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

ABSTRACT

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 acknowldege 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

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-*

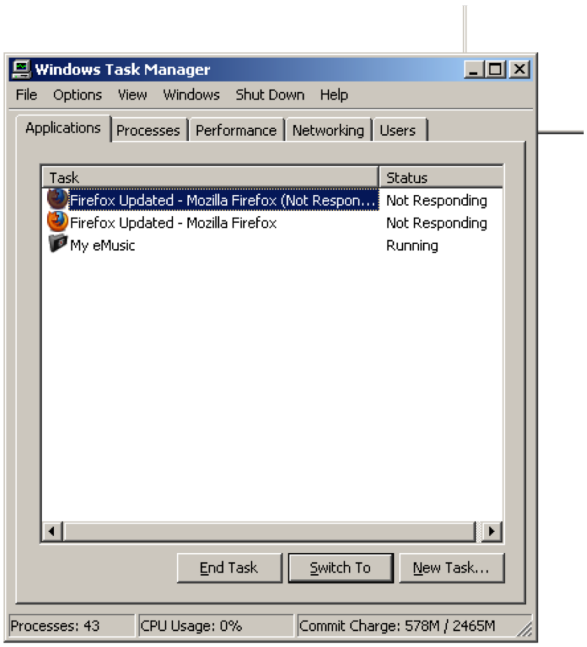


Figure 1.1: Livelocked Mozilla Firefox.

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

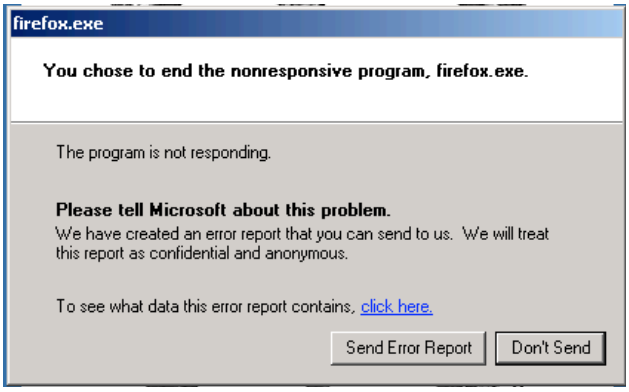


Figure 1.2: Error Reporting System used by Windows.

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) $\mathbf{m}^0$ 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 illutrative 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 $card(A)$. 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, $\mathbf{m}^0(p)$, 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 $\mathbf{x}(\sigma)$ to denote the *Parikh vector* of σ. That is, the i-th entry, $\mathbf{x}_i(\sigma)$, corresponds to the number of occurrences of transition t_i in σ.

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 $\mathbf{m}^0 : \Pi \to \mathcal{N}$, which identifies the number of *tokens* in each place. A marking $\mathbf{m} : \Pi \to \mathcal{N}$ is sometimes represented by an integer-valued vector $\mathbf{m} \in \mathcal{N}^n$,

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

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

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

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

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

$$\mathbf{IN}_{i,j} = \begin{cases} 1 & \text{if } p_i \in^\bullet t_j \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{OUT}_{i,j} = \begin{cases} 1 & \text{if } p_i \in t_j^\bullet \\ 0 & \text{otherwise} \end{cases}$$

The *incidence matrix* $\mathbf{C}$ of the PN N is an $n \times m$ matrix, where $\mathbf{C} = \mathbf{OUT} - \mathbf{IN}$. If $\mathbf{x}(\sigma)$ 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 $\mathbf{m}^i \xrightarrow{\sigma} \mathbf{m}^{i+j}$, then $\mathbf{m}^{i+j} = \mathbf{m}^i + \mathbf{C}\mathbf{x}(\sigma)$.

A set of markings $\mathcal{M} \subseteq \mathcal{N}^n$ is said to be *right-closed* [7] if $((\mathbf{m}^1 \in \mathcal{M}) \wedge$

$(\mathbf{m}^2 \geq \mathbf{m}^1) \Rightarrow (\mathbf{m}^2 \in \mathcal{M}))$, 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 $\not\exists \widetilde{P} \subset P$, such that $(\widetilde{P}^\bullet) \subseteq ({}^\bullet \widetilde{P})$ (resp. $({}^\bullet \widetilde{P}) \subseteq (\widetilde{P}^\bullet)$).

A PN structure $N = (\Pi, T, \Phi, \Gamma)$ is *Free-Choice* (FC) if $\forall p \in \Pi, (card(p^\bullet) > 1 \Rightarrow {}^\bullet(p^\bullet) = \{p\})$, where $card(\bullet)$ denotes the cardinality of the set argument. A PN $N(\mathbf{m}^0)$ 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 $N(\mathbf{m}^0)$ 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 $\mathbf{m}^0$.

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 $N(\mathbf{m}^0)$ is live for an initial marking $\mathbf{m}^0$, the ordinary FCPN $N(\widehat{\mathbf{m}}^0)$ is also live for any $\widehat{\mathbf{m}}^0 \geq \mathbf{m}^0$. 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, $N_1(\mathbf{m}_1^0)$ is live if and only if the sum of tokens assigned to all places by the initial marking $\mathbf{m}_1^0$ 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

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

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

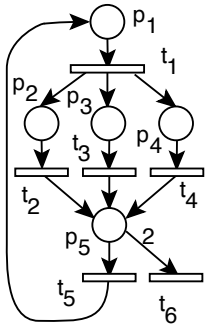


Figure 2.1: A general FCPN structure $N_1 = (\Pi_1, T_1, \Phi_1, \Gamma_1)$.

of a general SPN $N(\mathbf{m}^0)$. They note that if all siphons $P \subseteq \Pi$ of the SPN $N(\mathbf{m}^0)$ satisfy the requirement $\forall \mathbf{m} \in \Re(N, \mathbf{m}^0), \exists p \in P$ such that $\mathbf{m}(p)$ is greater than or equal to the largest weight among the arcs that originate from p, then $N(\mathbf{m}^0)$ 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 Π_2 is a siphon (and a trap). $N_2(\mathbf{m}_2^0)$ is live for any $\mathbf{m}_2^0 \neq \mathbf{0}$. Consequently, $N_2(\mathbf{m}_2^0)$ is live for $\mathbf{m}_2^0 = (1\ 0\ 0\ 0\ 0)^T$, but $\forall p \in \Pi_2$, $\mathbf{m}_2^0(p)$ is strictly less than the largest weight among all arcs that originate from p.

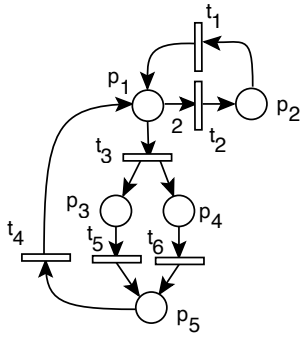


Figure 2.2: A general FCPN structure $N_2 = (\Pi_2, T_2, \Phi_2, \Gamma_2)$.

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 $\mathbf{m}^0$ 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 ω 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 ω symbols in its coverability graph. The coverability graph is finite for any PN.

Figure 2.4 represents a PN $N_3(\mathbf{m}_3^0)$ that is not bounded and not live. The reason $N_3(\mathbf{m}_3^0)$ 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 ω-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 $\mu(v_0) = \mathbf{m}^0$.
2: **for** $v_i \in V$ **do**
3: **if** $\mu(v_i)$ is identical to $\mu(v_j)$ for some $v_j \in V$ **then**
4: v_i has no children, and is marked as the *duplicate* of v_j.
5: **end if**
6: **if** no transition is enabled under the marking $\mu(v_i)$ **then**
7: v_i has no children, and is marked as a *terminal* vertex.
8: **end if**
9: **if** v_i is not a duplicate-vertex **then**
10: **for** t_j that is enabled under $\mu(v_i)$ **do**
11: Create a new vertex v_k. $V \leftarrow V \cup \{v_k\}$.
12: Create a new directed edge starting from v_i and ending at v_k. Label this edge with the transition t_j.
13: **if** The number of tokens in p is ω under $\mu(v_i)$, for some $p \in \Pi$ **then**
14: The number of tokens in p is ω under $\mu(v_k)$ too.
15: **else**
16: The number of tokens in p under $\mu(v_k)$ is what results when t_j is fired under $\mu(v_k)$
17: **end if**
18: **if** ($\exists v_q \in V$ on the directed path from v_0 to v_k such that $\mu(v_q) \leq \mu(v_k)$) **then**
19: **for** ($p \in \Pi$) **do**
20: **if** p has fewer tokens under $\mu(v_q)$ than under $\mu(v_k)$ **then**
21: The number of tokens in p is ω under $\mu(v_k)$.
22: **end if**
23: **end for**
24: **end if**
25: **end for**
26: **end if**
27: **end for**

Figure 2.3: The procedure for the construction of the *Reachability Tree* of a PN $N(\mathbf{m}^0)$, where $N = (\Pi, T, \Phi, \Gamma)$.

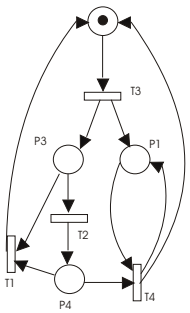


Figure 2.4: A PN $N_3(\mathbf{m}_3^0)$ that is not bounded and not live (cf. section V.C, [4]).

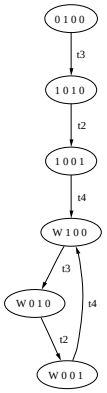


Figure 2.5: Coverability graph for the PN $N_3(\mathbf{m}_3^0)$ in 2.4. It can be seen that the transition t_1 is not fired even once.

To reiterate, while the reachable set of markings of a PN can be infinitely large, its coverability graph is always finite. The PN $N_4(\mathbf{m}_4^0)$ 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.

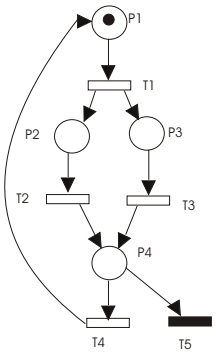


Figure 2.6: A PN $N_4(\mathbf{m}_4^0)$ with infinitely large set of markings

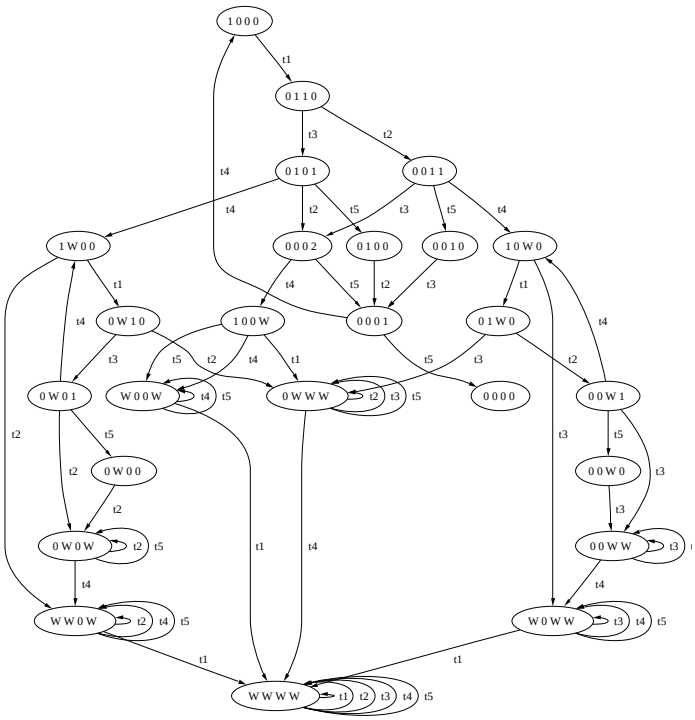


Figure 2.7: Finite coverability graph for $N_4(\mathbf{m}_4^0)$ of figure 2.6 with infinitely large set of markings

CHAPTER 3

MARKING-BASED SUPERVISORY CONTROL OF PNS

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 $\mathbf{m}^i$, only if $\mathcal{P}(\mathbf{m}^i, t_j) = 1$. A policy $\mathcal{P}$ is *marking monotone* if $\forall t_i \in T, \forall \mathbf{m}^2 \geq \mathbf{m}^1, (\mathcal{P}(\mathbf{m}^1, t_i) = 1) \Rightarrow (\mathcal{P}(\mathbf{m}^2, t_i) = 1)$.

If $t_j \in T_e(N, \mathbf{m}^i)$ for some marking $\mathbf{m}^i$, we say the transition t_j is *state-enabled* at $\mathbf{m}^i$. If $\mathcal{P}(\mathbf{m}^i, t_j) = 1$, we say the transition t_j is *control-enabled* at $\mathbf{m}^i$. 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 $\forall \mathbf{m}^i \in \mathcal{N}^n, \mathcal{P}(\mathbf{m}^i, t_j) = 1$, 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 $\mathbf{m}^i$, if, (1) $t_1 \in T_e(N, \mathbf{m}^i), \mathcal{P}(\mathbf{m}^i, t_1) = 1$, and (2) for $j \in \{1, \ldots, k-1\}$ the firing of the transition t_j produces a marking $\mathbf{m}^{i+j}$ and $t_{j+1} \in T_e(N, \mathbf{m}^{i+j})$ and $\mathcal{P}(\mathbf{m}^{i+j}, t_{j+1}) =$

1.

The set of reachable markings under the supervision of $\mathcal{P}$ in N from the initial marking $\mathbf{m}^0$ is denoted by $\Re(N, \mathbf{m}^0, \mathcal{P})$. We use the symbol $\Re(N, \mathbf{m}^0)$ to denote the set of reachable markings when the PN $N(\mathbf{m}^0)$ is unsupervised, or when the supervisory policy $\mathcal{P}$ is a trivial policy that permits every transition at all markings (as would be the case when $T = T_u$, for instance).

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

For a marking monotone supervisory policy $\mathcal{P}$, the construction procedure for the coverability graph of a PN (cf. section 2.1.1) can be extended to accommodate the supervisory policy $\mathcal{P}$, which results in the coverability graph $G(N(\mathbf{m}^0, \mathcal{P}))$ (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 $G(N(\mathbf{m}^0, \mathcal{P}))$.

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

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

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

$$\forall \mathbf{m}^i \in \mathcal{N}^n, \forall t \in T, \mathcal{P}(\mathbf{m}^i, t) \geq \widehat{\mathcal{P}}(\mathbf{m}^i, t).$$

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 $N(\mathbf{m}^0)$, then there is a unique minimally restrictive LESP for PN $N(\mathbf{m}^0)$ (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

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

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 $\Gamma(\mathcal{M}) = \{\mathbf{m}^i \in \mathcal{N}^n \mid \exists \sigma \in T_u^*, \exists \mathbf{m}^j \in \mathcal{M}, \text{ such that } \mathbf{m}^j \xrightarrow{\sigma} \mathbf{m}^i\}$. Note, $\mathcal{M} \subseteq \Gamma(\mathcal{M})$ in general. Alternately, if $\mathcal{M}$ is control-invariant with respect to N, $\mathbf{m}^i \in \mathcal{M}$, $\mathbf{m}^i \xrightarrow{\sigma} \mathbf{m}^j$ in N, and $\mathbf{m}^j \notin \mathcal{M}$, 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 $\mathbf{m}^1 \in \Delta(N), t_u \in T_e(N, \mathbf{m}^1) \cap T_u$ and $\mathbf{m}^1 \xrightarrow{t_u} \mathbf{m}^2$ in N, then $\mathbf{m}^2 \in \Delta(N)$. 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 $\mathbf{m}^0 \in \Delta(N)$, 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 $N(\mathbf{m}^0)$. 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) Ψ 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 Ψ if its firing would result in a new marking that is not in Ψ, and (4) $\mathbf{m}^0 \in \Psi$, we can construct the coverability graph, $G(N(\mathbf{m}^0), \mathcal{P}_\Psi)$, of $N(\mathbf{m}^0)$ 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 $N(\mathbf{m}^0)$ if and only if

1. $\mathbf{m}^0 \in \Psi$, and
2. (*Path-requirement*) there is a vertex v, and a closed-path $v \xrightarrow{\sigma} v$ in $G(N(\mathbf{m}^i), \mathcal{P}_\Psi)$ ($\sigma \in T^*$), for each $\mathbf{m}^i \in \min(\Psi)$ where
   (a) all transitions appear at least once in σ (i.e. $\mathbf{x}(\sigma) \geq \mathbf{1}$), and
   (b) the net-change in the token-load in each place after the firing of σ is non-negative (i.e. $\mathbf{C}\mathbf{x}(\sigma) \geq \mathbf{0}$).

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 $G(N(\mathbf{m}^0), \mathcal{P}_\Psi)$ (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 Ψ 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

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

which is known to be right-closed (cf. corollary 5.1, [12]; equation 3, [11]). If Ψ is the largest subset that meets the aforementioned requirements, and $\mathbf{m}^0 \in$

Ψ, then the minimal elements, $\min(\Psi)$, effectively represent the minimally restrictive LESP for the PN $N(\mathbf{m}^0)$.

The LESP synthesis procedure is described in figure 3.1.

1: **if** $\mathbf{m}^0 \notin \Psi_i$ **then**
2: The procedure terminates with the conclusion that there is no LESP for $N(\mathbf{m}^0)$.
3: **else if** $\mathbf{m}^0 \in \Psi_i$, and Ψ_i is not control invariant with respect to N **then**
4: Ψ_i is replaced by its largest control invariant subset, Ψ_{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: **else**
6: Each minimal element of the control invariant, right-closed set Ψ_i is tested for the path-requirement on its coverability graph described above.
7: **if** If all minimal elements satisfy this requirement **then**
8: The members of $\min(\Psi_i)$ are presented as a description of the LESP for $N(\mathbf{m}^0)$.
9: **else**
10: Each minimal element $\mathbf{m}^i$ that fails the requirement is "elevated" by $card(\Pi)$-many unit-vectors as follows

$$\mathbf{m}^i \leftarrow \{\mathbf{m}^i + \mathbf{1}_i \mid i \in \{1, 2, \ldots, card(\Pi)\}\}$$

where $\mathbf{1}_i$ is the i-th unit-vector. That is, the above process replaces the minimal element $\mathbf{m}^i$ with $card(\Pi)$-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: **end if**
13: **end if**

Figure 3.1: The procedure for testing the existence of an LESP for a PN $N(\mathbf{m}^0)$, 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 $N(\mathbf{m}^0)$, 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*) complier [14, 15, 16].

3.1.1 LESP Synthesis via an Illustrative Example

The FCPN $N_5(\mathbf{m}_5^0)$ 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.

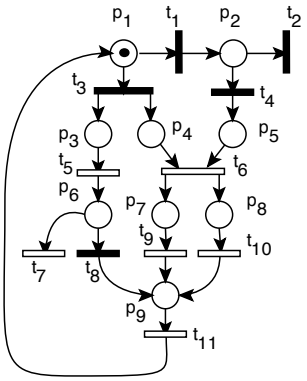


Figure 3.2: An FCPN $N_5(\mathbf{m}_5^0)$ where $N_5 = (\Pi_5, T_5, \Phi_5, \Gamma_5)$.

```
pn5.res        Tue Nov 26 04:30:44 2013        1

 Input File = "pn5"

 Incidence Matrix :

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

 Initial Marking : ( 1 0 0 0 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 0 0 0 )
  2: ( 0 0 1 0 0 0 0 0 0 )
  3: ( 0 0 0 0 0 1 0 0 0 )
  4: ( 0 0 0 0 0 0 0 1 0 )
  5: ( 0 0 0 0 0 0 0 0 1 )
  6: ( 0 0 0 0 0 0 0 0 1 )
  7: ( 0 0 0 0 1 1 0 0 0 )
  8: ( 0 1 0 1 0 0 0 0 0 )


 List of Controllable Transitions
 --------------------------------
 t1 t2 t3 t4 t8

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


  1: ( 1 0 0 0 0 0 0 0 0 )
  2: ( 0 0 0 0 0 0 1 0 0 )
  3: ( 0 0 0 0 0 0 0 0 1 )
  4: ( 0 0 0 0 0 0 0 0 1 )
  5: ( 0 0 0 1 1 0 0 0 0 )
  6: ( 0 1 0 1 0 0 0 0 0 )

 The loop-test failed for the minimal_element: ( 1 0 0 0 0 0 0 0 0 )

 The loop-test failed for the minimal_element: ( 0 0 0 0 0 0 1 0 0 )

 The loop-test failed for the minimal_element: ( 0 0 0 0 0 0 0 0 1 )

 The loop-test failed for the minimal_element: ( 0 0 0 0 0 0 0 0 1 )

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


  1: ( 0 0 0 1 1 0 0 0 0 )
  2: ( 0 1 0 1 0 0 0 0 0 )
  3: ( 2 0 0 0 0 0 0 0 0 )
  4: ( 1 1 0 0 0 0 0 0 0 )
  5: ( 1 0 1 0 0 0 0 0 0 )
  6: ( 1 0 0 1 0 0 0 0 0 )
  7: ( 1 0 0 0 1 0 0 0 0 )
  8: ( 1 0 0 0 0 1 0 0 0 )
  9: ( 1 0 0 0 0 0 1 0 0 )
 10: ( 1 0 0 0 0 0 0 1 0 )
 11: ( 1 0 0 0 0 0 0 0 1 )
 12: ( 1 0 0 0 0 0 0 1 0 )
 13: ( 0 1 0 0 0 0 0 1 0 )
 14: ( 0 0 1 0 0 0 0 1 0 )
 15: ( 0 0 0 1 0 0 0 1 0 )
 16: ( 0 0 0 0 1 0 0 1 0 )
 17: ( 0 0 0 0 0 1 1 0 0 )
 18: ( 0 0 0 0 0 0 2 0 0 )
 19: ( 0 0 0 0 0 0 0 1 1 )
 20: ( 0 0 0 0 0 0 0 1 0 1 )
 21: ( 1 0 0 0 0 0 0 0 1 0 )
 22: ( 0 1 0 0 0 0 0 0 1 0 )
 23: ( 0 0 1 0 0 0 0 0 1 0 )
 24: ( 0 0 0 1 0 0 0 0 1 0 )
 25: ( 0 0 0 0 1 0 0 0 1 0 )
 26: ( 0 0 0 0 0 0 1 0 1 0 )
 27: ( 0 0 0 0 0 0 0 1 1 0 )
 28: ( 0 0 0 0 0 0 0 0 2 0 )
 29: ( 0 0 0 0 0 0 0 0 1 1 )
 30: ( 1 0 0 0 0 0 0 0 0 1 )
 31: ( 0 1 0 0 0 0 0 0 0 1 )
 32: ( 0 0 1 0 0 0 0 0 0 1 )
 33: ( 0 0 0 1 0 0 0 0 0 1 )
 34: ( 0 0 0 0 1 0 0 0 0 1 )
 35: ( 0 0 0 0 0 1 0 0 0 1 )
 36: ( 0 0 0 0 0 0 0 1 0 1 )
 37: ( 0 0 0 0 0 0 0 0 1 1 )
 38: ( 0 0 0 0 0 0 0 0 0 2 )

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


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

 This is An LESP
```

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 Ψ_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 Ψ_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.

$$
\begin{aligned}
1 &: (\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ )^T \\
2 &: (\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ )^T \\
3 &: (\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ )^T \\
4 &: (\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ )^T \\
5 &: (\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ )^T \\
6 &: (\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ )^T,
\end{aligned}
$$

and $\mathbf{m}_3^0 \in \Psi_0$.

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,

$$(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T \xrightarrow{t_1} (0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T,$$

and

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

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 $\mathcal{P}_{\Psi_0}$ 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,

$$\begin{aligned}
(0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0)^T &\xrightarrow{t_9 t_{11}} (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T, \\
(0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0)^T &\xrightarrow{t_{10} t_{11}} (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T, \text{ and} \\
(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)^T &\xrightarrow{t_{11}} (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T.
\end{aligned}$$

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 Ψ_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 $N_5(\mathbf{m}_3^0)$.

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).

CHAPTER 4

RECENT THEORETICAL RESULTS AND OBSERVATIONS ON LESP-SYNTHESIS

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 $N(\mathbf{m}^0)$. 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 $N_1(\mathbf{m}_1^0)$ if and only if it is live, and $N_1(\mathbf{m}_1^0)$ is live if and only if the sum of the tokens assigned to the places by $\mathbf{m}_1^0$ 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. $\forall p \in \Pi$,

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

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 $\{(2\ 0)^T, (0\ 2)^T\}$.

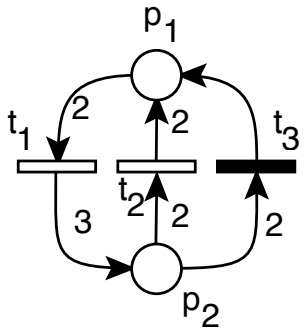


Figure 4.1: General FCPN structure N_6

The general FCPN structure N_7 in Figure 4.2 and N_8 in Figure 4.3 belong to $\mathcal{F}$ class.

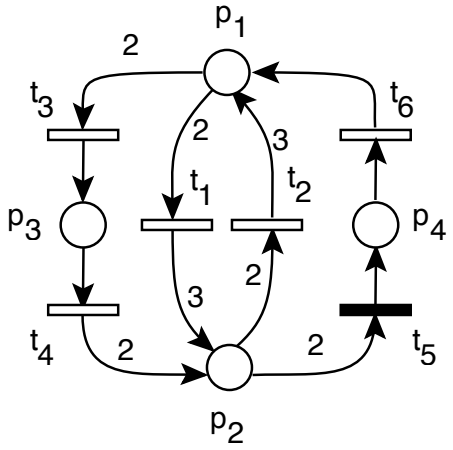


Figure 4.2: General FCPN structure N_7

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 $\{(0\,0\,1\,0)^T, (2\,0\,0\,0)^T, (1\,0\,0\,1)^T, (0\,2\,0\,0)^T, (0\,0\,0\,2)^T\}$. 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 $\{(1\,0\,0\,0\,0)^T, (0\,0\,0\,1\,1)^T\}$.

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 $N(\widehat{\mathbf{m}}^0)$ compared to $N(\mathbf{m}^0)$ where $\widehat{\mathbf{m}}^0 \geq \mathbf{m}^0$, then it is always possible to extend the firing strings in $N(\mathbf{m}^0)$ and $N(\widehat{\mathbf{m}}^0)$ in such a way the Parikh vectors of the extended firing strings are identical, provided there is an LESP for $N(\mathbf{m}^0)$.

Lemma 4.1.1. *Let $\mathcal{P} : \mathcal{N}^n \times T \to \{0, 1\}$ be an LESP for a general FCPN $N(\mathbf{m}^0)$, where $N = (\Pi, T, \Phi, \Gamma)$, $T = T_c \cup T_u$, and $N \in \mathcal{F}$. Suppose $\mathbf{m}^0 \xrightarrow{\sigma} \mathbf{m}^i$ under the supervision of $\mathcal{P}$ in N, and $\widehat{\mathbf{m}}^0 \xrightarrow{\widehat{\sigma}} \widehat{\mathbf{m}}^j$ in the absence of any supervision in N for some $\widehat{\mathbf{m}}^0 \geq \mathbf{m}^0$. Further, let us suppose that the number of*

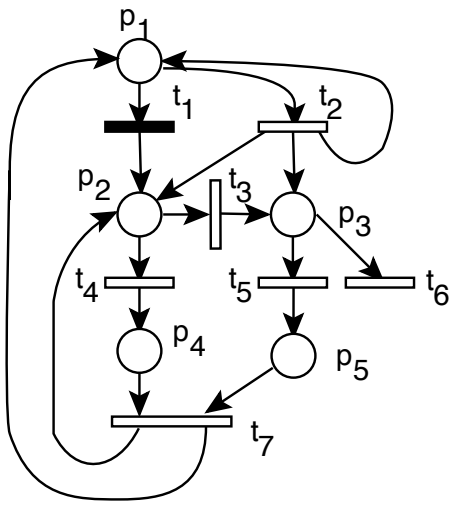


Figure 4.3: General FCPN structure N_8

occurrences of each controllable transition in $\widehat{\sigma}$ *and* σ *are identical; however, the string* $\widehat{\sigma}$ *has a few more uncontrollable transitions than the string* σ*. That is,* $\{t_j \in T \mid \mathbf{x}(\widehat{\sigma})_j > \mathbf{x}(\sigma)_j\} \subseteq T_u$*. Then* $\exists \widetilde{\sigma}_1, \widetilde{\sigma}_2 \in T^*$*, such that*

1. $\widehat{\mathbf{m}}^j \stackrel{\widetilde{\sigma}_1}{\rightarrow} \widehat{\mathbf{m}}^k$ *in* N *in the absence of any supervision,*

2. $\mathbf{m}^i \stackrel{\widetilde{\sigma}_2}{\rightarrow} \mathbf{m}^l$ *under the supervision of* $\mathcal{P}$ *in* N*, and*

3. $\mathbf{x}(\widehat{\sigma}\widetilde{\sigma}_1) = \mathbf{x}(\sigma\widetilde{\sigma}_2)$ $(\Rightarrow \widehat{\mathbf{m}}^k \geq \mathbf{m}^l)$*.*

That is, $\mathbf{m}^0 \stackrel{\sigma\widetilde{\sigma}_2}{\rightarrow} \mathbf{m}^l$ *under the supervision of* $\mathcal{P}$*, and* $\widehat{\mathbf{m}}^0 \stackrel{\widehat{\sigma}\widetilde{\sigma}_1}{\rightarrow} \widehat{\mathbf{m}}^k$ *in the absence of any supervision in* N*. If* $\widehat{\mathbf{m}}^0 \geq \mathbf{m}^0$ *and* $\mathbf{x}(\sigma\widetilde{\sigma}_2) = \mathbf{x}(\widehat{\sigma}\widetilde{\sigma}_1)$*, then* $\widehat{\mathbf{m}}^k \geq \mathbf{m}^l$*.*

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

1. $\mathbf{m}^i \stackrel{\sigma_1}{\rightarrow} \mathbf{m}^{i+1}$ under $\mathcal{P}$ in N,

2. $\forall \overline{\sigma}_1 \in pr(\sigma_1) - \{\sigma_1\}$, if $\mathbf{m}^i \stackrel{\overline{\sigma}_1}{\rightarrow} \overline{\mathbf{m}}$, then $T_e(N, \overline{\mathbf{m}}) \cap \{t_j \in T \mid \mathbf{x}(\widehat{\sigma})_j > \mathbf{x}(\sigma)_j\} = \emptyset$, where $pr(\bullet)$ is the prefix-set of the string argument, and

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

That is, none of the transitions in the set $\{t_j \in T \mid \mathbf{x}(\widehat{\sigma})_j > \mathbf{x}(\sigma)_j\} \subseteq T_u$ are state-enabled (and trivially control-enabled) following the firing of any proper prefix of the firing string σ_1. Additionally, at least one member of the

set $\{t_j \in T \mid \mathbf{x}(\widehat{\sigma})_j > \mathbf{x}(\sigma)_j\}$ is state-enabled (and trivially control-enabled) at the marking $\mathbf{m}^{i+1}$ that results from the firing of the string σ_1 at $\mathbf{m}^i$.

It follows that $\widehat{\mathbf{m}}^j \stackrel{\sigma_1}{\rightarrow} \widehat{\mathbf{m}}^{j+1}$ in the absence of any supervision in N, which can be established by contradiction. Suppose $\sigma_1 = \overline{t}_1 \cdots \overline{t}_i \overline{t}_{i+1} \cdots$, and $\widehat{\mathbf{m}}^j \stackrel{\overline{t}_1 \cdots \overline{t}_i}{\longrightarrow} \widehat{\mathbf{m}}^{j+2}$, but $\overline{t}_{i+1} \notin T_e(N, \widehat{\mathbf{m}}^{j+2})$. This must be due to the reduction in the number of tokens in an input place of $\overline{t}_{i+1}$, as a result of the firing of some transition $t_u \in \{t_j \in T \mid \mathbf{x}(\widehat{\sigma})_j > \mathbf{x}(\sigma)_j\} (\subseteq T_u)$. Since N is an FCPN structure, transitions t_u and $\overline{t}_{i+1}$ must share a unique input place (i.e. ${}^\bullet t_u \cap {}^\bullet \overline{t}_{i+1} = \{p\}$ for some $p \in \Pi$). Since $N \in \mathcal{F}$, whenever p has sufficient tokens to state-enable $\overline{t}_{i+1}$, it follows that t_u is also state-enabled at the same marking. This contradicts the second of three conditions required of σ_1.

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

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

1. $\forall t \in T, \widehat{\mathcal{P}}(\widehat{\mathbf{m}}^0, t) = \mathcal{P}(\mathbf{m}^0, t)$.
2. Suppose $\widehat{\mathbf{m}}^0 \stackrel{\widehat{\sigma}}{\rightarrow} \widehat{\mathbf{m}}^j$ in N under the supervision of $\widehat{\mathcal{P}}$,
   (a) $\forall t_i \in T_u, \widehat{\mathcal{P}}(\widehat{\mathbf{m}}^j, t_i) = 1$.
   (b) $\forall t_i \in T_c, (\widehat{\mathcal{P}}(\widehat{\mathbf{m}}^j, t_i) = 1) \Leftrightarrow$
      i. $t_i \in T_e(N, \widehat{\mathbf{m}}^j)$, and
      ii. $\exists \sigma \in T^*$, such that
         A. $\mathbf{m}^0 \stackrel{\sigma}{\rightarrow} \mathbf{m}^k$ under the supervision of $\mathcal{P}$ in N,
         B. $\forall k \in \{1, 2, \ldots, m\}, \mathbf{x}(\widehat{\sigma} t_i)_k \geq \mathbf{x}(\sigma)_k$, and
         C. $\{t_j \in T \mid \mathbf{x}(\widehat{\sigma} t_i)_j > \mathbf{x}(\sigma)_j\} \subseteq T_u$.

That is, the policy $\widehat{\mathcal{P}}$ 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 $\widehat{\mathcal{P}}$ is an LESP for

$N(\widehat{\mathbf{m}}^0)$ (cf. proof of lemma 5.4, [11]), which in turn leads to the following result, which parallels theorem 5.6 of reference [11].

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 $N(\mathbf{m}^0)$ for any $\mathbf{m}^0 \in \Delta(N)$. As an illustration, the minimally restrictive LESP for $N_6(\mathbf{m}_6^0)$ for any $\mathbf{m}_6^0 \in \Delta(N_6)$, 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 $N_7(\mathbf{m}_7^0)$, where N_7 is the PN structure of figure 4.2, and $\mathbf{m}_7^0 \in \Delta(N_7)$. Likewise, the minimally restrictive LESP for $N_8(\mathbf{m}_8^0)$, where N_8 is the PN structure of figure 4.3, and $\mathbf{m}_8^0 \in \Delta(N_8)$), 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.

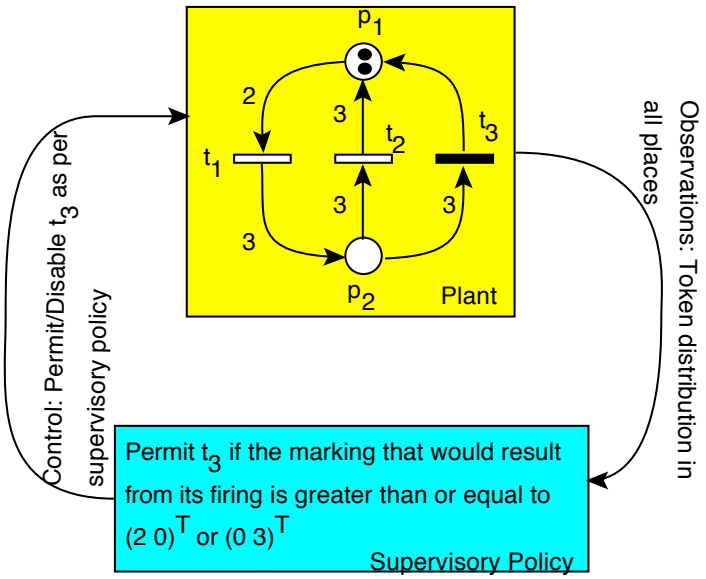


Figure 4.4: The minimally-restrictive LESP that ensures all reachable markings of the plant FCPN PN $N_6(\mathbf{m}_6^0)$ are in the right-closed set $\Delta(N_6)$ identified by the minimal elements $\{(2\ 0)^T, (0\ 3)^T\}$. This policy is a minimally restrictive LESP for any $\mathbf{m}_6^0 \in \Delta(N_6)$.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

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 $N \in \mathcal{H}$.

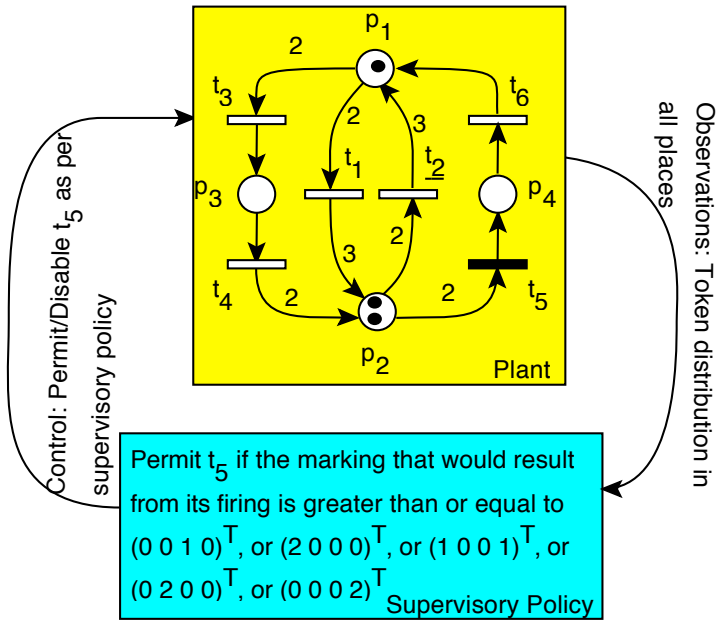


Figure 4.5: The minimally restrictive LESP for the general plant PN $N_7(\mathbf{m}_7^0)$. This LESP ensures the markings reachable under its supervision from any $\mathbf{m}_7^0 \in \Delta(N_7)$ remain within the right-closed set $\Delta(N_7)$, where $\min(\Delta(N_7))$
$= \{(0\ 0\ 1\ 0)^T, (2\ 0\ 0\ 0)^T, (1\ 0\ 0\ 1)^T, (0\ 2\ 0\ 0)^T, (0\ 0\ 0\ 2)^T\}$.

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 $\mathcal{H} \subseteq \widetilde{\mathcal{H}}$, where $\widetilde{\mathcal{H}}$ is defined in the following paragraphs.

Let, $\Omega(t) = \{\widehat{t} \in T \mid^\bullet t \cap^\bullet \widehat{t} \neq \emptyset\}$, 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 $\widetilde{\mathcal{H}}$ 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, $\widetilde{\mathcal{H}}$ 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 $N \in \widetilde{H}$, we get the result shown below.

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

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

There is an LESP for the PN $N(\mathbf{m}^0)$ if and only if $\mathbf{m}^0 \in \Delta(N)$, 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

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

This requirement defines a class of PNs, which we denote as $\mathcal{H}(\subseteq \widetilde{\mathcal{H}})$, 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 $\forall p \in \Pi, \forall t_u \in p^\bullet \cap T_u$,*

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

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

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

1. $\Gamma(p, t_u) > \min_{t \in p^\bullet} \Gamma(p, t)$, or
2. $\exists t \in \Omega(t_u),\ {}^\bullet t_u - {}^\bullet t \neq \emptyset$.

In each of these cases we construct a marking $\mathbf{m} \in \mathcal{N}^n$ such that $\exists t \in \Omega(t_u), t \in T_e(N, \mathbf{m})$ and $t_u \notin T_e(N, \mathbf{m})$, which leads to the conclusion that $N \notin \mathcal{H}$.

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

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

There is an $O(n^2 m^2)$ algorithm that decides if an arbitrary PN structure belongs to the class $\mathcal{H}$, where $n = card(\Pi)$ and $m = card(T)$. 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 $N(\mathbf{m}^0)$ is decidable. Furthermore, the software package described in references [14, 15, 16, 5] can be used to compute the minimally restrictive LESP for $N(\mathbf{m}^0)$, 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 $({}^\bullet t)^\bullet \neq \{t\}$ $(({}^\bullet t)^\bullet = \{t\})$. The minimally restrictive LESP for an ordinary FCPN $N(\mathbf{m}^0)$ 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 $N(\mathbf{m}^0)$ where $N \in \mathcal{H}$.

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

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).

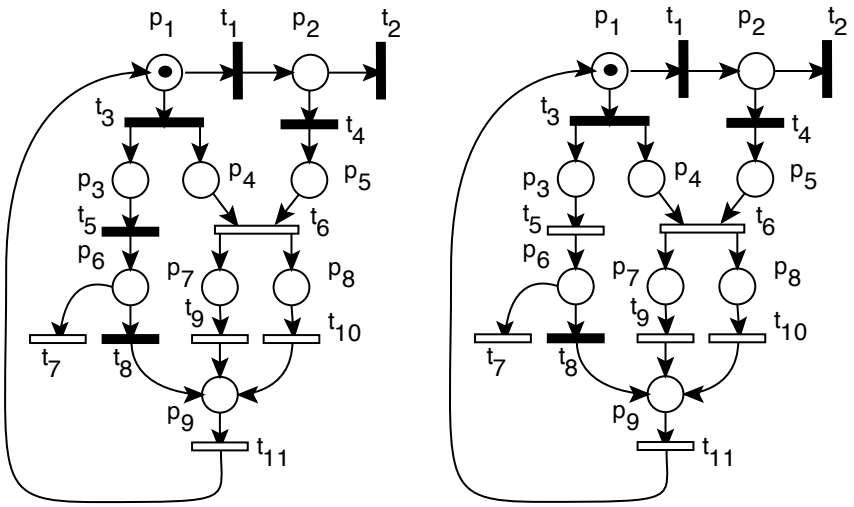
The sets $\Delta(N_9)$ and $\Delta(N_5)$ are identical, and are identified by the twenty-four minimal elements shown in figure 3.3, which shows the output generated by the above mentioned software for N_7.

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 (Ψ_0) is identified by the six minimal elements of figure 3.3 along with the vector $(0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0)^T$. 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 Ψ_1 has the twenty-four minimal elements shown in figure 3.3 together with eight new elements

$$\{(1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0)^T, (0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0)^T, (0\ 0\ 2\ 0\ 0\ 0\ 0\ 0\ 0)^T,$$
$$(0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0)^T, (0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0)^T, (0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 0)^T,$$
$$(0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0)^T, (0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1)^T\}.$$

That is, the minimal element $(0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0)^T$ of Ψ_0 is replaced by these eight elements, which defines Ψ_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-



(a) $N_6 \in \mathcal{H}$ (b) $N_7 \in \mathcal{H}$

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 LESPs for a PN from that of another PN that is "similar" to it.

4.4 LESPs 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 LESPs 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 $\widehat{N} = (\widehat{\Pi}, \widehat{T}, \widehat{\Phi}, \widehat{\Gamma})$ be two PN structures. Suppose

$\alpha : T \to \widehat{T}$ is an injection (i.e. every member of T has an image in $\widehat{T}$; 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 $\alpha : T^* \to \widehat{T}^*$ for $\sigma \in T^*$ and $t \in T$, as

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

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

$$\begin{aligned} \alpha^{-1}(\widehat{\sigma}\widehat{t}) &= \alpha^{-1}(\widehat{\sigma}), \qquad \text{when } \widehat{t} \notin \alpha(T), \\ \alpha^{-1}(\widehat{\sigma}\widehat{t}) &= \alpha^{-1}(\widehat{\sigma})\alpha^{-1}(\widehat{t}), \qquad \text{when } \widehat{t} \in \alpha(T), \end{aligned}$$

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

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

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

1. $\mathbf{m}^0 = \beta(\widehat{\mathbf{m}}^0)$,
2. *Suppose $\mathbf{m}^1 = \beta(\widehat{\mathbf{m}}^1), \widehat{\mathbf{m}}^1 \in \Re(\widehat{N}, \widehat{\mathbf{m}}^0, \widehat{\mathcal{P}})$ and $\mathbf{m}^1 \in \Re(N, \mathbf{m}^0, \mathcal{P})$, then*
   (a) *whenever $\mathbf{m}^1 \xrightarrow{t} \mathbf{m}^2$ in N under $\mathcal{P}$, then $\exists \widehat{\mathbf{m}}^2 \in \beta^{-1}(\mathbf{m}^2), \exists \widehat{\sigma} \in \widehat{T}^*$ such that $\widehat{\mathbf{m}}^1 \xrightarrow{\widehat{\sigma}} \widehat{\mathbf{m}}^2$ under $\widehat{\mathcal{P}}$ in $\widehat{N}$, and $\alpha^{-1}(\widehat{\sigma}) = t$.*
   (b) *whenever $\widehat{\mathbf{m}}^1 \xrightarrow{\widehat{\sigma}} \widehat{\mathbf{m}}^2$ under $\widehat{\mathcal{P}}$ in $\widehat{N}$ for some $\widehat{\sigma} \in \widehat{T}^*$, then $\mathbf{m}^1 \xrightarrow{\alpha^{-1}(\widehat{\sigma})} \beta(\widehat{\mathbf{m}}^2)$ under $\mathcal{P}$ in N.*

As noted in reference [26], the fact that $\alpha : T \to \widehat{T}$ is an injective function would mean that $card(\widehat{T}) \geq card(T)$. The transitions in $\alpha(T)$ simulate the

transitions in T, while the remaining transitions (i.e. the set $\widehat{T} - \alpha(T)$) are to be viewed as an internal to $\widehat{N}$.

A marking $\widehat{\mathbf{m}} \in \mathcal{N}^{card(\widehat{\Pi})}$ of $\widehat{N}$ represents the marking $\beta(\widehat{\mathbf{m}})$ in N. There could be many markings of $\widehat{N}$ 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 $\widehat{N}$.

Item 1 of definition 4.4.1 requires that the initial marking of $\widehat{N}$ 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 $\widehat{N}$ under the supervision of $\widehat{\mathcal{P}}$, while item 2b requires that every firing string in $\widehat{N}$ under the supervision of $\widehat{\mathcal{P}}$ 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 $\widehat{N} = (\widehat{\Pi}, \widehat{T}, \widehat{\Phi}, \widehat{\Gamma})$ are two PN structures, and $\alpha : T \to \widehat{T}$ is an injective function. Let $N(\mathbf{m}^0)$ (resp. $\widehat{N}(\widehat{\mathbf{m}}^0)$) be supervised by policy $\mathcal{P} : \mathcal{N}^{card(\Pi)} \times T \to \{0,1\}$ (resp. $\widehat{\mathcal{P}} : \mathcal{N}^{card(\widehat{\Pi})} \times \widehat{T} \to \{0,1\}$), and $\widehat{N}(\widehat{\mathbf{m}}^0)$ under the supervision of $\widehat{\mathcal{P}}$ simulates $N(\mathbf{m}^0)$ under $\mathcal{P}$ with respect to $\alpha(\bullet)$, then $\mathcal{P}$ is an LESP for $N(\mathbf{m}^0)$ if and only if $\forall \widehat{t} \in \alpha(T)$, $\widehat{t}$ is live under $\widehat{\mathcal{P}}$.*

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

Since $\mathcal{P}$ is an LESP for N, $\forall t \in T, \exists \sigma_2 \in T^*, \exists \mathbf{m}^2 \in \mathcal{N}^{card(\Pi)}$ such that $\beta(\widehat{\mathbf{m}}^1) \xrightarrow{\sigma_2} \mathbf{m}^2$ in N under $\mathcal{P}$, where t occurs once in σ_2. From item 2a of definition 4.4.1, $\exists \widehat{\sigma}_2 \in \widehat{T}^*, \exists \widehat{\mathbf{m}}^2 \in \Re(\widehat{N}, \widehat{\mathbf{m}}^0, \widehat{\mathcal{P}})$ such that $\widehat{\mathbf{m}}^1 \xrightarrow{\widehat{\sigma}_2} \widehat{\mathbf{m}}^2$ and $\alpha(t)$ occurs in $\widehat{\sigma}_2$. Therefore, all $\widehat{t} \in \alpha(T)$ is live under $\widehat{\mathcal{P}}$ in $\widehat{N}(\widehat{\mathbf{m}}^0)$.

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

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

$t \in T$	$\alpha(t) \in \widehat{T}$
t_0	$\widehat{t}_1$
t_1	$\widehat{t}_8$
t_2	$\widehat{t}_9$
t_3	$\widehat{t}_{10}$
t_4	$\widehat{t}_{11}$
t_5	$\widehat{t}_{12}$
t_6	$\widehat{t}_{13}$

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

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

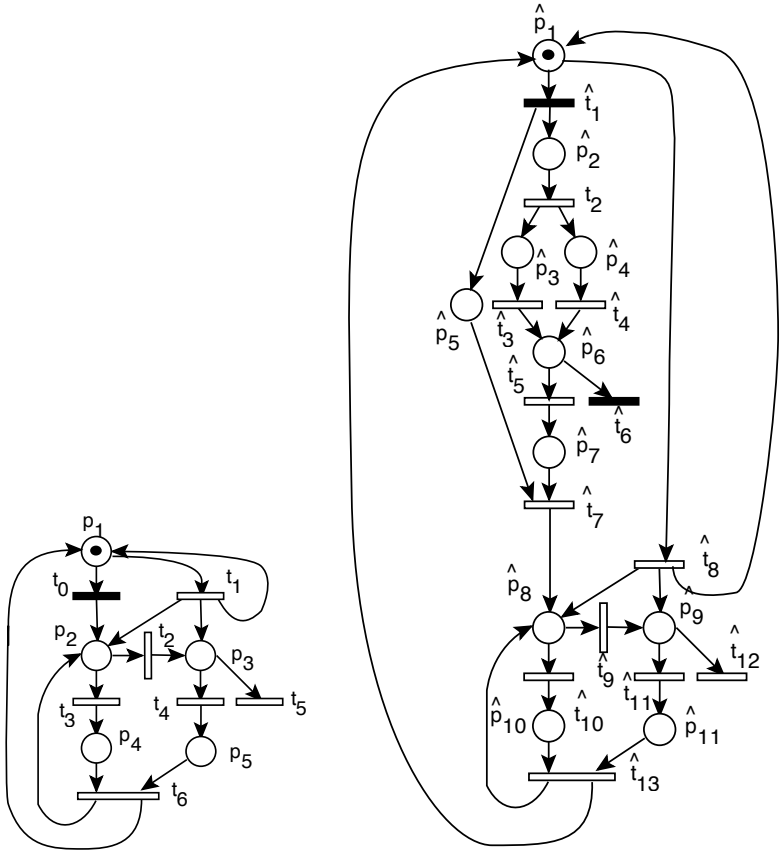
Consider the PN structures $N = (\Pi, T, \Phi, \Gamma)$ and $\widehat{N} = (\widehat{\Pi}, \widehat{T}, \widehat{\Phi}, \widehat{\Gamma})$ shown in figure 1.1(a) and (b) respectively. The injective function $\alpha : T \to \widehat{T}$ is defined in table 4.1.

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

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

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

From method of references [27, 28], we know that the supervisory policy $\mathcal{P}$



(a) $N(\mathbf{m}^0)$ (b) $\widehat{N}(\widehat{\mathbf{m}}^0)$

Figure 4.7: If $\widehat{\mathcal{P}}$ is a policy that permits the firing of $\widehat{t_6}$ if and only if $((\widehat{\mathbf{m}}_1 + \widehat{\mathbf{m}}_2 + \widehat{\mathbf{m}}_3 + \widehat{\mathbf{m}}_4 + \widehat{\mathbf{m}}_6 + \widehat{\mathbf{m}}_7) \geq 2)$; and, permits the firing of $\widehat{t_1}(= \alpha(t_0))$ 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 $\widehat{N}(\widehat{\mathbf{m}}^0)$ under $\widehat{\mathcal{P}}$ *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 $\alpha(T) = \{\widehat{t_1}, \widehat{t_8}, \widehat{t_9}, \widehat{t_{10}}, \widehat{t_{11}}, \widehat{t_{12}}, \widehat{t_{13}}\}$ is live under $\widehat{\mathcal{P}}$ in $\widehat{N}$.

that permits the firing of t_0 if and only if $((\mathbf{m}_1 \geq 2) \vee (\mathbf{m}_4 \geq 1) \wedge (\mathbf{m}_5 \geq 2))$ is an LESP. As a consequence of the above observation, we can conclude that the supervisory policy $\widehat{\mathcal{P}}$ that permits the firing of $\widehat{t}_1$ if and only if $((\widehat{\mathbf{m}}_1 \geq 2) \vee (\widehat{\mathbf{m}}_{10} \geq 1) \wedge (\widehat{\mathbf{m}}_{11} \geq 2))$; and permits the firing of $\widehat{t_6}$ if and only if $((\widehat{\mathbf{m}}_1 + \widehat{\mathbf{m}}_2 + \widehat{\mathbf{m}}_3 + \widehat{\mathbf{m}}_4 + \widehat{\mathbf{m}}_6 + \widehat{\mathbf{m}}_7) \geq 2)$ is an LESP for $\widehat{N}(\mathbf{m}^0)$. That is, the LESP for the larger PN $\widehat{N}(\mathbf{m}^0)$ was synthesized from the LESP for the smaller PN $N(\mathbf{m}^0)$ 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 $\min(\Delta(N))$.

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 $N_7(\mathbf{m}_7^0)$. The input file for $N_7(\mathbf{m_7^0})$ is shown in Figure 5.3. Figure 5.4 shows the output generated by the software, which lists five minimal elements of $\min(\Delta(N_7))$. 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 $N_7(\mathbf{m_7^0})$ for any $\mathbf{m_7^0} \in \Delta(N_7)$.

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

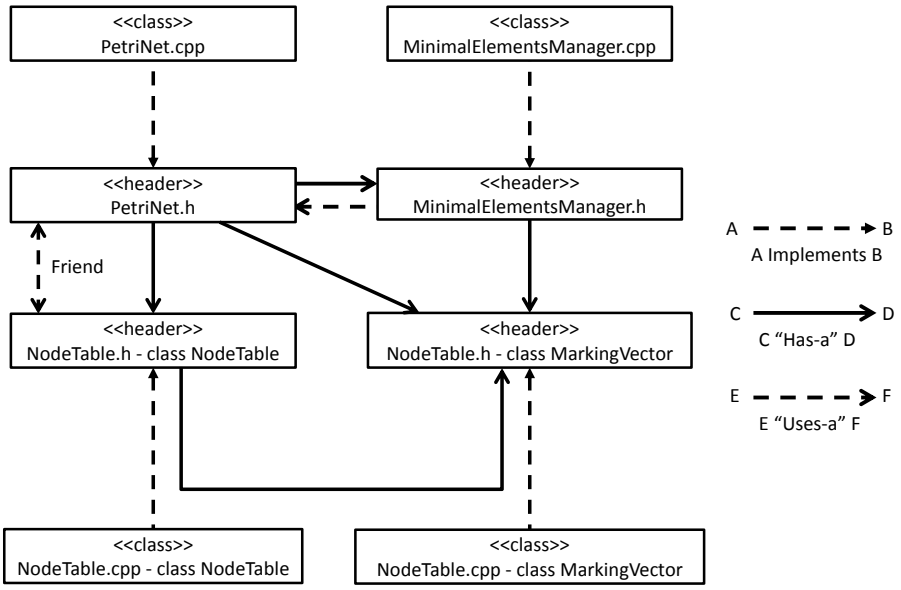


Figure 5.1: Class Diagram.

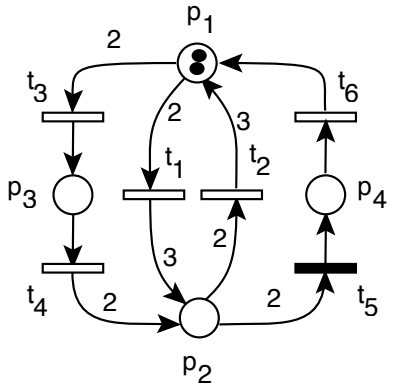


Figure 5.2: $N_7(\mathbf{m_7^0})$

complexity of the system, i.e. with 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. The

```
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
```

Figure 5.3: Input file for $N_7(\mathbf{m_7^0})$. The first line shows that there are four places and six transitions in the PN structure. This is followed by the (4×6) **IN** and **OUT** matrices, which accounts for the eight lines that follow the first line. The penultimate line identifies the initial marking $\mathbf{m}_7^0$ 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 $\mathbf{m^0}$. 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( ),

```
Input File = "pn1"

Incidence Matrix :

   T  1  2  3  4  5  6
P
  1  -2  3 -2  .  .  1
  2   3 -2  .  2 -2  .
  3   .  .  1 -1  .  .
  4   .  .  .  .  1 -1

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
--------------------------------
t5


(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. 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,

$$p_a^{\bullet} \cap T_u - \{t_m\} = \emptyset.$$

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

A path $p_a \xrightarrow{w_1} t_m \xrightarrow{w_2} p_b \xrightarrow{w_3} t_n$ in the original PN is simplified as $\widetilde{p}_a \xrightarrow{\frac{w_1 w_3}{w_2}} t_n$. 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 $\frac{w_1 w_3}{w_2}$ 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 $\frac{w_3 w_1}{w_2}$ number of tokens being removed from $\widetilde{p}_a$. 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. $\frac{w_3}{w_2}$ is an integer. This can be deduced from the argument above where we proved that for m many firing of t_m , $\frac{w_3}{w_2} = m$.

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 $\widetilde{p}_a$ 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

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

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

$$b = p + \frac{w_1}{w_2} q \tag{6.1}$$

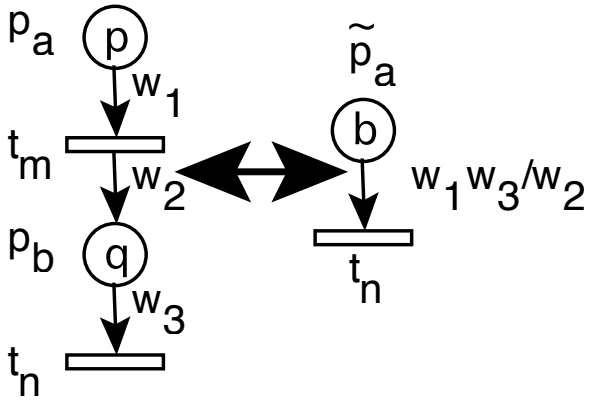


Figure 6.1: Rule #1

6.1.2 Rule 2

Rule 2 is pictorially depicted in Figure 6.2. A path $t_a \stackrel{w_1}{\rightarrow} p_m \stackrel{w_2}{\rightarrow} t_b \stackrel{w_3}{\rightarrow} p_n$ in the original PN is simplified as $t_a \stackrel{\frac{w_1 w_3}{w_2}}{\rightarrow} \widetilde{p}_n$. 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 $(\frac{w_1 w_3}{w_2})$-many tokens in $\widetilde{p}_n$. 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 $\widetilde{p}_n$ 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,

$$a = y + \frac{w_3}{w_2}x \tag{6.2}$$

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:

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

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 $\widetilde{p}_n$, then we

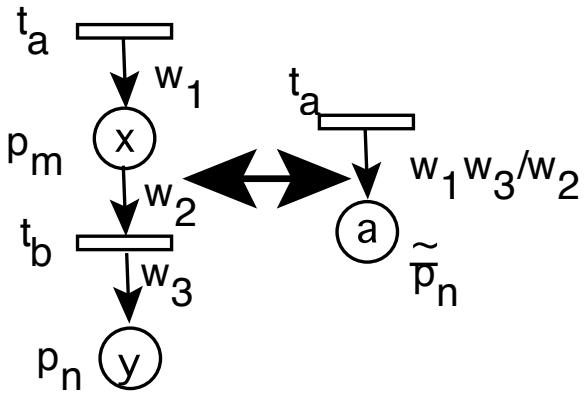


Figure 6.2: Rule #2

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,

$$a = z + \frac{w_2}{w_1}x + \frac{w_4}{w_3}y \tag{6.4}$$

The condition under which this rule can be used is:

1. $\frac{w_1}{w_3}$ has to be an integer, and
2. $\frac{w_2}{w_4}$ has to be an integer.

The reduction rule says each firing of t_a in the reduced PN will place $(\frac{w_1 w_5}{w_3} + \frac{w_2 w_6}{w_4})$-many (i.e. integer-many) tokens in $\widetilde{p}_n$. 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.

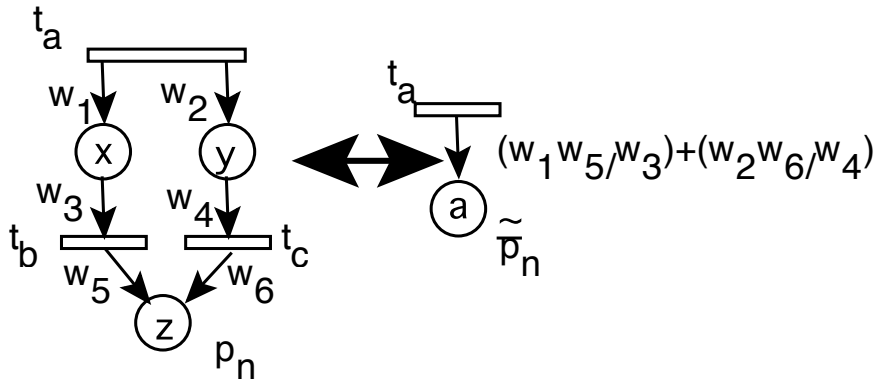


Figure 6.3: Rule #3

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

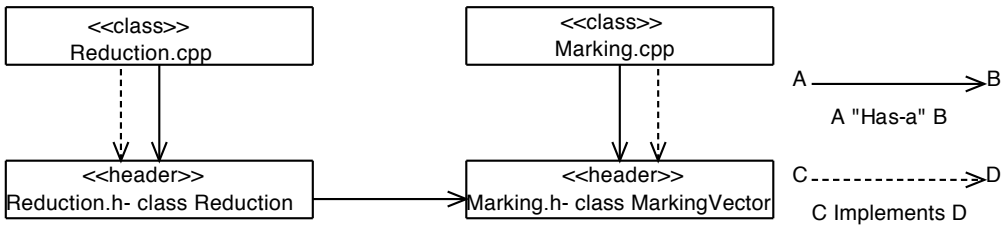


Figure 6.4: Class Diagram for Reduction Techniques

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 $\mathbf{m}^i$ 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 $<$, $\leq$, $==$ and $\geq$ on *MarkingVector* objects. The Figure 6.5 shows the structure of this class.

Figure 6.5: Class structure of *MarkingVector*

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* (**IN**), the *output matrix* (**OUT**) and the initial marking $\mathbf{m}^0$. 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.

noOfPlaces, noOfTransitions, noOfPlaces_global and noOfTransitions_global are integer members. initialMarking is of type MarkingVector. The members inputWeightsToTransition, outputWeightsFromTransition, inputWeightsToTran-

sition_global and OutputWeightsFromTransition_global are defined as std::vector of pointers to objects of type MarkingVector.

To compute the reduced net the algorithm first identifies the places and the transtions 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 placestobereduced and transitionstobereduced 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*

Name	Type	Description
incidenceMatrix	*vector < MarkingVector* >*	Represents the Incidence Matrix **C** that is used to characterize any PetriNet.
initialMarking	*MarkingVector*	Stores the initial marking $\mathbf{m}^0$ of a PetriNet
inputWeightsToTransition	*vector<MarkingVector* >*	Represents the input matrix of the current PetriNet at any given step.

Table 6.1 (cont.)		
inputWeightsToTransition_global	*vector<MarkingVector* >*	Represents the input matrix of the original PetriNet.
noOfPlaces	*int*	Represents the number of places of the current PetriNet at any given step.
noOfPlaces_global	*int*	Represents the number of places for the original PetriNet provided by the user.
noOfTransitions	*int*	Represents the number of transitions of the current PetriNet at any given step.
noOfTransitions_global	*int*	Represents the number of transitions for the original PetriNet provided by the user.
outputWeightsToTransition	*vector<MarkingVector* >*	Represents the output matrix of the current PetriNet at any given step.

Table 6.1 (cont.)		
outputWeightsToTransition_global	*vector<MarkingVector* >*	Represents the output of the original PetriNet.
place_i	*int*	Represents one of the variables passed to the function that computes the path for reduction .
place_reduced	*vector <int>*	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 .
place_reduced_global	*vector <int>*	Assigns the value 1 corresponding to a place that satisfies the condition to be removed for the original PetriNet

Table 6.1 (cont.)		
placestobereduced	*int*	Increments everytime a value 1 is assigned to *places_reduced*.
transition_j	*int*	Represents one of the variables passed to the function that computes the path for reduction .
transitions_reduced	*vector <int>*	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 .
transitions_reduced_global	*vector <int>*	Assigns the value 1 corresponding to a transition that satisfies the condition to be removed for the original PetriNet

Table 6.1 (cont.)		
transitionstobereduced	*int*	Increments everytime a value 1 is assigned to *transitions_reduced*.

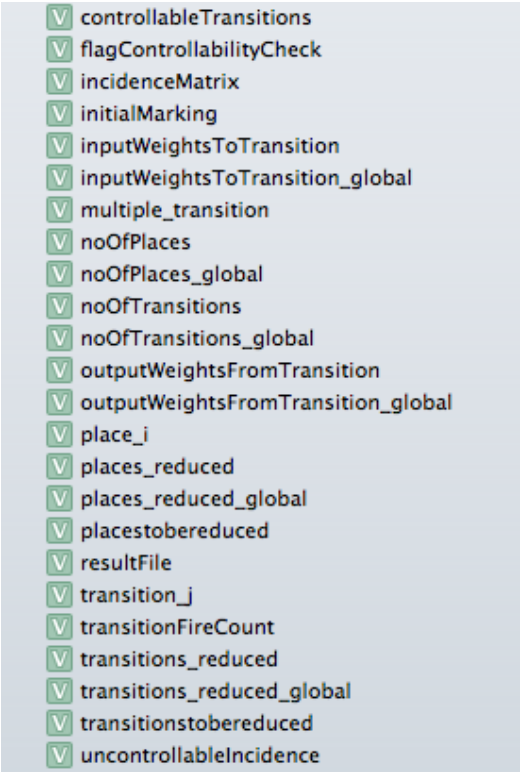


Figure 6.6: Class structure - variables of class *Reduction*

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**, $\mathbf{m}^0$, 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 reduced input matrix **IN** and output matrix **OUT** are computed in primarily three modules.

1. Identifying the places and the transitions that can be removed.
2. Identifying the path $p_a \rightarrow t_m \rightarrow p_b \rightarrow t_n$ or $t_a \rightarrow p_m \rightarrow t_b \rightarrow p_n$ 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 Reduc(), reduction_path() and reduction_matrix() or reduction_matrix_t() respectively.

The member function 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 std::vector places_reduced. In a similar manner the value 1 is assigned to the corresponding transition in the std::vector transitions_reduced provided there is exactly one arc to and from the transition. Following these computations the member functions pathstobereduced() and reduction_path() are invoked. The std::vector, places_reduced and transitions_reduced are cleared at the start of the 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 $p_a \rightarrow t_m \rightarrow p_b \rightarrow t_n$ or $t_a \rightarrow p_m \rightarrow t_b \rightarrow p_n$ that can be eliminated the function pathstobereduced() is defined. For every place p_b identified in function Reduc() that can be eliminated we look for a path $p_a \rightarrow t_m \rightarrow p_b \rightarrow t_n$ such that the std::vector places_reduced has the value 1 corresponding to p_b and the 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 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.

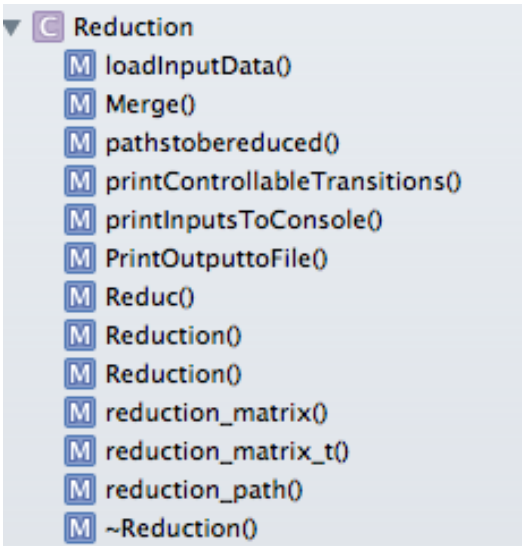


Figure 6.7: Class structure - methods of class *Reduction*

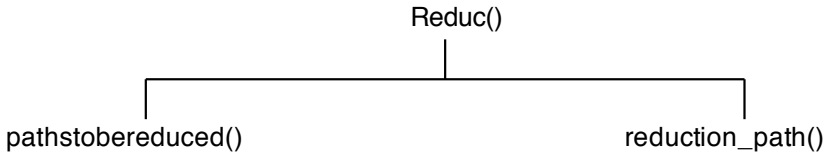


Figure 6.8: Flowchart of *Reduc*() method

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.

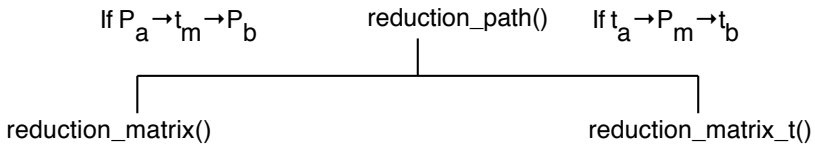


Figure 6.9: Flowchart of *reduction_path*() method

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 $\widetilde{p}_a \xrightarrow{\frac{w_1 w_3}{w_2}} t_n$
- 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 $t_a \xrightarrow{\frac{w_1 w_3}{w_2}} \widetilde{p}_n$

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 $w_1 * w_3$ 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.

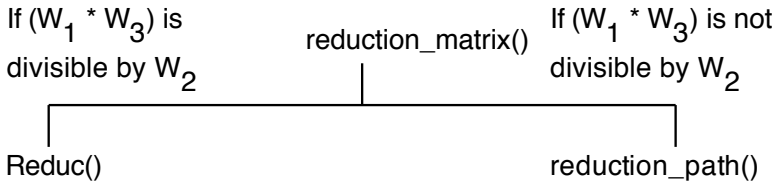


Figure 6.10: Flowchart of *reduction_matrix*() method

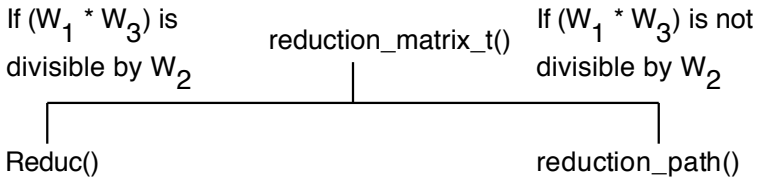


Figure 6.11: Flowchart of *reduction_matrix_t*() method

The Figure 6.12 represents the flowchart for member functions of class *Reduction*.

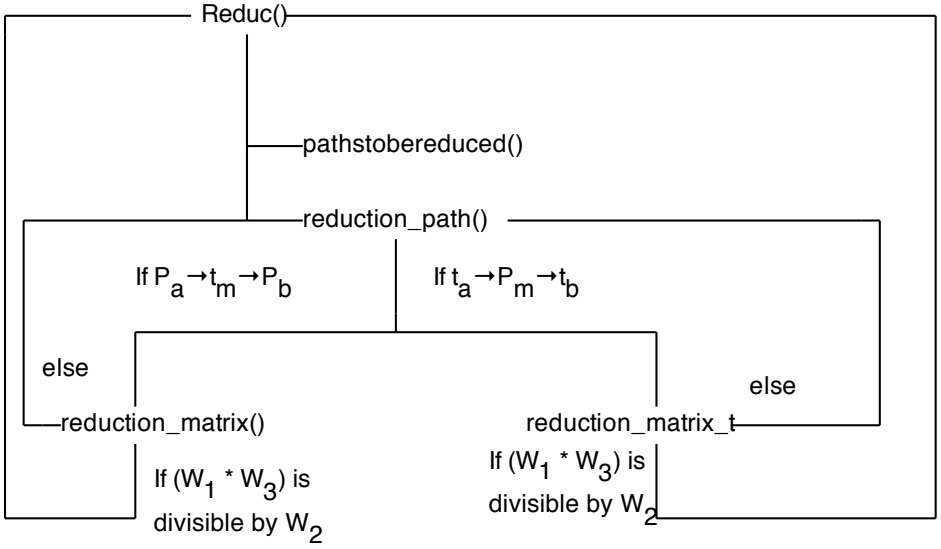


Figure 6.12: Flowchart of member functions of class *Reduction*

Table 6.2: Method definitions of class *Reduction*

Name	Description	Parameter	Returntype
loadInputData()	Assigns the PN's inputs to the corresponding members.	*char* * : input file name	*void*
printControllableTransitions()	Print method to write the set of controllable transitions to the console.	*void*	*void*
printInputsToConsole()	Print method to write the inputs to the console.	*void*	*void*
Reduc()	Compute the places and transitions that can be eliminated	*int noOfPlaces, int noOfTransitions*	*void*

Table 6.2 (cont.)			
pathstobereduced()	Prints the paths that can be reduced and the transitions that can be merged for the original net.	*int* $place_0$, *int* $transition_0$	*void*
reduction_path()	Invokes the appropriate functions *reduction_matrix* or *reduction_matrix_t* and determines if the PN cannot be reduced further	*int place, int transition*	*void*
reduction_matrix()	Computes the new input and output matrix	*int place, int transition*	*void*
reduction_matrix_t()	Computes the new input and output matrix	*int place, int transition*	*void*
PrintOutputtoFile()	Prints the final values on an output file.	*char* * : output file name	*void*

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.

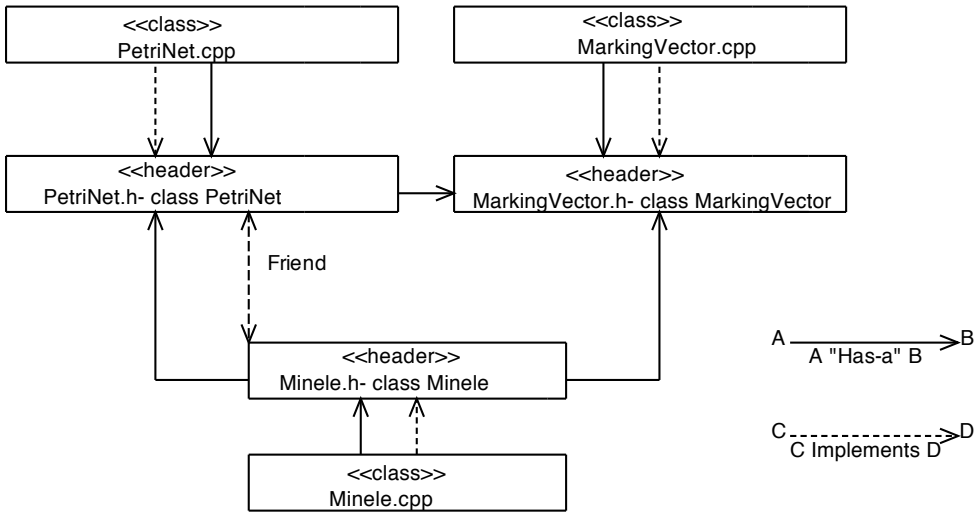


Figure 6.13: Class Diagram for Deducing Minimal Elements

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 $\mathbf{m}^0$. 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

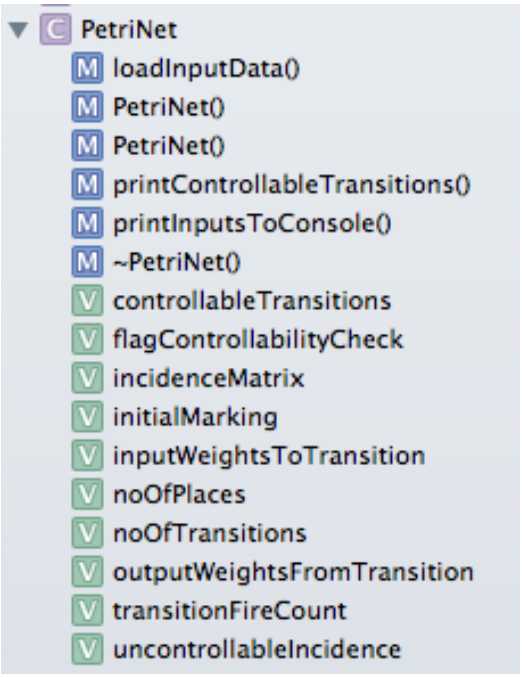


Figure 6.14: Class structure of *PetriNet*

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 loadInputData_1() 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.

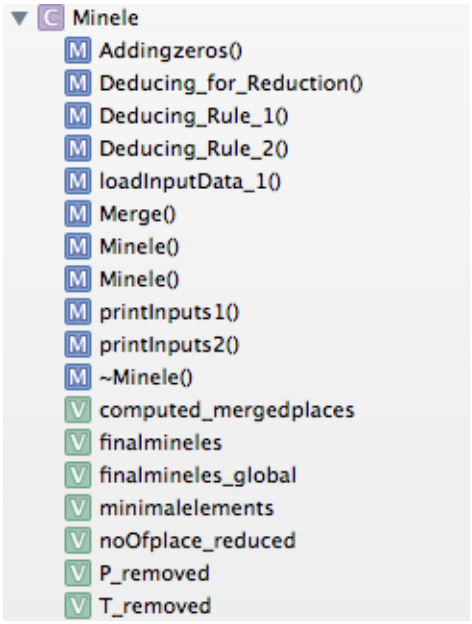


Figure 6.15: Class structure of *Minele*

Table 6.3: Method definitions of class *Minele*

Name	Description	Parameter	Returntype
loadInputData_1()	Initializes the set of minimal elements for the reduced PN.	*char* * : input file name	*void*
printInputs1()	Print method to write the output to the console.	*void*	*void*
printInputs2()	Print method to write the output to the console by omitting the duplicates.	*void*	*void*
Addingzeros()	Loads the value 0 to the corresponding places that have been removed.	*vector<int> Placesremoved, const PetriNet*	*void*
Deducing_for_Reduction()	Invokes appropriate function for Rule 1, 2 or 3	*vector<int> Rulefollowed, vector<int> Placesremoved, vector<int> Transitionsremoved, vector<int> Placesmerged, const PetriNet*	*void*

Table 6.3 (cont.)			
Deducing_Rule_1()	Uses equation 6.1 to compute all possible integer solutions	*int T_removed, int P_removed, vector<int> Placesremoved, const PetriNet*	*void*
Deducing_Rule_2()	Uses equation 6.2 to compute all possible integer solutions	*int T_removed, int P_removed, vector<int> Placesremoved, const PetriNet*	*void*
Merge()	Uses equation 6.4 to compute all possible integer solutions	*int Mergedplace1, int Mergedplace2, const PetriNet*	*void*

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

Name	Type	Description
finalmineles	*vector<MarkingVector* > >*	Represents the minimal elements of the original PN and is subject to changes through the code.
finalmineles_global	*vector<MarkingVector* > >*	Used to omit all the duplicate members.
noOfplace_reduced	*int*	Represents the number of places of the reduced PetriNet.

Table 6.4 (cont.)		
P_removed	*int*	Represents the place that has been removed that is used in the current computation step.
T_removed	*int*	Represents the transition that has been removed that is used in the current computation step.

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 ouput file has the same format as the input file and is the 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.

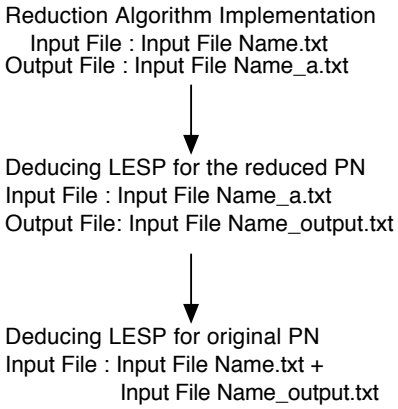


Figure 6.16: Flow of Implementation

6.4.1 Illustrations

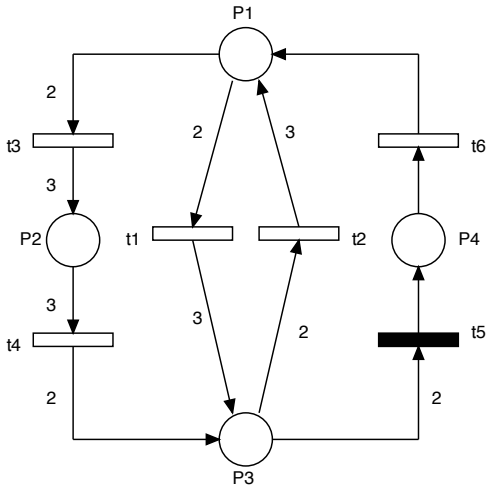


Figure 6.17: Example-1

```
4 6
2 0 2 0 0 0
0 0 0 3 0 0
0 2 0 0 2 0
0 0 0 0 0 1
0 3 0 0 0 1
0 0 3 0 0 0
3 0 0 2 0 0
0 0 0 0 1 0
2 0 0 0
0 0 0 0 1 0
```

Figure 6.18: Input file for Example-1

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

Outputs :
    T  1  2  3  4  5  6
 P
   1   .  3  .  .  .  1
   2   .  .  3  .  .  .
   3   3  .  .  2  .  .
   4   .  .  .  .  1  .

Tansition T3and place P2can be removed
Place P2and transition T4can be removed
Place P4and transition T6can be removed
----------------------------------------------------------------------
new initial Marking( 2 0 0  )
new Input :
    T  1  2  3  4  5
 P
   1   2  .  2  .  .
   2   .  2  .  2  .
   3   .  .  .  .  1

new Output :
    T  1  2  3  4  5
 P
   1   .  3  .  .  1
   2   3  .  2  .  .
   3   .  .  .  1  .

----------------------------------------------------------------------
new initial Marking( 2 0  )
new Input :
    T  1  2  3  4
 P
   1   2  .  2  .
   2   .  2  .  2

new Output :
    T  1  2  3  4
 P
   1   .  3  .  1
   2   3  .  2  .
```

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

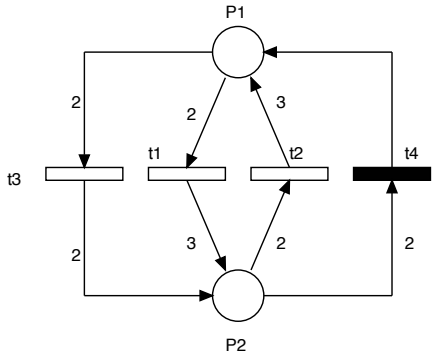


Figure 6.20: Reduced Example-1

```
./PN_minele 1c_new.txt_a.txt


 Incidence Matrix :

    T  1  2  3  4
 P
   1  -2  3 -2  1
   2   3 -2  2 -2

 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
 --------------------------------
 t4


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

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

checking if fine
 This is An LESP
```

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

```
./reduction_deducingminele 1c_new.txt 1c_output.txt
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
-1
Number of places for reduced net:
2
no of reduced places 2
open file name: 1c_output.txt
( 0 2 )

( 2 0 )

 Initial Marking : ( 2 0 0 0 )

Inputs :
    T  1  2  3  4  5  6
 P
   1   2  .  2  .  .  .
   2   .  .  .  3  .  .
   3   .  2  .  .  2  .
   4   .  .  .  .  .  1

Outputs :
    T  1  2  3  4  5  6
 P
   1   .  3  .  .  .  1
   2   .  .  3  .  .  .
   3   3  .  .  2  .  .
   4   .  .  .  .  1  .

Controllable transitions( 0 0 0 0 1 0 )

number of places 4
new final elements:
( 0 0 2 0 )

( 2 0 0 0 )

( 0 3 0 0 )

( 0 0 0 2 )

( 1 0 0 1 )
```

Figure 6.22: Deducing minimal elements for Example-1

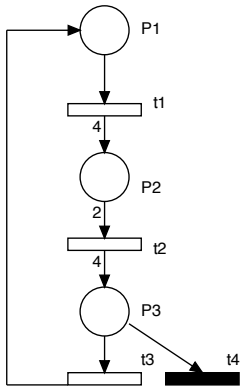


Figure 6.23: Example-2

```
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
```

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  .  .

Tansition T1and place P2can be removed
Place P2and transition T2can be removed
-----------------------------------------------------------------
new initial Marking( 1 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

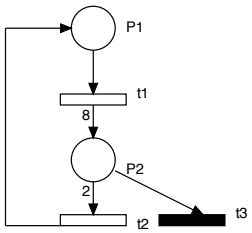


Figure 6.26: Reduced Example-2

```
./PN_minele Example_2.txt_a.txt


 Incidence Matrix :

    T  1  2  3
 P
   1  -1  1  .
   2   8 -2 -1

 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

```
./reduction_deducingminele Example_2.txt  Example_2_output.txt
places removed:
2
-1
transitions removed in the same order:
2
-1
places merged:
-1
Rule followed in same order: - Enter 1 for Rule 1 or 2 for Rule 2:
2
-1
Number of places for reduced net:
2
no of reduced places 2
open file name: Example_2_output.txt
( 0 2 )

( 1 0 )

 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  .  .

Controllable transitions( 0 0 0 1 )

number of places 3
new final elements:
( 0 0 2 )

( 1 0 0 )

( 0 2 0 )
```

Figure 6.28: Deducing minimal elements for Example-2

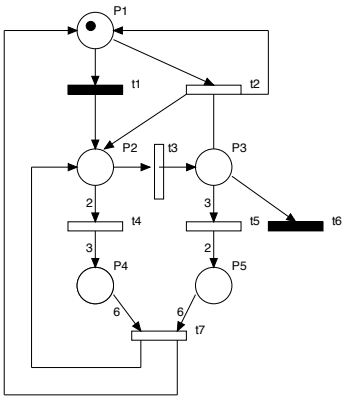


Figure 6.29: Example-3

```
5 7
1 1 0 0 0 0 0
0 0 1 2 0 0 0
0 0 0 0 3 1 0
0 0 0 0 0 0 6
0 0 0 0 0 0 6
0 1 0 0 0 0 1
1 1 0 0 0 0 1
0 1 1 0 0 0 0
0 0 0 3 0 0 0
0 0 0 0 2 0 0
1 0 0 0 0
1 0 0 0 0 1 0
```

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   1  1  .  .  .  .  .
   2   .  .  1  2  .  .  .
   3   .  .  .  .  3  1  .
   4   .  .  .  .  .  .  6
   5   .  .  .  .  .  .  6

Outputs :
    T  1  2  3  4  5  6  7
 P
   1   .  1  .  .  .  .  1
   2   1  1  .  .  .  .  1
   3   .  1  1  .  .  .  .
   4   .  .  .  3  .  .  .
   5   .  .  .  .  2  .  .
Tansition T4and place P4can be removed
Tansition T5and place P5can be removed
-------------------------------------------------------------------
new initial Marking( 1 0 0 0  )
new Input :
    T  1  2  3  4  5  6
 P
   1   1  1  .  .  .  .
   2   .  .  1  .  .  4
   3   .  .  .  3  1  .
   4   .  .  .  .  .  6

new Output :
    T  1  2  3  4  5  6
 P
   1   .  1  .  .  .  1
   2   1  1  .  .  .  1
   3   .  1  1  .  .  .
   4   .  .  .  2  .  .

-------------------------------------------------------------------
new initial Marking( 1 0 0  )
new Input :
    T  1  2  3  4  5
 P
   1   1  1  .  .  .
   2   .  .  1  .  4
   3   .  .  .  1  9

new Output :
    T  1  2  3  4  5
 P
   1   .  1  .  .  1
   2   1  1  .  .  1
   3   .  1  1  .  .
```

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

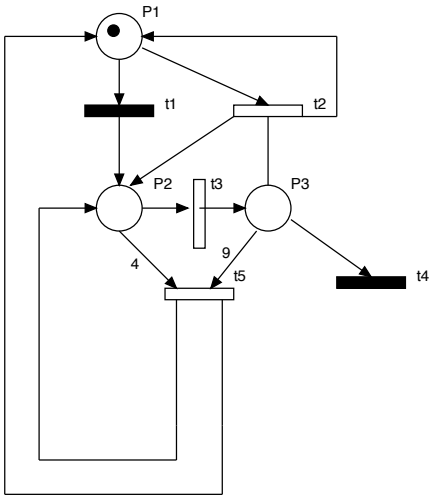


Figure 6.32: Reduced Example-3

```
./PN_minele wtnew5.txt_a.txt


 Incidence Matrix :

    T  1  2  3  4  5
P
   1  -1 *S  .  .  1
   2   1  1 -1  . -3
   3   .  1  1 -1 -9

 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

```
./reduction_deducingminele wtnew5.txt wtnew5_output.txt
places removed:
4
5
-1
transitions removed in the same order:
4
5
-1
places merged:
-1
Rule followed in same order: - Enter 1 for Rule 1 or 2 for Rule 2:
1
1
-1
Number of places for reduced net:
3
no of reduced places 3
open file name: wtnew5_output.txt
( 1 0 0 )

 Initial Marking : ( 1 0 0 0 0 )

Inputs :
    T  1  2  3  4  5  6  7
 P
   1   1  1  .  .  .  .  .
   2   .  .  1  2  .  .  .
   3   .  .  .  .  3  1  .
   4   .  .  .  .  .  .  6
   5   .  .  .  .  .  .  6

Outputs :
    T  1  2  3  4  5  6  7
 P
   1   .  1  .  .  .  .  1
   2   1  1  .  .  .  .  1
   3   .  1  1  .  .  .  .
   4   .  .  .  3  .  .  .
   5   .  .  .  .  2  .  .

Controllable transitions( 1 0 0 0 0 1 0 )

number of places 5
new final elements:
( 1 0 0 0 0 )
```

Figure 6.34: Deducing minimal elements for Example-3

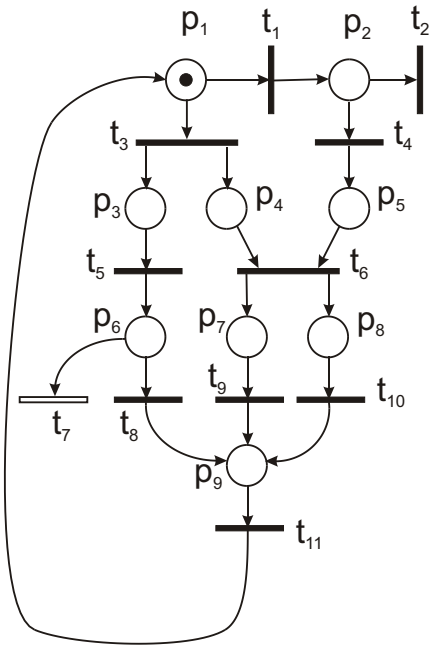


Figure 6.35: PN-9 (cf. [5]).

```
9 11
1 0 1 0 0 0 0 0 0 0 0
0 1 0 1 0 0 0 0 0 0 0
0 0 0 0 1 0 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 1 0 0 0
0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1 0
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 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 0 0 0 0 0 0 0
0 0 0 0 1 0 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 0 1 1 1 0
1 0 0 0 0 0 0 0 0
1 1 1 1 0 0 0 1 0 0 0
```

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

```
./reduction_finalversion pn9.txt
 Initial Marking : ( 1 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  .  .  .  .  .  .  .  .
   2   .  1  .  1  .  .  .  .  .  .  .
   3   .  .  .  .  1  .  .  .  .  .  .
   4   .  .  .  .  .  1  .  .  .  .  .
   5   .  .  .  .  .  1  .  .  .  .  .
   6   .  .  .  .  .  .  1  1  .  .  .
   7   .  .  .  .  .  .  .  .  1  .  .
   8   .  .  .  .  .  .  .  .  .  1  .
   9   .  .  .  .  .  .  .  .  .  .  1

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  1  .

Place P3and transition T5can be removed
Place P7and transition T9can be removed
Place P8and transition T10can be removed
transitions to place9 can be merged
-----------------------------------------------------------------------
new initial Marking( 1 0 0 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7  8  9 10
 P
   1   1  .  1  .  .  .  .  .  .  .
   2   .  1  .  1  .  .  .  .  .  .
   3   .  .  .  .  1  .  .  .  .  .
   4   .  .  .  .  1  .  .  .  .  .
   5   .  .  .  .  .  1  1  .  .  .
   6   .  .  .  .  .  .  .  1  .  .
   7   .  .  .  .  .  .  .  .  1  .
   8   .  .  .  .  .  .  .  .  .  1

new Output :
    T  1  2  3  4  5  6  7  8  9 10
 P
   1   .  .  .  .  .  .  .  .  .  1
   2   1  .  .  .  .  .  .  .  .  .
   3   .  .  1  .  .  .  .  .  .  .
   4   .  .  .  1  .  .  .  .  .  .
   5   .  .  1  .  1  .  .  .  .  .
   6   .  .  .  .  .  .  .  .  .  .
   7   .  .  .  .  1  .  .  .  .  .
   8   .  .  .  .  .  .  1  1  2  .

-----------------------------------------------------------------------
new initial Marking( 1 0 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7  8  9
 P
   1   1  .  1  .  .  .  .  .  .
   2   .  1  .  1  .  .  .  .  .
   3   .  .  .  .  1  .  .  .  .
   4   .  .  .  .  1  .  .  .  .
   5   .  .  .  .  .  1  1  .  .
   6   .  .  .  .  .  .  .  1  .
   7   .  .  .  .  .  .  .  .  1

new Output :
    T  1  2  3  4  5  6  7  8  9
 P
   1   .  .  .  .  .  .  .  .  .  1
   2   1  .  .  .  .  .  .  .  .  .
   3   .  .  1  .  .  .  .  .  .  .
   4   .  .  .  1  .  .  .  .  .  .
   5   .  .  1  .  .  .  .  .  .  .
   6   .  .  .  .  1  .  .  .  .  .
   7   .  .  .  .  1  .  1  2  .

-----------------------------------------------------------------------
new initial Marking( 1 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7  8
 P
   1   1  .  1  .  .  .  .  .
   2   .  1  .  1  .  .  .  .
   3   .  .  .  .  1  .  .  .
   4   .  .  .  .  1  .  .  .
   5   .  .  .  .  .  1  1  .
   6   .  .  .  .  .  .  .  1

new Output :
    T  1  2  3  4  5  6  7  8
 P
   1   .  .  .  .  .  .  .  .  1
   2   1  .  .  .  .  .  .  .  .
   3   .  .  1  .  .  .  .  .  .
   4   .  .  .  1  .  .  .  .  .
   5   .  .  1  .  .  .  .  .  .
   6   .  .  .  .  2  .  1  .
```

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

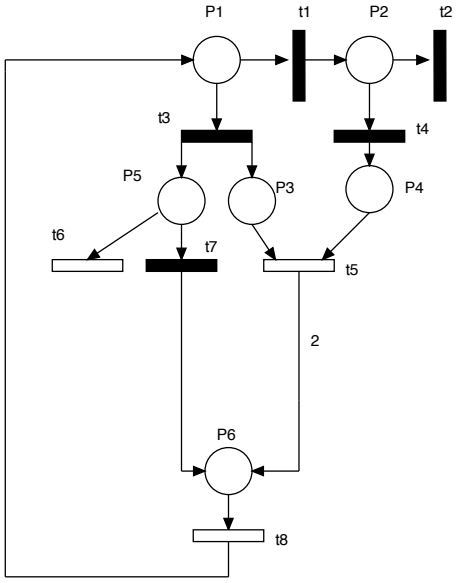


Figure 6.38: Reduced PN-9 (cf. [5]).

```
./PN_minele pn9.txt_a.txt

Incidence Matrix :

   T  1  2  3  4  5  6  7  8
P
  1  -1  . -1  .  .  .  .  1
  2   1 -1  . -1  .  .  .  .
  3   .  .  1  . -1  .  .  .
  4   .  .  .  1 -1  .  .  .
  5   .  .  1  .  . -1 -1  .
  6   .  .  .  .  2  .  1 -1

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


List of Controllable Transitions
--------------------------------
t1 t2 t3 t4 t7


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

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

checking if fine
The loop-test failed for the minimal_element: ( 1 0 0 0 0 0 0 )

The loop-test failed for the minimal_element: ( 0 0 0 0 0 0 1 )

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

 1: ( 0 0 1 1 0 0 0 )
 2: ( 0 1 1 0 0 0 0 )
 3: ( 2 0 0 0 0 0 0 )
 4: ( 1 1 0 0 0 0 0 )
 5: ( 1 0 1 0 0 0 0 )
 6: ( 1 0 0 1 0 0 0 )
 7: ( 1 0 0 0 0 1 0 )
 8: ( 1 0 0 0 0 0 1 )
 9: ( 1 0 0 0 0 0 1 )
10: ( 0 1 0 0 0 0 1 )
11: ( 0 0 1 0 0 0 1 )
12: ( 0 0 0 1 0 1 1 )
13: ( 0 0 0 0 0 1 1 )
14: ( 0 0 0 0 0 0 2 )


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

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

checking if fine
 This is An LESP
```

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

```
./reduction_deducingminele pn9.txt pn9_output.txt
places removed:
3
7
8
-1
transitions removed in the same order:
5
9
10
-1
places merged:
7
8
-1
Rule followed in same order: - Enter 1 for Rule 1 or 2 for Rule 2:
2
2
2
-1
Number of places for reduced net:
6
no of reduced places 6
open file name: pn9_output.txt
( 0 0 1 1 0 0 )

( 0 1 1 0 0 0 )

( 2 0 0 0 0 0 )

( 1 1 0 0 0 0 )

( 1 0 1 0 0 0 )

( 1 0 0 1 0 0 )

( 1 0 0 0 1 0 )

( 0 1 0 0 0 1 )

( 0 0 1 0 0 1 )

( 0 0 0 1 0 1 )

( 0 0 0 0 0 2 )

 Initial Marking : ( 1 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  .  .  .  .  .  .  .  .
   2   .  1  .  1  .  .  .  .  .  .  .
   3   .  .  .  .  1  .  .  .  .  .  .
   4   .  .  .  .  .  1  .  .  .  .  .
   5   .  .  .  .  .  1  .  .  .  .  .
   6   .  .  .  .  .  .  1  1  .  .  .
   7   .  .  .  .  .  .  .  .  1  .  .
   8   .  .  .  .  .  .  .  .  .  1  .
   9   .  .  .  .  .  .  .  .  .  .  1

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  1  .

Controllable transitions( 1 1 1 1 0 0 0 1 0 0 0 )

number of places 9
new final elements:
( 0 0 0 1 1 0 0 0 0 )

( 0 1 0 1 0 0 0 0 0 )

( 2 0 0 0 0 0 0 0 0 )

( 1 1 0 0 0 0 0 0 0 )

( 1 0 0 1 0 0 0 0 0 )

( 1 0 0 0 1 0 0 0 0 )

( 1 0 0 0 0 0 0 0 1 )

( 0 1 0 0 0 0 0 0 1 )

( 0 0 0 1 0 0 0 0 1 )

( 0 0 0 0 1 0 0 0 1 )

( 0 0 0 0 0 0 0 0 2 )

( 1 0 0 0 0 0 1 0 0 )

( 0 1 0 0 0 0 1 0 0 )

( 0 0 0 1 0 0 1 0 0 )

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

( 0 0 0 0 0 0 1 0 1 )

( 1 0 0 0 0 0 0 1 0 )

( 0 1 0 0 0 0 0 1 0 )

( 0 0 0 1 0 0 0 1 0 )

( 0 0 0 0 1 0 0 1 0 )

( 0 0 0 0 0 0 0 2 0 )

( 0 0 0 0 0 0 0 1 1 )

( 0 0 0 0 0 0 1 1 0 )
```

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

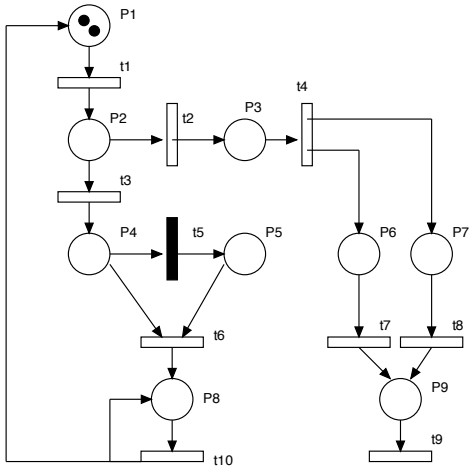


Figure 6.41: PN-13 (cf. [5]).

```
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
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
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 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1
0 0 0 0 0 0 1 1 0 0
2 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
```

Figure 6.42: Input file for PN-13 (cf. [5]).

```
./reduction_finalversion PN-13.txt
 Initial Marking : ( 2 0 0 0 0 0 0 0 0 0  )
Inputs :
    T  1  2  3  4  5  6  7  8  9 10
 P
   1   1  .  .  .  .  .  .  .  .  .  .
   2   .  1  1  .  .  .  .  .  .  .  .
   3   .  .  .  1  .  .  .  .  .  .  .
   4   .  .  .  .  1  1  .  .  .  .  .
   5   .  .  .  .  .  1  .  .  .  .  .
   6   .  .  .  .  .  .  1  .  .  .  .
   7   .  .  .  .  .  .  .  1  .  .  .
   8   .  .  .  .  .  .  .  .  .  .  1
   9   .  .  .  .  .  .  .  .  .  1  .

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

Tansition T2and place P3can be removed
Place P6and transition T7can be removed
Place P7and transition T8can be removed
transitions to place9 can be merged
------------------------------------------------------------------------
new initial Marking( 2 0 0 0 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7  8  9
 P
   1   1  .  .  .  .  .  .  .  .  .
   2   .  1  1  .  .  .  .  .  .  .
   3   .  .  .  1  1  .  .  .  .  .
   4   .  .  .  .  1  .  .  .  .  .
   5   .  .  .  .  .  1  .  .  .  .
   6   .  .  .  .  .  .  1  .  .  .
   7   .  .  .  .  .  .  .  .  .  1
   8   .  .  .  .  .  .  .  .  1  .

new Output :
    T  1  2  3  4  5  6  7  8  9
 P
   1   .  .  .  .  .  .  .  .  . 1
   2   1  .  .  .  .  .  .  .  .  .
   3   .  1  .  .  .  .  .  .  .  .
   4   .  .  .  1  .  .  .  .  .  .
   5   .  .  .  .  1  .  .  .  .  .
   6   .  .  .  .  .  1  .  .  .  .
   7   .  .  .  .  1  .  .  .  .  1
   8   .  .  .  .  .  .  1  2  .  .

------------------------------------------------------------------------
new initial Marking( 2 0 0 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7  8
 P
   1   1  .  .  .  .  .  .  .  .
   2   .  1  1  .  .  .  .  .  .
   3   .  .  .  1  1  .  .  .  .
   4   .  .  .  .  1  .  .  .  .
   5   .  .  .  .  .  1  .  .  .
   6   .  .  .  .  .  .  .  .  1
   7   .  .  .  .  .  .  .  1  .

new Output :
    T  1  2  3  4  5  6  7  8
 P
   1   .  .  .  .  .  .  .  .  1
   2   1  .  .  .  .  .  .  .  .
   3   .  1  .  .  .  .  .  .  .
   4   .  .  .  1  .  .  .  .  .
   5   .  .  1  .  .  .  .  .  .
   6   .  .  .  .  1  .  .  1  .
   7   .  .  1  .  .  2  .  .  .

------------------------------------------------------------------------
new initial Marking( 2 0 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7
 P
   1   1  .  .  .  .  .  .  .
   2   .  1  1  .  .  .  .  .
   3   .  .  .  1  1  .  .  .
   4   .  .  .  .  1  .  .  .
   5   .  .  .  .  .  .  .  1
   6   .  .  .  .  .  1  .  .

new Output :
    T  1  2  3  4  5  6  7
 P
   1   .  .  .  .  .  .  .  1
   2   1  .  .  .  .  .  .  .
   3   .  1  .  .  .  .  .  .
   4   .  .  .  1  .  .  .  .
   5   .  .  .  .  1  .  1  .
   6   .  .  2  .  .  .  .  .
```

Figure 6.43: Output for Reduction part of the algorithm for PN-13 (cf. [5]).

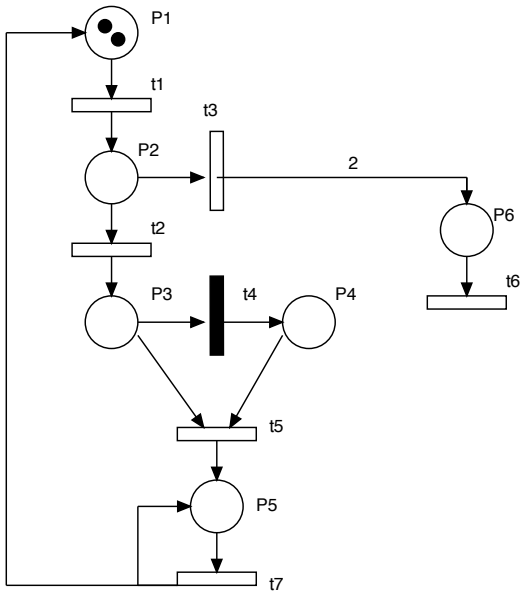


Figure 6.44: Reduced PN-13 (cf. [5]).

```
./PN_minele PN-13.txt_a.txt


 Incidence Matrix :

    T  1  2  3  4  5  6  7
 P
   1  -1  .  .  .  .  .  1
   2   1 -1 -1  .  .  .  .
   3   .  1  . -1 -1  .  .
   4   .  .  .  1 -1  .  .
   5   .  .  .  .  1  . *S
   6   .  .  2  .  . -1  .

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


 List of Controllable Transitions
 --------------------------------
 t4


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

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

checking if fine
 This is An LESP
```

Figure 6.45: Minimal Elements of the Reduced net PN-13 (cf. [5]).

```
./reduction_deducingminele PN-13.txt PN-13_output.txt
places removed:
3
6
7
-1
transitions removed in the same order:
2
7
8
-1
places merged:
6
7
-1
Rule followed in same order: - Enter 1 for Rule 1 or 2 for Rule 2:
1
2
2
-1
Number of places for reduced net:
6
no of reduced places 6
open file name: PN-13_output.txt
( 0 0 0 0 1 0 )

( 0 0 1 1 0 0 )

( 0 0 2 0 0 0 )

 Initial Marking : ( 2 0 0 0 0 0 0 0 0 )

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

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

Controllable transitions( 0 0 0 0 1 0 0 0 0 0 )

number of places 9
new final elements:
( 0 0 0 0 0 0 0 0 1 0 )

( 0 0 0 1 1 0 0 0 0 )

( 0 0 0 2 0 0 0 0 0 )
```

Figure 6.46: Deducing minimal elements for PN-13 (cf. [5]).

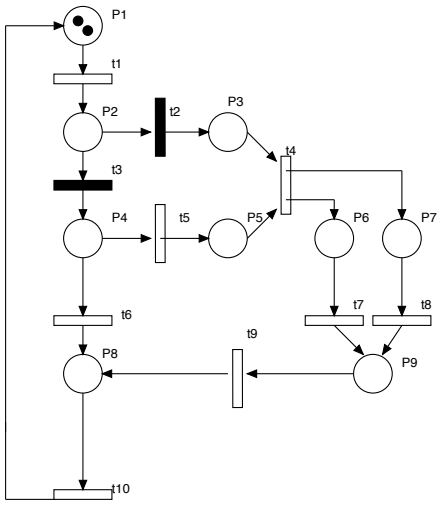


Figure 6.47: PN-11 (cf. [5]).

```
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 1 0 0 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
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
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 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 0
0 0 0 0 0 0 1 1 0 0
2 0 0 0 0 0 0 0 0
0 1 1 0 0 0 0 0 0 0
```

Figure 6.48: Input file for PN-11 (cf. [5]).

```
./reduction_finalversion pn11
 Initial Marking : ( 2 0 0 0 0 0 0 0 0 0  )
Inputs :
    T  1  2  3  4  5  6  7  8  9 10
 P
   1   1  .  .  .  .  .  .  .  .  .  .
   2   .  1  1  .  .  .  .  .  .  .  .
   3   .  .  .  1  .  .  .  .  .  .  .
   4   .  .  .  .  1  1  .  .  .  .  .
   5   .  .  .  1  .  .  .  .  .  .  .
   6   .  .  .  .  .  .  .  1  .  .  .
   7   .  .  .  .  .  .  .  .  1  .  .
   8   .  .  .  .  .  .  .  .  .  .  1
   9   .  .  .  .  .  .  .  .  .  1  .

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

Place P6and transition T7can be removed
Place P7and transition T8can be removed
transitions to place9 can be merged
------------------------------------------------------------------
new initial Marking( 2 0 0 0 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7  8  9
 P
   1   1  .  .  .  .  .  .  .  .  .
   2   .  1  1  .  .  .  .  .  .  .
   3   .  .  .  1  .  .  .  .  .  .
   4   .  .  .  .  1  1  .  .  .  .
   5   .  .  .  1  .  .  .  .  .  .
   6   .  .  .  .  .  .  .  1  .  .
   7   .  .  .  .  .  .  .  .  .  1
   8   .  .  .  .  .  .  .  .  1  .

new Output :
    T  1  2  3  4  5  6  7  8  9
 P
   1   .  .  .  .  .  .  .  .  .  1
   2   1  .  .  .  .  .  .  .  .  .
   3   .  1  .  .  .  .  .  .  .  .
   4   .  .  1  .  .  .  .  .  .  .
   5   .  .  .  .  1  .  .  .  .  .
   6   .  .  .  1  .  .  .  1  .  .
   7   .  .  .  .  .  .  .  .  .  .
   8   .  .  .  1  .  .  2  .  .  .

------------------------------------------------------------------
new initial Marking( 2 0 0 0 0 0 0 0  )
new Input :
    T  1  2  3  4  5  6  7  8
 P
   1   1  .  .  .  .  .  .  .  .
   2   .  1  1  .  .  .  .  .  .
   3   .  .  .  1  .  .  .  .  .
   4   .  .  .  .  1  1  .  .  .
   5   .  .  .  1  .  .  .  .  .
   6   .  .  .  .  .  .  .  .  1
   7   .  .  .  .  .  .  .  1  .

new Output :
    T  1  2  3  4  5  6  7  8
 P
   1   .  .  .  .  .  .  .  .  1
   2   1  .  .  .  .  .  .  .  .
   3   .  1  .  .  .  .  .  .  .
   4   .  .  1  .  .  .  .  .  .
   5   .  .  .  .  1  .  .  .  .
   6   .  .  .  .  .  .  1  1  .
   7   .  .  .  2  .  .  .  .  .
```

Figure 6.49: Output for Reduction part of the algorithm for PN-11 (cf. [5]).

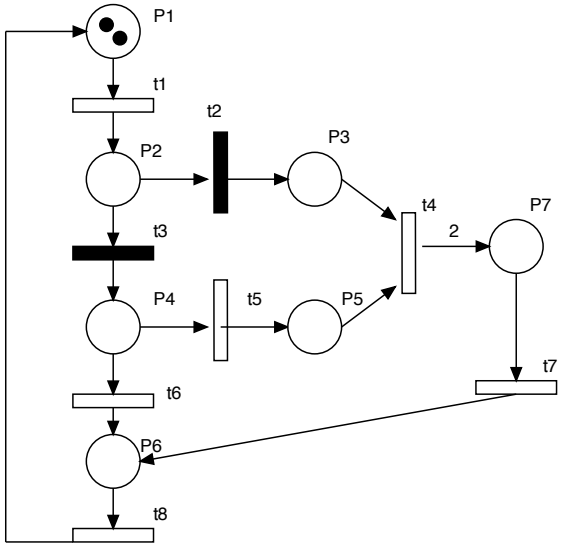


Figure 6.50: Reduced PN-11 (cf. [5]).

```
./PN_minele pn11_a.txt

Incidence Matrix :

    T  1  2  3  4  5  6  7  8
 P
   1  -1  .  .  .  .  .  .  1
   2   1 -1 -1  .  .  .  .  .
   3   .  1  . -1  .  .  .  .
   4   .  .  1  . -1 -1  .  .
   5   .  .  . -1  1  .  .  .
   6   .  .  .  .  .  1  1 -1
   7   .  .  .  2  .  . -1  .

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


 List of Controllable Transitions
 --------------------------------
 t2 t3


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

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

```
./reduction_deducingminele pn11 pn11_output.txt
places removed:
6
7
-1
transitions removed in the same order:
7
8
-1
places merged:
6
7
-1
Rule followed in same order: - Enter 1 for Rule 1 or 2 for Rule 2:
2
2
-1
Number of places for reduced net:
7
no of reduced places 7
open file name: pn11_output.txt
( 1 0 0 0 0 0 1 )

( 0 1 0 0 0 0 1 )

( 0 0 1 0 0 0 1 )

( 0 0 0 1 0 0 1 )

( 0 0 0 0 1 0 1 )

( 0 0 0 0 0 1 1 )

( 0 0 0 0 0 0 2 )

( 1 0 0 0 0 1 0 )

( 0 1 0 0 0 1 0 )

( 0 0 1 0 0 1 0 )

( 0 0 0 1 0 1 0 )

( 0 0 0 0 1 1 0 )

( 0 0 0 0 0 2 0 )

( 1 0 0 0 1 0 0 )

( 0 1 0 0 1 0 0 )

( 0 0 1 0 1 0 0 )

( 1 0 0 1 0 0 0 )
( 0 0 0 0 1 0 0 1 )

( 0 0 0 0 0 0 0 1 1 )

( 0 0 0 0 0 0 0 0 2 )

( 1 0 0 0 0 0 0 1 0 )

( 0 1 0 0 0 0 0 1 0 )

( 0 0 1 0 0 0 0 1 0 )

( 0 0 0 1 0 0 0 1 0 )

( 0 0 0 0 1 0 0 1 0 )

( 0 0 0 0 0 0 0 2 0 )

( 1 0 0 0 1 0 0 0 0 )

( 0 1 0 0 1 0 0 0 0 )

( 0 0 1 0 1 0 0 0 0 )

( 1 0 0 1 0 0 0 0 0 )

( 0 1 0 1 0 0 0 0 0 )

( 0 0 1 1 0 0 0 0 0 )

( 1 0 1 0 0 0 0 0 0 )

( 0 1 1 0 0 0 0 0 0 )

( 1 1 0 0 0 0 0 0 0 )

( 0 2 0 0 0 0 0 0 0 )

( 2 0 0 0 0 0 0 0 0 )

( 1 0 0 0 0 1 0 0 0 )

( 0 1 0 0 0 1 0 0 0 )

( 0 0 1 0 0 1 0 0 0 )

( 0 0 0 1 0 1 0 0 0 )

( 0 0 0 0 1 1 0 0 0 )

( 0 0 0 0 0 1 0 1 0 )

( 0 0 0 0 0 2 0 0 0 )

( 0 0 1 0 1 0 0 0 0 )

( 0 0 1 1 0 0 0 0 0 )

( 1 0 1 0 0 0 0 0 0 )

( 0 1 1 0 0 0 0 0 0 )

( 0 2 0 0 0 0 0 0 0 )

( 2 0 0 0 0 0 0 0 0 )

 Initial Marking : ( 2 0 0 0 0 0 0 0 0 )

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

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

Controllable transitions( 0 1 1 0 0 0 0 0 0 0 )

number of places 9
new final elements:
( 1 0 0 0 0 0 0 0 1 )

( 0 1 0 0 0 0 0 0 1 )

( 0 0 1 0 0 0 0 0 1 )

( 0 0 0 1 0 0 0 0 1 )
( 0 0 0 0 1 0 0 0 1 )

( 1 0 0 0 0 1 0 0 0 )

( 0 1 0 0 0 0 1 0 0 )

( 0 0 1 0 0 0 1 0 0 )

( 0 0 0 1 0 0 1 0 0 )

( 0 0 0 0 1 0 1 0 0 )

( 0 0 0 0 0 1 1 0 0 )

( 0 0 0 0 0 0 2 0 0 )

( 0 0 0 0 0 0 1 0 1 )

( 0 0 0 0 0 1 1 0 0 )
```

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 Figure6.40.

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 $\{(1\,0\,0\,0\,0)^T$, $(0\,0\,0\,6\,6)^T, (0\,0\,3\,6\,4)^T, (0\,0\,6\,6\,2)^T, (0\,0\,9\,6\,0)^T\}$. However, once the PN is simplified the minimally restrictive LESP for the reduced PN Figure 6.32 has only one minimal element $\{(1\,0\,0)^T\}$. 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 $t_2 \xrightarrow{4} p_1 \xrightarrow{2} t_1 \xrightarrow{4} p_2$ is replaced by $t_2 \xrightarrow{8} \widetilde{p}_2$

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 enfore 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.

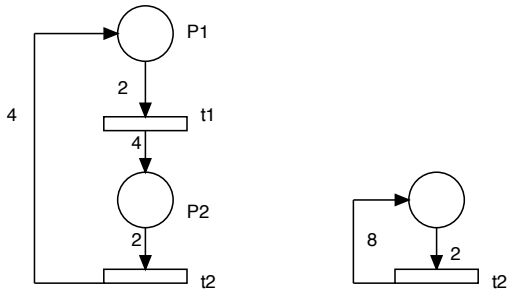


Figure 6.53: Example-5

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 LESPs 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 techinques 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 similarilty 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 exisiting 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.

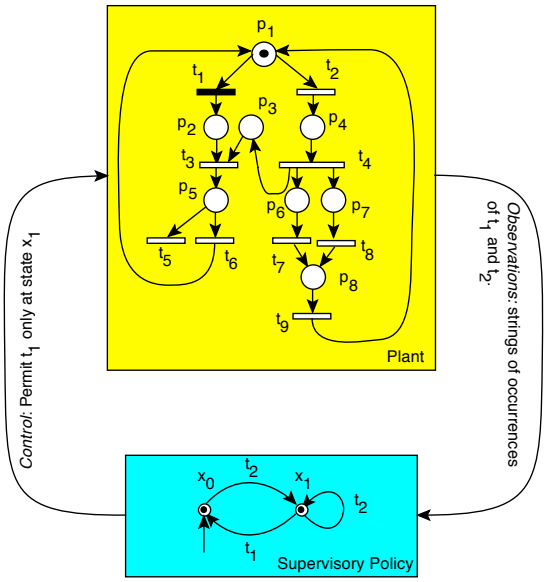
Figure 7.1 shows two different LESPs for a PN $N_{10}(\mathbf{m}_{10}^0)$. Policy 1 uses an event-based LESP. Policy 2 uses the $\Delta(N)$-set based LESP. The $\Delta(N)$-set based LESP of Policy 2 is minimally-restrictive (cf. chapter 3 of this thesis). Policy 1 is <u>not</u> minimally restrictive, as $(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)^T \rightarrow (t_2t_4t_7t_8t_9)^2t_1 \rightarrow (2\ 1\ 2\ 0\ 0\ 0\ 0\ 0)^T$ 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 $N_1(\mathbf{m}_1^0)$, when projected on the alphabet-set $\{t_1, t_2\}$ is a subset of the set identified by the regular-expression $(t_2t_2^*t_1)^*$. Since,

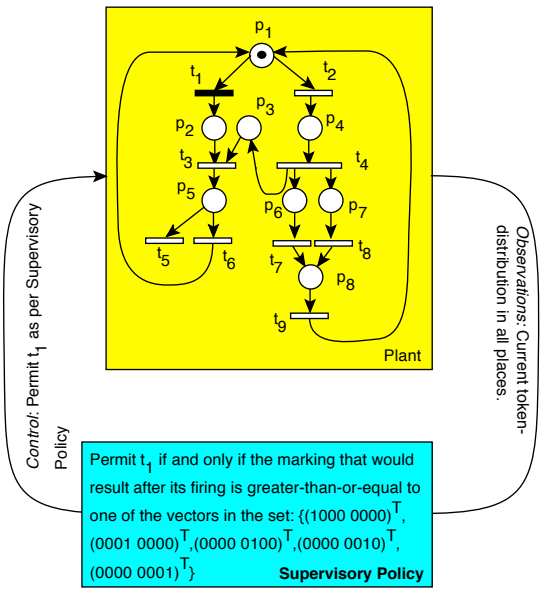
$$\begin{aligned}(t_2t_4t_7t_8t_9)^2t_1\mid_{\{t_1,t_2\}} &= t_2^2t_1(\in (t_2t_2^*t_1)^*) \text{ and}\\ (t_2t_4t_7t_8t_9)^2t_1^2\mid_{\{t_1,t_2\}} &= t_2^2t_1^2(\notin (t_2t_2^*t_1)^*),\end{aligned}$$

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 <u>not</u> 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.



(a) Policy 1



(b) Policy 2

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

REFERENCES

[1] N. Somnath and R. Sreenivas, "On Deciding the Existence of a Liveness Enforcing Supervisory Policy in a Class of Partially-Controlled General Free-Choice Petri Nets," *IEEE Transactions on Automation Science and Engineering*, vol. 10, pp. 1157–1160, October 2013.

[2] E. Salimi, N. Somnath, and R. Sreenivas, "A Software Tool for Live-Lock Avoidance in Systems Modeled Using a Class of Petri Nets," *International Journal of Computer Science, Engineering and Applications (IJCSEA)*, vol. 5, no. 2, pp. 1–13, April 2015.

[3] E. Salimi, N. Somnath, and R. Sreenivas, "On supervisory policies that enforce liveness in controlled petri nets that are similar," in *Proceedings of the 7th IEEE International Conference on Cybernetics and Intelligent Systems (CIS) and the 7th IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, Angkor Wat, Cambodia, July 2015.

[4] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[5] S. Chandrasekaran, N. Somnath, and R. Sreenivas, "A Software Tool for the Automatic Synthesis of Minimally Restrictive Liveness Enforcing Supervisory Policies for a class of General Petri Nets," *Journal of Intelligent Manufacturing*, 2014, to appear (DOI 10.1007/s10845-014-0888-5).

[6] J. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[7] R. Valk and M. Jantzen, "The residue of vector sets with applications to decidability problems in Petri nets," *Acta Informatica*, vol. 21, pp. 643–674, 1985.

[8] W. Reisig, *Petri Nets*. Berlin: Springer-Verlag, 1985.

[9] R. Sreenivas, "On Commoner's liveness theorem and supervisory policies that enforce liveness in Free-choice Petri nets," *Systems & Control Letters*, vol. 31, pp. 41–48, 1997.

[10] K. Barkaoui and J. Pradat-Peyre, "On Liveness and Controlled Siphons in Petri Nets," vol. 1091, January 1996, Proc. the 17th International Conference on Applications and Theory of Petri Nets, Osaka, Japan, pages 57-72.

[11] R. Sreenivas, "On the existence of supervisory policies that enforce liveness in partially controlled free-choice petri nets," *IEEE Transactions on Automatic Control*, vol. 57, no. 2, pp. 435–449, February 2012.

[12] R. Sreenivas, "On the existence of supervisory policies that enforce liveness in discrete-event dynamic systems modeled by controlled Petri nets," *IEEE Transactions on Automatic Control*, vol. 42, no. 7, pp. 928–945, July 1997.

[13] R. Sreenivas, "On a Decidable Class of Partially Controlled Petri Nets With Liveness Enforcing Supervisory Policies," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, pp. 1256–1261, August 2013.

[14] S. Chandrasekaran, "Object-oriented implementation of the minimally restrictive liveness enforcing supervisory policy in a class of petri nets," M.S. thesis, University of Illinois at Urbana-Champaign, Industrial and Enterprise Systems Engineering, December 2012.

[15] S. Chandrasekaran and R. Sreenivas, "A software tool for the synthesis of supervisory policies that avoid livelocks in petri net models of manufacturing- and service-systems," in *Proceedings of the XVI Annual International Conference of the Society of Operations Management (SOM-12)*, New Delhi, India, December 2012.

[16] S. Chandrasekaran and R. Sreenivas, "On the automatic generation of the minimally restrictive liveness enforcing supervisory policy for manufacturing- and service-systems modeled by a class of general free choice petri nets," in *Proceedings of the IEEE International Conference on Networking, Sensing and Control (ICNSC-13)*, Paris, France, April 2013, session WeC01.3.

[17] A. Giua, "Petri nets as discrete event models for supervisory control," Ph.D. dissertation, ECSE Dept., Rensselaer Polytechnic Institute, Troy, NY., 1992.

[18] J. Moody and P. Antsaklis, *Supervisory Control of Discrete Event Systems using Petri Nets*. MA: Kluwer Academic Publishers, 1998.

[19] M. Iordache and P. Ansaklis, *Supervisory control of Concurrent Systems: A Petri net Structural Approach*. MA: Kulwer Academic Publishers, 2006.

[20] S. Reveliotis, E. Roszkowska, and J. Choi, "Generalized algebraic deadlock avoidance policies for sequential Resource Allocation Systems," *IEEE Transactions on Automatic Control*, vol. 53, no. 7, pp. 2345–2350, December 2007.

[21] A. Ghaffari, N. Rezg, and X. Xie, "Design of a Live and Maximally Permissive Petri net Controller using the Theory of Regions," *IEEE Transactions on Robotics and Automation*, vol. 19, no. 1, pp. 137–142, January 2003.

[22] D. Liu, Z. Li, and M. Zhou, "Liveness of an extended S^3PR," *Automatica*, vol. 46, pp. 1008–1018, 2010, also, Erratum to "Liveness of an extended S^3PR", Automatica, 48, (2012), 1003-1004.

[23] O. Marchetti and A. Munier-Kordon, "A sufficient condition for the liveness of weighted event graphs," *European Journal of Operations Research*, vol. 197, pp. 532–540, 2009.

[24] F. Basile, L. Recalde, P. Chiacchio, and M. Silva, "Closed-loop Live Marked Graphs under Generalized Mutual Exclusion Constraint Enforcement," *Discrete Event Dynamic Systems*, vol. 19, no. 1, pp. 1–30, 2009.

[25] R. Sreenivas, "Some observations on supervisory policies that enforce liveness in partially controlled Free Choice Petri nets," *Mathematics and Computers in Simulation*, vol. 70, pp. 266–274, 2006.

[26] E. Best, *Lecture Notes in Computer Science*. Springer-Verlag, 1987, vol. 254, ch. Structure Theory of Petri Nets: The Free Choice Hiatus.

[27] V. Deverakonda and R. Sreenivas, "On a sufficient information structure for supervisory policies that enforce liveness in a class of general petri nets," *IEEE Transactions on Automatic Control*, vol. 60, no. 7, pp. 1915–1921, July 2015.

[28] E. Salimi, N. Somnath, and R. Sreenivas, "A tutorial on the synthesis of the maximally permissive liveness enforcing supervisory policy in discrete-event/discrete-state systems modeled by a class of general petri nets," in *Proceedings of the Indian Control Conference (ICC-15)*, January 2015, I.I.T. Madras.

[29] R. Sreenivas, "On asymptotically efficient solutions for a class of supervisory control problems," *IEEE Transactions on Automatic Control*, vol. 41, no. 12, pp. 1736–1750, December 1996.

[30] P. Ramadge and W. Wonham, "The control of Discrete Event Systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, January 1989.

[31] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event systems," *SIAM Journal of Control and Optimization*, vol. 25, no. 1, pp. 206–230, January 1987.

[32] W. Wonham and P. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM Journal of Conrol and Optimization*, vol. 25, no. 3, pp. 637–659, May 1987.

[33] R. Sreenivas, "Towards a system theory for interconnected condition/event systems," Ph.D. dissertation, Carnegie Mellon University, 1990.

[34] R. Sreenivas, "An application of independent, increasing, free-choice petri nets to the synthesis of policies that enforce liveness in arbitrary petri nets," *Automatica*, vol. 34, no. 12, pp. 1613–1615, December 1998.

[35] R. Sreenivas, "On supervisory policies that enforce liveness in in a class of completely controlled petri nets obtained via refinement," *IEEE Transactions on Automatic Control*, vol. 44, no. 1, pp. 173–177, January 1999.

[36] R. Sreenivas, "On supervisory policies that enforce liveness in completely controlled petri nets with directed cut-places and cut-transitions," *IEEE Transactions on Automatic Control*, vol. 44, no. 6, pp. 1221–1225, June 1999.

[37] R. Sreenivas and B. Krogh, "On condition/event systems with discrete state realizations," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 1, pp. 209–236, 1991.

[38] R. Sreenivas, "A note on deciding the controllability of a language K with respect to a language L," *IEEE Trans. on Automatic Control*, vol. 38, no. 4, April 1993.

[39] R. Sreenivas, "On a weaker notion of controllability of a language K with respect to a language L," *IEEE Trans. on Automatic Control*, vol. 38, no. 9, September 1993.

[40] R. Sreenivas, "On minimal representations of petri net languages," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 799–804, May 2006.

[41] L. Li, C. Hadjicostis, and R. S. Sreenivas, "Designs of bisimilar petri net controllers with fault tolerance capabilities," *IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans*, vol. 38, no. 1, pp. 207–217, January 2008.

[42] L. Li, C. Hadjicostis, and R. S. Sreenivas, "Fault detection and identification in petri net controllers," in *Proceedings of the 43rd IEEE Conference on Decision and Control (CDC)*, Bahamas, December 2004, pp. 5248–5253.