitions_reduced_global |
| ✓ transitionstobereduced|
| ✓ uncontrollableIncidence|

Figure 6.6: Class structure - variables of class Reduction

Methods and Implementations

This section describes the member functions of the Reduction class and their implementation. The Figure 6.7 and Table 6.2 shows the list and the description of the member functions of this class.

We begin with functions that were used in the original implementation [14]. The loadInputData() method initializes IN, OUT, m0, T_u and computes the incidence matrix C of the net. Two print methods printInputsToConsole() and
printControllableTransitions() have been included with overloads for std::out for printing the inputs to the code.

The reduced input matrix \( \mathbf{IN} \) and output matrix \( \mathbf{OUT} \) are computed in primarily three modules.

1. Identifying the places and the transitions that can be removed.

2. Identifying the path \( p_a \to t_m \to p_b \to t_n \) or \( t_a \to p_m \to t_b \to p_n \) where every place and transition were identified in the above procedure.

3. Computing the reduced input and output matrix following the elimination of the identified path.

The member functions that correspond to each of these procedures are \texttt{Reduc()}, \texttt{reduction\_path()} and \texttt{reduction\_matrix()} or \texttt{reduction\_matrix\_t()} respectively.

The member function \texttt{Reduc()} is used to compute the places and the transitions that can be eliminated. If there is exactly one arc from and to a place, then that place is identified as a place that can be reduced or eliminated and the value 1 is assigned to the corresponding place in the \texttt{std::vector places\_reduced}. In a similar manner the value 1 is assigned to the corresponding transition in the \texttt{std::vector transitions\_reduced} provided there is exactly one arc to and from the transition. Following these computations the member functions \texttt{pathstobereduced()} and \texttt{reduction\_path()} are invoked. The \texttt{std::vector, places\_reduced} and \texttt{transitions\_reduced} are cleared at the start of the \texttt{Reduc()} function since with every call of this function every reduced PN is treated as a new net. The code flow for this procedure is represented by Figure 6.8.

To compute if there is a path \( p_a \to t_m \to p_b \to t_n \) or \( t_a \to p_m \to t_b \to p_n \) that can be eliminated the function \texttt{pathstobereduced()} is defined. For every place \( p_b \) identified in function \texttt{Reduc()} that can be eliminated we look for a path \( p_a \to t_m \to p_b \to t_n \) such that the \texttt{std::vector places\_reduced} has the value 1 corresponding to \( p_b \) and the \texttt{std::vector transitions\_reduced} has the value 1 corresponding to \( t_m \). A similar procedure is followed for every transition \( t_b \) identified in the function \texttt{Reduc()}. This function also computes if two transitions can be merged and is invoked only once for the input PN. All the possible paths that can be reduced for an input PN are identified and printed in the console.
The member function reduction_path() is similar to pathstobereduced() except that once a path \( p_a \rightarrow t_m \rightarrow p_b \rightarrow t_n \) is identified it invokes the function reduction_matrix() and the function reduction_matrix_t() for a path \( t_a \rightarrow p_m \rightarrow t_b \rightarrow p_n \). This function is called iteratively until there is no path that can be reduced. The code flow for this procedure is represented by Figure 6.9.

The functions reduction_matrix() and reduction_matrix_t() compute the new input and output matrix for a path that can be reduced. The algorithm to compute the new input and output matrix is as illustrated:
• For path $p_a \xrightarrow{w_1} t_m \xrightarrow{w_2} p_b \xrightarrow{w_3} t_n$ the reduced path is computed as $\tilde{p}_a \xrightarrow{\frac{w_1 w_3}{w_2}} t_n$

• For path $t_a \xrightarrow{w_1} p_m \xrightarrow{w_2} t_b \xrightarrow{w_3} p_n$ the reduced path is computed as $t_a \xrightarrow{\frac{w_1 w_3}{w_2}} \tilde{p}_n$

Following the identification of a path $p_a \rightarrow t_m \rightarrow p_b \rightarrow t_n$ in reduction_path(), the function reduction_matrix() is called with parameters $p_a$ and $t_m$. This function implements the algorithm above to modify noOfPlaces, noOfTransitions, inputWeightsToTransition, outputWeightsFromTransition and initialMarking. The modified matrices and values are printed on the console accordingly. Subsequently the member function `Reduc()` is invoked where the variables are reset and the current reduced PN repeats all the above procedures. This continues iteratively until there exists no path that can be reduced.

The variables noOfPlaces, noOfTransitions, inputWeightsToTransition, outputWeightsFromTransition and initialMarking are modified only if $w_1 \ast w_3$ is divisible by $w_2$. If the above condition is not satisfied the function reduction_path() is invoked. The intention is to look for a new path and repeat the steps again. The code flow for this procedure is represented by Figure 6.10. The function reduction_matrix_t() is similar to the function reduction_matrix() and is implemented following the identification of a path $t_a \rightarrow p_m \rightarrow t_b \rightarrow p_n$. The code flow for this procedure is represented by Figure 6.11.

```
If (W1 * W3) is divisible by W2
    reduction_matrix()
Else
    reduction_path()
Reduc()
```

Figure 6.10: Flowchart of reduction_matrix() method

```
If (W1 * W3) is divisible by W2
    reduction_matrix_t()
Else
    reduction_path()
Reduc()
```

Figure 6.11: Flowchart of reduction_matrix_t() method
The Figure 6.12 represents the flowchart for member functions of class Reduction.

![Flowchart of member functions of class Reduction](image)

Figure 6.12: Flowchart of member functions of class Reduction

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Returntype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>loadInputData()</code></td>
<td>Assigns the PN’s inputs to the corresponding members.</td>
<td><code>char * : input file name</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>printControllableTransitions()</code></td>
<td>Print method to write the set of controllable transitions to the console.</td>
<td><code>void</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>printInputsToConsole()</code></td>
<td>Print method to write the inputs to the console.</td>
<td><code>void</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>Reduc()</code></td>
<td>Compute the places and transitions that can be eliminated</td>
<td><code>int noOfPlaces, int noOfTransitions</code></td>
<td><code>void</code></td>
</tr>
</tbody>
</table>
6.3 Deducing Minimal Elements of the Original PN

This section describes the object-oriented implementation for deducing the minimal elements of the original PN. The input to this implementation is the incidence matrix of the original PN and the minimal elements of the reduced PN. In addition, the input to this implementation also consists of user defined inputs listing the places and transitions of the original PN that have been merged and removed. This implementation deduces the minimal elements of the original PN.
6.3.1 Class Definitions and Diagram

There are three main classes in this implementation PetriNet, Minele and MarkingVector. PetriNet.h is the header file that contains the declaration for the PetriNet class and its variables and functions. PetriNet.h also includes the header file Markingvector.h. Class PetriNet has one or more objects of the Class MarkingVector and hence is marked by a “Has-a” relationship with MarkingVector. Class PetriNet characterizes the input PN. Minele.h is the header file that contains the declaration for the Minele class and its variables and functions. Minele.h also includes the header files Markingvector.h and PetriNet.h. Class Minele has one or more objects of the Class MarkingVector and the Class PetriNet and hence is marked by a “Has-a” relationship with MarkingVector and PetriNet. Class PetriNet and Class Minele are tightly coupled with each other and are declared as friends of each other, allowing both the classes to access each others private variables.

Marking.h contains the declaration for the MarkingVector class and its variables and functions. The Figure 6.13 below shows a diagram of the class. The classes and the functions are described in detail in the sections that follow.

![Class Diagram for Deducing Minimal Elements](image-url)

Figure 6.13: Class Diagram for Deducing Minimal Elements
6.3.2 Class **MarkingVector**

The original implementation of the **MarkingVector** class [5] is retained in this implementation. For description on this class refer to section 6.2.2.

6.3.3 Class **PetriNet**

Class **PetriNet** characterizes the input PN. To solve for LESP for any PN, the inputs that need to be provided by the user are number of places $m$, number of transitions $n$, the input matrix (IN), the output matrix (OUT) and the initial marking $m^0$. In the original implementation these are stored in the members noOfPlaces, noOfTransitions, inputWeightsToTransition, outputWeightsFromTransition and initialMarking respectively. For a Controlled PetriNet, there is an additional input which corresponds to the transitions that are controllable. The input is given as a switch with 1 and 0 denoting controllable transitions and uncontrollable transitions respectively.

The members inputWeightsToTransition and outputWeightsFromTransition are defined as `std::vector` of pointers to objects of type **MarkingVector**. The member incidenceMatrix represents the Incidence Matrix $C$ that characterizes a PN. incidenceMatrix is defined as `std::vector` of pointers to objects of type **MarkingVector**. The loadInputData() method initializes IN, OUT, $m^0$, $T_u$ and computes the incidence matrix $C$ of the net. Two print methods printInputsToConsole() and printControllableTransitions() have been included with overloads for `std::out` for printing the inputs to the code. The Figure 6.14 shows the structure of this class.

6.3.4 Class **Minele**

The **Minele** class implements the algorithm to deduce the minimal elements of the original PN from the minimal elements of the reduced PN. Class **PetriNet** and Class **Minele** are tightly coupled with each other and are declared as friends of each other, allowing both the classes to access each others private variables. Hence, member variables of **PetriNet** such as inputWeightsToTransition and outputWeightsFromTransition can be accessed from **Minele**.

To deduce the minimal elements of the original PN, the inputs that need to be provided are minimal elements of the reduced PN and the number of
places in the reduced PN. These are represented by *minimal elements* and *noOfplace_reduced* respectively. Additionally, the input file for the original PN and a list of places that have been removed from the original PN are required inputs. The input file for the original PN is characterized in Class *PetriNet*. The global variable *Placesremoved* is a `std::vector` that stores the places that have been reduced in the original PN. The members *finalmineles* and *finalmineles_global* are defined as `std::vector` of pointers to objects of type *MarkingVector*. The member variable *finalmineles* is subject to changes through the code and is used to compute the final minimal elements of the original PN. The member variable *finalmineles_global* is assigned to omit the duplicates generated by *finalmineles* and stores all the final minimal elements of the original PN.

The member function `loadInputData1()` initializes the set of minimal elements of the reduced PN. Two print methods `printInputs1()` and `printInputs2()` have been included for printing the output of the code. `printInputs2()` has an additional segment to omit all the duplicates while printing the final result.

The minimal elements of the original PN are deduced using the member function `Deducing_for_Reduction()`. This function invokes `Addingzeros()` which loads the value 0 to the corresponding places that have been removed.
This method is in fact one integer solution to equation 6.1 or equation 6.2 whichever is being solved. Addingzeros() uses the input parameter Placesremoved which consists of the places that have been removed from the original PN and loads the value 0 corresponding to these places in the set of minimal elements obtained for the reduced PN. The member printInputs1() is invoked within this member function to print the set of minimal elements obtained by this method. Following the user-input for the rule that is followed the function Deducing_Rule_1() or Deducing_Rule_2() is invoked appropriately. If the input for places merged is anything other than the value 1 the method Merge() is invoked. Deducing_Rule_1() uses 6.1 to alter std::vector finalmineles while Deducing_Rule_2() and Merge() use 6.2 and 6.4 respectively to change std::vector finalmineles. All possible integer solutions are computed within each of these methods following which the member printInputs2() is invoked to print the minimal elements obtained in each of these methods. The Figure 6.15 shows the structure of this class and Table 6.3 shows the list and the description of the member functions of this class while Table 6.4 describes the variables of this class.

![Diagram](image)

Figure 6.15: Class structure of Minele
Table 6.3: Method definitions of class *Minele*

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Parameter</th>
<th>Returntype</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>loadInputData_1()</code></td>
<td>Initializes the set of minimal elements for the reduced PN.</td>
<td><code>char *</code>: input file name</td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>printInputs1()</code></td>
<td>Print method to write the output to the console.</td>
<td><code>void</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>printInputs2()</code></td>
<td>Print method to write the output to the console by omitting the duplicates.</td>
<td><code>void</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>Addingzeros()</code></td>
<td>Loads the value 0 to the corresponding places that have been removed.</td>
<td><code>vector&lt;int&gt;</code> Placesremoved, <code>const PetriNet</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><code>Deducing_for_Reduction()</code></td>
<td>Invokes appropriate function for Rule 1, 2 or 3</td>
<td><code>vector&lt;int&gt;</code> Rulefollowed, <code>vector&lt;int&gt;</code> Placesremoved, <code>vector&lt;int&gt;</code> Transitionremoved, <code>vector&lt;int&gt;</code> Placesmerged, <code>const PetriNet</code></td>
<td><code>void</code></td>
</tr>
</tbody>
</table>
### Table 6.3 (cont.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Parameters</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Deducing_Rule_1()</strong></td>
<td>Uses equation 6.1 to compute all possible integer solutions</td>
<td><code>int T_removed, int P_removed, vector&lt;int&gt; Places_removed, const PetriNet</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><strong>Deducing_Rule_2()</strong></td>
<td>Uses equation 6.2 to compute all possible integer solutions</td>
<td><code>int T_removed, int P_removed, vector&lt;int&gt; Places_removed, const PetriNet</code></td>
<td><code>void</code></td>
</tr>
<tr>
<td><strong>Merge()</strong></td>
<td>Uses equation 6.4 to compute all possible integer solutions</td>
<td><code>int Merged-place1, int Merged-place2, const PetriNet</code></td>
<td><code>void</code></td>
</tr>
</tbody>
</table>

### Table 6.4: Description of variables used in class Minele

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>finalmineles</td>
<td><code>vector&lt;MarkingVector*&gt;</code></td>
<td>Represents the minimal elements of the original PN and is subject to changes through the code.</td>
</tr>
<tr>
<td>finalmineles_global</td>
<td><code>vector&lt;MarkingVector*&gt;</code></td>
<td>Used to omit all the duplicate members.</td>
</tr>
<tr>
<td>noOfplace_reduced</td>
<td><code>int</code></td>
<td>Represents the number of places of the reduced PetriNet.</td>
</tr>
</tbody>
</table>
Table 6.4 (cont.)

<table>
<thead>
<tr>
<th>$P_{removed}$</th>
<th>int</th>
<th>Represents the place that has been removed that is used in the current computation step.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_{removed}$</td>
<td>int</td>
<td>Represents the transition that has been removed that is used in the current computation step.</td>
</tr>
</tbody>
</table>

6.4 Examples

This section gives some illustrations of the implemented reduction algorithm, the input and the output files that are generated. The input file format for the reduction algorithm implementation is consistent with [14] (cf. figure 5.3).

The output file has the same format as the input file and is saved with an extension _a.txt to the input file name. This file is used to deduce the LESP for the reduced PN. The final minimal elements obtained is saved in a file with an extension _output.txt to the input file name.

The implementation to deduce minimal elements for the original PN requires two input files:

1. The input file used in the reduction algorithm implementation - the incidence matrix of the original PN

2. The file with the extension _output.txt - the final minimal elements of the reduced PN

In addition, this implementation also has user-defined inputs

- The list of places that have been removed from the original net - integer values only
- The list of transitions that have been removed from the original net - integer values only
• The places in the original PN that have been merged to obtain the reduced PN - integer values only

• The rule followed to reduced the PN - enter value 1 for rule 1 or 2 for rule 2

• The integer value for the total number of places of the reduced PN

The code uses the value $-1$ to disable or exit the user-input, that is, after listing all the places that have been removed from the original PN in order to move to the list of transitions the user has to type $-1$. The flow of the implementation is illustrated in the Figure 6.40.

![Flow of Implementation](image)

Figure 6.16: Flow of Implementation
6.4.1 Illustrations

Figure 6.17: Example-1

Figure 6.18: Input file for Example-1
Initial Marking: (2 0 0 0)

Inputs:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>.</td>
<td>3</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Outputs:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>.</td>
<td>3</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transition T3 and place P2 can be removed
Place P2 and transition T4 can be removed
Place P4 and transition T6 can be removed

-----------------------------------------------------------------

new initial Marking(2 0 0 0)
new Input:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

new Output:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>.</td>
<td>3</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>.</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

-----------------------------------------------------------------

new initial Marking(2 0 0 0)
new Input:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td></td>
</tr>
</tbody>
</table>

new Output:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>1</td>
<td>3</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>.</td>
<td>2</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

-----------------------------------------------------------------

new initial Marking(2 0 0 0)
new Input:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>2</td>
<td>2</td>
<td>.</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

new Output:

<table>
<thead>
<tr>
<th>T</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>1</td>
<td>3</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>.</td>
<td>2</td>
<td>.</td>
</tr>
</tbody>
</table>

Figure 6.19: Output for Reduction part of the algorithm for Example-1

Figure 6.20: Reduced Example-1
Incidence Matrix:

\[
\begin{array}{cccc}
T & 1 & 2 & 3 & 4 \\
1 & -2 & 3 & -2 & 1 \\
2 & 3 & -2 & 2 & -2 \\
\end{array}
\]

Initial Marking: (2 0)

There is an LESP for this (fully controlled) PN.

---

Minimal Elements of the fully controlled Net:

1: (0 2)
2: (2 0)

List of Controllable Transitions:

\[ t_4 \]

(Final) Minimal Elements of the control-invariant set:

1: (0 2)
2: (2 0)

checking if fine

This is An LESP

Elapsed Time: 5.278 secs

---

Figure 6.21: Minimal Elements of the Reduced net for Example-1

---

Places removed:

2
4
-1

Transitions removed in the same order:

4
6
-1

Places merged:

-1

Rule followed in same order: Enter 1 for Rule 1 or 2 for Rule 2:

2
2

Number of places for reduced net:

2

No of reduced places 2

Open file name: lc_output.txt

(0 2)

(2 0)

Initial Marking: (2 0 0 0)

Inputs:

\[
\begin{array}{cccccc}
T & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & Z & Z & \ldots & \ldots & \ldots & \ldots \\
2 & \ldots & 3 & \ldots & \ldots & \ldots \\
3 & \ldots & Z & \ldots & \ldots & \ldots \\
4 & \ldots & \ldots & 1 & \\
\end{array}
\]

Outputs:

\[
\begin{array}{cccccc}
T & 1 & 2 & 3 & 4 & 5 & 6 \\
1 & . & Z & \ldots & \ldots & \ldots & \ldots \\
2 & \ldots & 3 & \ldots & \ldots & \ldots \\
3 & \ldots & Z & \ldots & \ldots & \ldots \\
4 & \ldots & \ldots & 1 & \\
\end{array}
\]

Controllable transitions(0 0 0 1 0)

Number of places 4

New final elements:

(0 0 0 0)

(2 0 0 0)

(0 1 0 0)

(0 0 0 1)

(1 0 0 1)

Figure 6.22: Deducing minimal elements for Example-1
Figure 6.23: Example-2

\[
\begin{pmatrix}
3 & 4 \\
1 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 \\
0 & 0 & 2 & 1 \\
0 & 0 & 1 & 0 \\
4 & 0 & 0 & 0 \\
0 & 4 & 0 & 0 \\
1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

Figure 6.24: Input file for Example-2

```
./reduction_finalversion Example_2.txt
Initial Marking : (1 0 0)
Inputs :
T 1 2 3 4
P
1 1 . .
2 . 2 .
3 . 2 1

Outputs :
T 1 2 3 4
P
1 . 1 .
2 4 . .
3 . 4 .

Transition T1and place P2 can be removed
Place P2 and transition T2 can be removed

new initial Marking(1 0 0)
new Input :
T 1 2 3
P
1 1 .
2 . 2 1

new Output :
T 1 2 3
P
1 . 1 .
2 8 .
```

Figure 6.25: Output for Reduction part of the algorithm for Example-2
Figure 6.26: Reduced Example-2

\[\text{./PN_minele Example_2.txt_a.txt}\]

Incidence Matrix:

\[
\begin{array}{c|ccc}
T & 1 & 2 & 3 \\
P & 1 & -1 & 1 . \\
   & 2 & 8 & -2 -1 \\
\end{array}
\]

Initial Marking: \((1 \ 0)\)

There is an LESP for this (fully controlled) PN

---------------------------------------------------------------------

Minimal Elements of the fully controlled Net

---------------------------------------------------------------------

1: \((1 \ 0)\)

2: \((0 \ 2)\)

List of Controllable Transitions

---------------------------------------------------------------------

t3

(Final) Minimal Elements of the control-invariant set

---------------------------------------------------------------------

1: \((1 \ 0)\)

2: \((0 \ 2)\)

checking if fine
This is an LESP

Figure 6.27: Minimal Elements of the Reduced net for Example-2
Figure 6.28: Deducing minimal elements for Example-2

Figure 6.29: Example-3
Figure 6.30: Input file for Example-3

```
./reduction_finalversion wtnew5.txt
Initial Marking : ( 1 0 0 0 0  )
Inputs :
    T  1  2  3  4  5  6  7
 P
   1   . . . . .
   2   .   1  2  . . .
   3   . . . 3  1 .
   4   . . . . . . 6
   5   . . . . . . 6

Outputs :
    T  1  2  3  4  5  6  7
 P
   1   . . . . . 1
   2   1  1 . . . 1
   3   . 1  1 . . .
   4   . . 3 . . .
   5   . . . 2 . .
Transition T4 and place P4 can be removed
Transition T5 and place P5 can be removed
-----------------------------------------------------------------
new initial Marking( 1 0 0 0  )
new Input :
    T  1  2  3  4  5  6
 P
   1   . . . . 1
   2   .   1  .  4
   3   . . . 1 .
   4   . . . . 2
-----------------------------------------------------------------
new Output :
    T  1  2  3  4  5  6
 P
   1   . . . . 1
   2   1  1 . . 1
   3   . 1  1 .
   4   . . 2 .
```

Figure 6.31: Output for Reduction part of the algorithm for Example-3
Figure 6.32: Reduced Example-3

./PN_minele wtnew5.txt_a.txt

Incidence Matrix:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>-1</td>
<td>*S</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>.</td>
<td>-3</td>
</tr>
<tr>
<td>P3</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-9</td>
</tr>
</tbody>
</table>

Initial Marking: (1 0 0)

There is an LESP for this (fully controlled) PN

------------------------------------------------------------------------

Minimal Elements of the fully controlled Net

------------------------------------------------------------------------

1: (1 0 0)
2: (0 5 8)
3: (0 4 9)
4: (0 6 7)
5: (0 7 6)
6: (0 8 5)
7: (0 9 4)
8: (0 10 3)
9: (0 11 2)
10: (0 12 1)
11: (0 13 0)

List of Controllable Transitions

-----------------------------------

t1 t4

(Final) Minimal Elements of the control-invariant set

------------------------------------------------------------------------

1: (1 0 0)

checking if fine
This is An LESP

Figure 6.33: Minimal Elements of the Reduced net for Example-3
Figure 6.34: Deducing minimal elements for Example-3
Figure 6.35: PN-9 (cf. [5]).

Figure 6.36: Input file for PN-9 (cf. [5]).
Initial Marking : ( 1 0 0 0 0 0 0 0 0 0 )

Inputs :
T  1  2  3  4  5  6  7  8  9 10 11
P
1   1  1  .  1  .  .  .  .  .  .  .
2   .  1  .  1  .  .  .  .  .  .  .
3   .  .  .  1  .  .  .  .  .  .  .
4   .  .  .  .  1  .  .  .  .  .  .
5   .  .  .  .  .  1  1  .  .  .
6   .  .  .  .  .  .  .  1  .  .
7   .  .  .  .  .  .  .  .  1  .
8   .  .  .  .  .  .  .  .  .  1
9

Outputs :
T  1  2  3  4  5  6  7  8  9 10 11
P
1   .  .  .  .  .  .  .  .  .  .  1
2   1  .  .  .  .  .  .  .  .  .  .
3   .  .  1  .  .  .  .  .  .  .  .
4   .  .  1  .  .  .  .  .  .  .  .
5   .  .  .  1  .  .  .  .  .  .  .
6   .  .  .  .  1  .  .  .  .  .  .
7   .  .  .  .  .  1  .  .  .  .  .
8   .  .  .  .  .  1  .  .  .  .  .
9   .  .  .  .  .  .  1  1

Place P3 and transition T5 can be removed
Place P7 and transition T9 can be removed
Place P8 and transition T10 can be removed
transitions to place 9 can be merged

Figure 6.37: Output for Reduction part of the algorithm for PN-9 (cf. [5]).
Figure 6.38: Reduced PN-9 (cf. [5]).
Incidence Matrix:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1</td>
<td>.</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>-1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Initial Marking: (100000)

There is an LESP for this (fully controlled) PN.

Minimal Elements of the fully controlled Net

1: (100000)
2: (000010)
3: (000001)
4: (001100)
5: (011000)

List of Controllable Transitions

t1 t2 t3 t4 t7

(Final) Minimal Elements of the control-invariant set

1: (100000)
2: (000000)
3: (001100)
4: (011000)

checking if fine

The loop-test failed for the minimal_element: (100000)

The loop-test failed for the minimal_element: (000000)

(Final) Minimal Elements of the control-invariant set

1: (001100)
2: (011000)
3: (100100)
4: (010000)
5: (100001)
6: (100010)
7: (100100)
8: (101000)
9: (010001)
10: (001001)
11: (000101)
12: (000011)
13: (000002)
14: (000002)

checking if fine

This is An LESP

Figure 6.39: Minimal Elements of the Reduced net for PN-9 (cf. [5]).

80
Figure 6.40: Deducing minimal elements for PN-9 (cf. [5]).
Figure 6.41: PN-13 (cf. [5]).

```
9 10
1 0 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 1 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1
```

Figure 6.42: Input file for PN-13 (cf. [5]).
**Figure 6.43: Output for Reduction part of the algorithm for PN-13 (cf. [5]).**
Figure 6.44: Reduced PN-13 (cf. [5]).
Incidence Matrix:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>-1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>*S</td>
</tr>
<tr>
<td>6</td>
<td>.</td>
<td>.</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>-1</td>
<td>.</td>
</tr>
</tbody>
</table>

Initial Marking: (2 0 0 0 0 0)

There is an LESP for this (fully controlled) PN

---

Minimal Elements of the fully controlled Net

1: (0 0 0 1 0)
2: (1 0 0 1 0)
3: (0 1 0 1 0)
4: (0 0 1 1 0)
5: (1 0 1 0 0)
6: (0 1 1 0 0)
7: (0 0 2 0 0)
8: (1 1 0 0 0)
9: (0 2 0 0 0)
10: (2 0 0 0 0)

List of Controllable Transitions

- t4

(Final) Minimal Elements of the control-invariant set

1: (0 0 0 1 0)
2: (0 0 1 1 0)
3: (0 0 2 0 0)

checking if fine
This is an LESP

Figure 6.45: Minimal Elements of the Reduced net PN-13 (cf. [5]).
Figure 6.46: Deducing minimal elements for PN-13 (cf. [5]).
Figure 6.47: PN-11 (cf. [5]).

Figure 6.48: Input file for PN-11 (cf. [5]).
Figure 6.49: Output for Reduction part of the algorithm for PN-11 (cf. [5]).
Figure 6.50: Reduced PN-11 (cf. [5]).
Incidence Matrix:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>-1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>-1</td>
<td>-1</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>-1</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>7</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
</tr>
</tbody>
</table>

Initial Marking: (2 0 0 0 0 0)

There is an LESP for this (fully controlled) PN.

Minimal Elements of the fully controlled Net
--------------------------------------------

1: (1 0 0 0 0 0 1)
2: (0 1 0 0 0 0 1)
3: (0 0 1 0 0 0 1)
4: (0 0 0 1 0 0 1)
5: (0 0 0 0 1 0 1)
6: (0 0 0 0 0 1 1)
7: (0 0 0 0 0 0 2)
8: (1 0 0 0 0 1 0)
9: (0 1 0 0 0 1 0)
10: (0 0 1 0 0 1 0)
11: (0 0 0 1 1 0 0)
12: (0 0 0 0 2 0 0)
13: (1 0 0 1 0 0 0)
14: (0 1 0 1 0 0 0)
15: (0 0 1 1 0 0 0)
16: (0 0 0 2 0 0 0)
17: (0 0 0 1 0 0 0)
18: (1 0 0 1 0 0 0)
19: (0 1 0 1 0 0 0)
20: (0 0 1 1 0 0 0)
21: (0 0 0 2 0 0 0)
22: (0 1 0 1 0 0 0)
23: (0 1 1 0 0 0 0)
24: (0 0 1 1 0 0 0)
25: (0 2 0 0 0 0 0)
26: (0 0 2 0 0 0 0)

List of Controllable Transitions
--------------------------------

(1 2 3)

(Final) Minimal Elements of the control-invariant set
-----------------------------------------------------

1: (1 0 0 0 0 1)
2: (0 1 0 0 0 1)
3: (0 0 1 0 0 1)
4: (0 0 0 1 0 1)
5: (0 0 0 0 1 1)
6: (0 0 0 0 0 1)
7: (0 0 0 0 0 0 2)
8: (1 0 0 0 0 1 0)
9: (0 1 0 0 0 1 0)
10: (0 0 1 0 0 1 0)
11: (0 0 0 1 0 1 0)
12: (0 0 0 0 1 1 0)
13: (0 0 0 0 0 1 1)
14: (1 0 0 0 1 0 0)
15: (0 1 0 0 1 0 0)
16: (0 0 1 0 1 0 0)
17: (0 0 0 1 1 0 0)
18: (1 0 0 1 0 0 0)
19: (0 1 0 1 0 0 0)
20: (0 0 1 1 0 0 0)
21: (0 0 0 2 0 0 0)
22: (0 1 0 1 0 0 0)
23: (0 1 1 0 0 0 0)
24: (0 0 1 1 0 0 0)
25: (0 2 0 0 0 0 0)
26: (0 0 2 0 0 0 0)

checking if fine
This is an LESP

Figure 6.51: Minimal Elements of the Reduced net for PN-11 (cf. [5]).
Figure 6.52: Deducing minimal elements for PN-11 (cf. [5]).
6.4.2 Discussion

The example Figure 6.35 takes unusually long time to compute the LESP using the existing software [5]. The computational time can be reduced considerably using the reduction techniques in this chapter. The illustrations show that the minimally restrictive LESP can be deduced for this example using the reduction techniques as seen in Figure 6.40.

One important observation to note while using these reduction techniques is that the resulting LESP need not always be minimally restrictive. This is illustrated using Example-3 Figure 6.29. The minimally restrictive LESP for this PN would yield a right-closed set with minimal elements \( \{(10\,0\,0\,0)^T, (0\,0\,6\,6\,4)^T, (0\,0\,6\,6\,2)^T, (0\,9\,6\,0)^T\} \). However, once the PN is simplified the minimally restrictive LESP for the reduced PN Figure 6.32 has only one minimal element \( \{(100)^T\} \). Using rule -1 and solving for Equation 6.1 the only possible solution would be \( \{(1\,0\,0\,0\,0)^T\} \). This is an LESP although not minimally restrictive.

The minimal elements for a PN that has been reduced using reduction technique 2 is computed using the Equation 6.2. It is important to note that while computing all possible integer solutions for \( x \) and \( y \) we take the following into consideration:

- if \( x \) is not equal to 0 , \( x = \max(x, w_2) \).

The importance of replacing \( x \) with \( w_2 \) when \( x < w_2 \) and \( x \) is not equal to 0 was discussed in the previous section. To demonstrate this further, let us look at the example illustrated in Figure 6.53. The path \( t_2 \rightarrow p_1 \rightarrow t_1 \rightarrow p_2 \) is replaced by \( t_2 \rightarrow 8 \rightarrow \bar{p}_2 \).

Here, \( w_1 = 4, w_2 = 2 \) and \( w_3 = 4 \). The minimally restrictive LESP for the reduced PN would be \( \{(2)^T\} \). Using Equation 6.2 we get,

\[
2 = y + \frac{4}{2} x.
\]

If we do not take the condition \( x = \max(x, w_2) \) when \( x \neq 0 \) into consideration then, all possible integer solutions for \( (x, y) \) would yield \( (1, 0) \) and \( (2, 0) \) as the minimal elements of the original PN. But \( (1, 0) \) cannot be a minimal element since it does not enforce liveness. However, if we were to replace \( x \) with \( \max(x, w_2) \) then for the solution \( (1, 0) \) the minimal element would be \( (\max(1, w_2(= 2)), 0) \) = \( (2, 0) \). This has been incorporated in the object-
oriented implementation to compute the minimal elements of the original PN.

Figure 6.53: Example-5

To space limitations we have not included large examples in this thesis. The largest PN model that the software described in this thesis was used on was an unbounded PN with eleven places and fifteen transitions with a coverability graph that had $\approx 10^7$ vertices. The $\Delta(N)$-set for this PN had forty-one minimal elements that were computed in less than a second of run-time on a Macbook Air.

The fact that the software described in this thesis can synthesize minimally restrictive LESP$s$ for unbounded PNs is an important feature that distinguishes the presented work from those that exist in the literature. As per reference [29], the results in this thesis can serve serve as critical milestones in the synthesis of asymptotically efficient LESP synthesis procedures for large PN models.
CHAPTER 7

CONCLUSIONS AND FUTURE WORK

We identified two classes of general PN structures, $\mathcal{F}$ and $\mathcal{H}$ where the existence of LESP for an instance initialized at a marking is sufficient to conclude that there is an LESP when the same instance is initialized at a larger marking (cf. sections 4.1 and 4.2). An object-oriented implementation of an algorithm that computes the members of $\min(\Delta(N))$ for any member of the $\mathcal{F}$ and $\mathcal{H}$ classes of the PNs (cf. chapters 5). We identified examples where the software of reference [5] takes an unusually long time to compute the minimally restrictive LESP for specific problem instances. We developed reduction techniques (cf. chapter 6) and other methods (cf. sections 4.3 and 4.4) to improve the performance of the software of reference [5]. Using several illustrative examples that are interspersed in this thesis, we have shown the utility of the various results obtained in course of this research.

The reduction techniques that have been developed reduce the computational time for computing LESP for PNs. However, the LESP deduced using these techniques are not always minimally restrictive. We suggest investigations into deducing minimally restrictive LESP for PNs using reduction techniques as a direction of possible future research. The techniques that have been developed exploit the property of similarity between the reduced PN and the original PN. This thesis covers three such reduction techniques. However, other techniques could be investigated as another direction of future research.

The object-oriented implementation of reduction techniques (cf. chapter 6) that were developed uses similar structure and some of the variables used in [14] under the assumption that an integration to the existing code in the future would be faster. However, all the steps involved in deducing the minimal elements i.e. reducing the PN, computing the minimal elements of the reduced PN and finally deducing the minimal elements of the original PN have not yet been made transparent to the user. One possible direction for
future research would be to develop the software integrating the reduction techniques in this thesis along with additional techniques with the existing software to make the entire computation transparent to the user.

The contributions of this thesis is limited to the paradigm of marking-based LESPs for PNs. There are other paradigms for liveness enforcement in PNs. For instance, references [30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40] deal with various aspects of event-based supervisory control of DEDS systems. We suggest investigations into event-based LESPs for PN as a future research topic.

Figure 7.1 shows two different LESPs for a PN $N_{10}(m_{10}^0)$. Policy 1 uses an event-based LESP. Policy 2 uses the $\Delta(N)$-set based LESP. The $\Delta(N)$-set based LESP of Policy 2 is minimally-restrictive (cf. chapter 3 of this thesis). Policy 1 is not minimally restrictive, as $(1 0 0 0 0 0 0 0)^T \rightarrow (t_2 t_4 t_7 t_8 t_9)^2 t_1 \rightarrow (2 1 2 0 0 0 0 0)^T$ under the supervision of Policy 1, and the firing of $t_1$ is prevented unnecessarily by this policy at marking $(2 1 2 0 0 0 0 0)^T$.

To explicate the role of the event-based LESP of Policy 1, the supervisor essentially ensures the language generated by $N_1(m_1^0)$, when projected on the alphabet-set $\{t_1, t_2\}$ is a subset of the set identified by the regular-expression $$(t_2 t_2^* t_1)^*.$$ Since,

\begin{align*}
(t_2 t_4 t_7 t_8 t_9)^2 t_1 \mid_{\{t_1, t_2\}} &= t_2^2 t_1 (\in (t_2 t_2^* t_1)^*) \text{ and } \\
(t_2 t_4 t_7 t_8 t_9)^2 t_1 \mid_{\{t_1, t_2\}} &= t_2^2 t_1 (\notin (t_2 t_2^* t_1)^*)
\end{align*}

Policy 1 does not permit the firing of the controllable transition $t_1$ at the marking $(2 1 2 0 0 0 0 0)^T$, which is unnecessary, which is the reason why this LESP is not the “best” LESP.

The implementation of LESPs (and other supervisory control policies) are susceptible to sensor-failures. We suggest investigations into the fault-tolerant implementations of LESPs, along the lines of references [41, 42], as another direction of future research.
Plant Control: Permit $t_1$ only at state $x_1$.

Observations: strings of occurrences of $t_1$ and $t_2$.

Supervisory Policy:

(a) Policy 1

(b) Policy 2

Figure 7.1: A PN $N_{10}(m_{10}^0)$ with (a) an event-based LESP (that is not minimally restrictive), and (b) A static-map based LESP that is minimally restrictive.
REFERENCES


