THE COMPLEXITY OF PETRI NET TRANSFORMATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE,
FACULTY OF SCIENCE
AT THE UNIVERSITY OF CAPE TOWN
IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Stephen Richard Donaldson
September 1993

Supervised by
Professor P. S. Kritzinger

Abstract

This study investigates the complexity of various reduction and synthesis Petri net transformations. Transformations that preserve liveness and boundedness are considered. Liveness and boundedness are possibly the two most important properties in the analysis of Petri nets. Unfortunately, although decidable, determining such properties is intractable in the general Petri net.

The thesis shows that the complexity of these properties imposes limitations on the power of any reduction transformations to solve the problems of liveness and boundedness. Reduction transformations and synthesis transformations from the literature are analysed from an algorithmic point of view and their complexity established. Many problems regarding the applicability of the transformations are shown to be intractable. For reduction transformations this confirms the limitations of such transformations on the general Petri net. The thesis suggests that synthesis transformations may enjoy better success than reduction transformations, and because of problems establishing suitable goals, synthesis transformations are best suited to interactive environments.

The complexity of complete reducibility, by reduction transformation, of certain classes of Petri nets, as proposed in the literature, is also investigated in this thesis. It is concluded that these transformations are tractable and that reduction transformation theory can provide insight into the analysis of liveness and boundedness problems, particularly in subclasses of Petri nets.

Acknowledgements

During the course of this study, a number of people assisted me in various ways. I'd like to express my appreciation for their help:

- Members of the Computer Science Department, in particular Donald Cook and Prof. Peter Wood for many helpful discussions, and Mary Wood for her expeditious handling of admin. problems.
- Members of the Mathematics Department, for answering questions on a number of technical issues, specifically Prof. Ronnie Becker and Dr Ruth Smart.
- Colleagues in the Data Network Architectures Laboratory and fellow MSc students: Geoffrey Letsoalo, Andrew Luppnow, Dieter Polzin and Sigrid Ewert. In particular, John Green, who helped verify the algorithms.
- Libraries: Inter-library loans staff and Fiona Jones of the Engineering and Science Library.
- Understanding friends with "open houses": Patric and Karen Hayward, and Craig and Peta Harrison.
- Dr Falko Bause of the Universität Dortmund, Germany, for his expertise, ideas and many hours discussing problems.
- My parents, for all that they have meant to me and done for me over the years.
- Prof. Pieter Kritzinger for keeping up such a keen interest in my work, for his support, guidance and insight throughout this study.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since their introduction in the early 1960's Petri nets have become accepted models, suitable for many types of systems: For example they can model concurrency, synchronization and finite resources. They are thus suitable for implementing qualitative models of mutual exclusion, the dining philosophers problem and the readers and writers problem, (all described in [Pet81]). Also, larger models, such as models of protocols of large distributed systems, for example, connection management services (CMS) of the Manufacturing Message Specification (MMS), are possible [WGR89]. Quantitative modelling is also possible with one of the timed extensions of the Petri net model, an example of this being models of multiprocessor architectures [ABC86]. Petri net models have also been used as a formal specification, for example, the specification of the CPU in the CDC 6600 [Mol89]. Petri nets are not only suited to computer systems either; for example, they have also been used for manufacturing systems [SX92].

Over the years a rich mathematical analysis of Petri nets has been developed which has helped to establish the effectiveness of Petri nets as a modelling technique. Petri nets are accepted readily as a modelling technique as they provide an easily understood graphical representation of the system being modeled; Petri nets are thus also suitable for communicating ideas about systems.

The graphical nature of Petri nets allows the easy formulation of models of systems. Furthermore, Petri nets are also a mathematical modelling tool, and as such make rigorous qualitative analysis tools, suitable for modelling a large body of applications, accessible to nonmathematicians. These tools usually provide a graphical user interface whereby the user *draws* the model of the system to be modelled. Examples of such systems are the QPN-Tool [BK91], and GreatSPN [Chi85]. The rich theory of Petri nets that has been developed could also been used indirectly in tools. For example, [Kri93] proposes the

translation of SDL (a CCITT formal description technique used for protocol specification) into a Petri net model in order that certain qualitative analyses be performed on the protocol, or quantitative analysis if the Petri net model is one of the timed Petri net models.

A common analysis technique of Petri nets is based on the matrices that represent the arcs and their associated weights between the nodes of the Petri net. Two types of node are distinguished: places, which hold tokens, are used to indicate resources or conditions, and transitions which are used to indicate "events". This analysis technique is based on equations and inequalities of these matrices. A problem with this technique is that certain important questions about modelled systems, in particular configurations (Petri nets with an initial marking or state), cannot be answered; for example, the absence of deadlocks and the overflowing of finite resources. By absence of deadlock is meant the ability of the system to always repeat every modelled "event" no matter how the Petri net has evolved. The matrix technique gives a sufficient condition for the configuration not to overflow any finite resources but does not give a necessary condition for this. While this matrix equation and inequality based technique cannot be used to decide whether a particular configuration is free from deadlocks, it can, however, be used to decide whether an initial state exists for a given Petri net such that the resulting configuration is live.

Another analysis technique is based on the enumeration of (possibly) all the states that a Petri net can reach from a particular initial marking. This technique requires the generation of the so-called coverability tree in which each node represents a certain state or set of states and the arcs correspond to the "events" leading to these states. The coverability tree can be used to decide whether the modelled system could overflow any bounded resources and, if so, whether the system is absent of deadlocks. However, the size of the coverability tree in terms of number of nodes has been shown by Lipton to be exponential in the number of places, transitions and initial marking [Pet81, JLL77]. Consequently, the problems of deciding those properties that require the generation of the coverability tree are intractable.

A number of authors have proposed property preserving transformations. These transformations can either be used to reduce the size of the Petri net, and possibly the size of the problem or they can be used to build up Petri nets with the desired properties and thus they do not need their coverability tree to be generated. Transformations used in the latter sense are termed synthesis transformations while those used in the former are termed reduction transformations. Usually properties are preserved by a transformation in both directions and such transformations can be used as both reduction and synthesis transformations. Properties preserved in only one direction can be used as synthesis

transformations as it is only the resultant system that is required to have the properties.

Another proposed use of reduction transformations (see, for example, [Ber86a]) is in tools for engineers, where the engineer modelling a system does not have to be concerned with certain redundancies, for example, as the modelling tool will remove these prior to performing an analysis.

Reduction transformations have also been proposed which completely reduce certain classes of Petri nets. Two of these proposals are investigated in this thesis. First, a set of reduction transformations that reduce the class of live and bounded free choice Petri nets to a reduced, simpler net in which either there is no concurrency or there is no choice, or possibly neither of these properties is investigated [Des90]. Secondly, sets of reduction rules that reduce the class of free choice nets for which there exists an initial marking under which the Petri net is live and bounded to an atomic Petri net, in which there is only one place and one transition are investigated [Esp91]. In each of these cases the ability for these sets of reduction transformations to reduce the Petri net to the simpler Petri net decides membership of the original Petri net in the class.

This thesis considers Petri net transformations as a mechanism for the analysis of Petri nets and emphasis is placed on those transformations that preserve liveness and boundedness. These are the properties that correspond to the absence of deadlocks and overflows, respectively, in the system being modelled by the Petri net and systems in which these properties hold are usually referred to as well-behaved.

1.1 Scope of Thesis

This thesis examines the Petri net transformations proposed in the literature, suggesting algorithms and the computational complexity regarding their applicability.

The problems dealt with in this thesis are the decision problems regarding the applicability of the proposed transformations. Also considered is the extent to which a reduction transformation can reduce the size of the coverability tree.

In order to show that certain problems concerning these Petri net transformations are intractable, this thesis proves that certain restrictions of the integer linear programming problem remains $\mathcal{NP}$-complete. The proofs of the complexity of deciding the applicability of certain Petri net transformations follows from this and polynomial-time reducibility.

Readers not familiar with the theory of Petri nets can find good tutorial introductions in [Mur89] and [Pet81]. The latter, although dated now, includes some interesting analysis problems in the Petri net area.

1.2 Thesis Outline

The remainder of this chapter outlines the rest of the thesis:

- **Chapter 2** is a review of the definitions and properties of Petri nets that are general and relevant throughout the thesis. Additional definitions, where required, are presented at the point of discussion. Also defined in Chapter 2 are subclasses of Petri nets. The purpose of these definitions is to formalise the notations and definitions used in this thesis.

- **Chapter 3** is a short review of the analysis techniques of Petri nets. This highlights the problems which the transformations are designed to tackle and so places the transformations in context.

- **Chapter 4** reviews computational complexity and presents some complexity results that are used in the proofs in later chapters. It presents a short collection of the results and definitions that are pertinent to this work. Included among these are the proofs of additional integer linear programming results due to this author.

- **Chapter 5** introduces and surveys all the relevant transformations described in the literature, including their examples. The transformations are divided into reduction transformations, synthesis transformations and transformations that can completely reduce certain classes of Petri nets.

- **Chapter 6**: For each of the transformations either an algorithm is sketched for the transformation rule showing that the decision problem is a polynomial-time problem or a proof of the intractability of the transformation rule is presented. Where appropriate, the reduction in size of the coverability tree that can be obtained is considered. Some of the authors have shown the complexity of their transformations.

- Finally, **Chapter 7** summarises the complexity results proved in this thesis and discusses the findings of the thesis. Also discussed, is possible future work in the area of Petri net transformations.

Chapter 2

Basic Definitions and Properties of Petri Nets

This chapter presents some of the basic definitions and properties of Petri nets required throughout the rest of the thesis independent from any of the Petri net transformations. Additional definitions, properties or results that may be needed as each of the transformations is considered will be included at the point of discussion. The notation presented here is quite common. Unless otherwise cited the definitions in this chapter are taken from [Mur89].

The properties of Petri nets can be split into two broad classes:

- **Behavioural properties.** These are those properties that depend on the initial marking (see Definition 2.1 below) and are thus also called *marking dependent* properties.
- **Structural properties.** These are those properties which are independent of the initial marking and hence only depend on the structure of the Petri net.

As far as possible, the notation used in this thesis is standard. The following points are included only to remove any possible ambiguity:

- $\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of all *natural numbers*.
- $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$ is the set of all *positive natural numbers*.
- $\mathbb{Z}$ is the set of all *integers*.
- $\mathbb{Q}$ is the set of all *rational numbers*.

- $\mathbb{Q}^+$ is the set of all *nonnegative rational numbers.*
- If X is a set, then $|X|$ denotes the *cardinality* of X.
- If X is a string over some alphabet Σ, then $|X|$ denotes the *length* of X in terms of the number of symbols in Σ.

The following section introduces Petri nets and gives a few basic definitions. This is then followed by a few definitions of behavioural properties and then a few structural properties. Finally, the chapter closes with a discussion of some classes of Petri nets.

2.1 Basic Definitions

A Petri net is a directed bipartite graph with arc weights. One set of the nodes are referred to as places and the other set of nodes are referred to as transitions. Places can be marked with tokens and it is sufficient to describe the initial distribution of tokens among the places when describing a system modelled by a Petri net. The following definitions provide a formal description of a Petri net and its evolution.

Definition 2.1 Petri Net
An unmarked Petri net *is a 4-tuple* $N = (P, T, B, F)$ *where:*

1. $P = \{p_1, \ldots, p_m\}$ *is a finite and non-empty set of* places. *Places are drawn as circles in the graphic representation of the Petri net.*
2. $T = \{t_1, \ldots, t_n\}$ *is a finite and non-empty set of* transitions. *Transitions are drawn as bars or rectangles in the graphic representation of the Petri net.*
3. $B \in \mathbb{N}^{m \times n}$ *is the* backward incidence matrix *and describes the arcs and their respective weights from the places to the transitions.*
4. $F \in \mathbb{N}^{m \times n}$ *is the* forward incidence matrix *and describes the arcs and their weights from the transitions to the places.*

Further, $P \cap T = \emptyset$. *The* incidence matrix C *of the Petri net* N *is defined by* $C = F - B$.

Definition 2.2 Markings, Marked Petri nets and Covering
A marking M *of a* marked Petri net $N = (P, T, B, F, M_0)$ *is a vector* $M \in \mathbb{N}^m$ *giving a distribution of tokens among the places in* P. M_0 *is the* initial marking *and is the marking of the Petri net in its initial state. A marking* M' *is said to* cover *a marking* M'' *if and only if* $M' \geq M''$, *that is, if and only if* $M'(p) \geq M''(p)\ \forall p \in P$.

This matrix notation of Petri nets will be used throughout this thesis but is equivalent to the functional notation often used in the literature. Furthermore, when referring to an element of the backward or forward incidence matrices or an element of a marking, a functional notation will be used; for example, $B(p,t)$ describes the arc from the place $p \in P$ to the transition $t \in T$ and its arc weight.

Definition 2.3 Enabled Transitions and Transition Firings
Given Petri net $N = (P, T, B, F)$, a transition $t_i \in T$ is said to be enabled *at a marking M' of N if and only if $M' \geq Be_i$ where e_i is the unit vector with a 1 as the i-th component. A transition $t_i \in T$ enabled at a marking M of N may* fire *yielding a new marking M'' of N defined by $M'' = M' + (F - B)e_i = M' + Ce_i$.*

Figure 2.1 shows an example of a marked Petri net $N = (P, T, B, F, M_0)$ with $P = \{p_1, p_2, p_3\}$, $T = \{t_1, t_2, t_3, t_4\}$,

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \qquad F = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

and $M_0^T = [1, 0, 1]$.

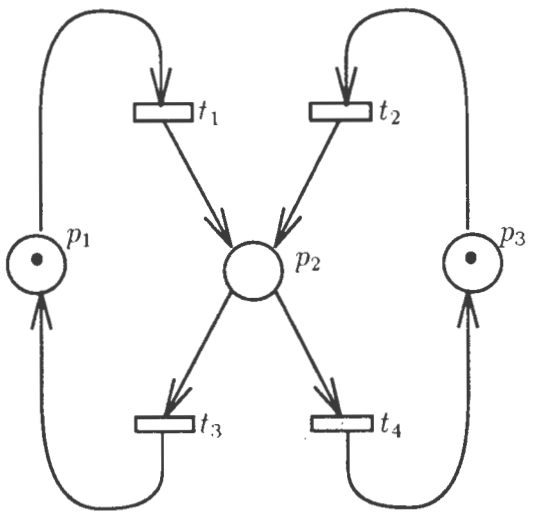


Figure 2.1: Example of a marked Petri net

Definition 2.4 Reachability
The firing of a transition usually changes the marking of a Petri net. If M_1 was the marking before the firing of a transition $t \in T$ and if M_2 the marking after firing t then

M_2 is said to be directly reachable *from M_1 by firing t and is written $M_1 \xrightarrow{t} M_2$. The* reachability set *is the transitive and reflexive closure of direct reachability and is written $R(M_0)$. $R(M)$ is the set of all possible markings reachable from the marking M.*

Equations (2.1) show the reachability set for the marked Petri net in Figure 2.1.

$$\begin{aligned} M_0^T &= [1,0,1] \\ M_1^T &= [0,1,1] \\ M_2^T &= [0,2,0] \\ M_3^T &= [1,1,0] \\ M_4^T &= [2,0,0] \\ M_5^T &= [0,0,2] \end{aligned} \tag{2.1}$$

Definition 2.5 Firing Sequence and Transition Sequence
Let $N = (P, T, B, F, M_0)$ be a Petri net with an initial marking, a sequence

$$\sigma = M_0 t_{i_1} M_1 t_{i_2} M_2 \ldots t_{i_k} M_k$$

of transitions $t_{i_1}, t_{i_2}, \ldots, t_{i_k} \in T$ and markings $M_0, M_1, M_2, \ldots, M_k \in R(M_0)$ is called a firing sequence *or* occurrence sequence *if and only if*

$$M_{j-1} \xrightarrow{t_{i_j}} M_j \text{ for } j = 1, 2, \ldots, k$$

this is also written $M_0 \xrightarrow{\sigma} M_k$. The firing sequence is often shown without any markings and in this case it is called a transition sequence *and is written $\sigma = t_{i_1} t_{i_2} \ldots t_{i_k}$.*

If $\sigma = t_1 t_2 \ldots t_k$ is a firing sequence then $\#(\sigma, t_i)$ is a count of the number of times the transition t_i appears in the sequence σ.

Definition 2.6 Language of a Petri Net
Given a Petri net $N = (P, T, B, F, M_0)$, the set of all possible firing sequences from M_0 is denoted by $L(N)$ and is called the language *of the Petri net N.*

Definition 2.7 Pre-set and Post-set
For an element $u \in P \cup T$ the set $\bullet u$ is called the pre-set *of u and $u\bullet$ is called the* post-set *of u and are defined as follows:*

1. *$\bullet t = \{p \in P : B(p,t) \neq 0\}$ and is the set of all input places of the transition $t \in T$.*
2. *$t\bullet = \{p \in P : F(p,t) \neq 0\}$ and is the set of output places of the transition $t \in T$.*

3. $\bullet p = \{t \in T : F(p,t) \neq 0\}$ *and is the set of all input transitions of the place* $p \in P$.

4. $p\bullet = \{t \in T : B(p,t) \neq 0\}$ *and is the set of all output transitions of the place* $p \in P$.

For a subset $U \subseteq P \cup T$, *the* pre-set *of* U, $\bullet U$ *and post-set of* U, $U\bullet$ *are defined as:*

$$\bullet U = \bigcup_{u \in U} \bullet u \text{ and}$$
$$U\bullet = \bigcup_{u \in U} u \bullet .$$

Definition 2.8 Siphons and Traps [BT87]
Let $N = (P, T, B, F)$ *be a Petri net. A nonempty subset* $S \subseteq P$ *is called a* siphon *if and only if* $\bullet S \subseteq S\bullet$. *A nonempty subset* $Q \subseteq P$ *is called a* trap *if and only if* $Q\bullet \subseteq \bullet Q$.

A siphon's behaviour is such that as the Petri net evolves (under firing of transitions) and the siphon is emptied of all tokens then it will remain empty for any further evolution of the Petri net. A trap's behaviour is just the opposite; as the Petri net evolves and the trap becomes marked with a token then it will be marked for any further evolution of the Petri net. Siphons are often called deadlocks in the literature, however [Mur89] prefers the term siphon as deadlocks can be confused with its more usual meaning of a circular wait condition.

Definition 2.9 Isolated Places and Isolated Transitions
A Petri net $N = (P, T, B, F)$ *is said to have an* isolated place $p \in P$ *if and only if* $\bullet p = \emptyset$ *and* $p\bullet = \emptyset$. *Similarly, the Petri net* N *is said to have an* isolated transition $t \in T$ *if and only if* $\bullet t = \emptyset$ *and* $t\bullet = \emptyset$.

The notions of *connected*, *strongly connected*, *(simple) paths* and *cycles*, etc. as defined for directed graphs in, for example, [Eve79] are required in this thesis.

This thesis assumes that Petri nets do not have isolated places or isolated transitions as an isolated place can never change its marking and an isolated transition can never change the marking on any place. Furthermore, it is possible to consider only connected Petri nets in the analysis of Petri nets, as disconnected components can be analysed separately as connected components.

Petri nets are suitable for modelling systems in which parallelism and synchronisation take place. These properties are embodied in the notions of *conflict* and *concurrency*. For a discussion of these notions and the applicability of Petri nets to the modelling of distributed systems with synchronisation, see [Pet81].

Definition 2.10 Conflict [Pet81]

Given a Petri net $N = (P, T, B, F, M_0)$*, if more than one transition is enabled at a marking* $M \in R(M_0)$ *and the firing of one of the enabled transitions, for instance* $t_i \in T$*, causes one of the other enabled transitions* $t_j \in T$ *to be disabled, then the two transitions,* $t_i, t_j \in T$ *are said to be in* conflict.

Definition 2.11 Concurrency [Pet81]

Given a Petri net $N = (P, T, B, F, M_0)$*, if two transitions* $t_i, t_j \in T$ *enabled at a marking* $M \in R(M_0)$ *do not affect each other in any way, and the possible firing sequences from* M *include some in which* t_i *occurs first and some in which* t_j *occurs first, then the Petri net* N *is said to display* concurrency.

Definition 2.12 Subnets and Generated Subnets [GSW80]

Let $N = (P, T, B, F)$ *be a Petri net. A net* $N' = (P', T', B', F')$ *is called a* subnet *of* N *if and only if* $P' \subseteq P$, $T' \subseteq T$ *and*

$$F'(p,t) = \begin{cases} F(p,t) & \text{if } p \in P' \text{ and } t \in T', \text{ and} \\ 0 & \text{otherwise, and} \end{cases}$$

$$B'(p,t) = \begin{cases} B(p,t) & \text{if } p \in P' \text{ and } t \in T', \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

The subnet generated *by the* $A \subseteq P$ *is the Petri net* $N' = (A, T', B', F')$ *where*

$$T' = \bigcup_{p \in A} {}^\bullet p \cup p^\bullet,$$

$$F'(p,t) = \begin{cases} F(p,t) & \text{if } p \in A \text{ and } t \in T', \text{ and} \\ 0 & \text{otherwise, and} \end{cases}$$

$$B'(p,t) = \begin{cases} B(p,t) & \text{if } p \in A \text{ and } t \in T', \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, the subnet *generated by* $A \subseteq T$ *is the Petri net* $N' = (P', A, B', F')$ *where*

$$P' = \bigcup_{t \in A} {}^\bullet t \cup t^\bullet,$$

$$F'(p,t) = \begin{cases} F(p,t) & \text{if } p \in P' \text{ and } t \in A, \text{ and} \\ 0 & \text{otherwise, and} \end{cases}$$

$$B'(p,t) = \begin{cases} B(p,t) & \text{if } p \in P' \text{ and } t \in A, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

2.2 Behavioural Properties

This section introduces a few definitions of properties that are *dependent* on the initial marking, such properties are termed *behavioural properties* as they pertain to the behavioural or dynamic evolution of the Petri net.

Definition 2.13 Bounded
A Petri net $N = (P, T, B, F, M_0)$ *is said to be* k-bounded, $k \in \mathbb{N}$, *or simply* bounded *if and only if* $\forall M \in R(M_0)$, $M(p) \leq k$ $\forall p \in P$ *where* $M(p)$ *is the number of tokens in the place* $p \in P$ *at marking* $M \in R(M_0)$.

Definition 2.14 Safe
A Petri net $N = (P, T, B, F, M_0)$ *is said to be* safe *if and only if it is 1-bounded.*

Definition 2.15 Live [Rei85]
In a Petri net $N = (P, T, B, F, M_0)$, *a transition* $t \in T$ *is said to be* live *if and only if* $\forall M \in R(M_0), \exists M'$ *reachable from* M *such that* M' *enables* t. *The Petri net* N *is said to be* live *if and only if* $\forall t \in T$, t *is live.*

Definition 2.16 Proper Termination [Ber86a]
A Petri net $N = (P, T, B, F, M_0)$ *is said to* terminate properly *if and only if* N *is bounded and a given final marking* $M \in R(M_0)$, *in which no transition is enabled, can be reached.*

Definition 2.17 Home State
For a Petri net $N = (P, T, B, F, M_0)$ *a marking* $M' \in R(M_0)$ *is said to be a* home state *if and only if* $\forall M \in R(M_0)$, M' *is reachable from* M.

Definition 2.18 Unavoidable State [Ber86a]
For a Petri net $N = (P, T, B, F, M_0)$, *a marking* $M \in R(M_0)$ *which is a home state is said to be an* unavoidable state *if and only if all firing sequences from* M_0 *lead to* M.

Definition 2.19 Pseudo-live [Ber86a]
A Petri net $N = (P, T, B, F, M_0)$ *is said to be* pseudo-live *if and only if* $\forall M \in R(M_0), \exists t \in T$ *such that* t *is enabled.*

Definition 2.20 Quasi-live [Ber86a]
A Petri net $N = (P, T, B, F, M_0)$ *is said to be* quasi-live *if and only if* $\forall t \in T, \exists M \in R(M_0)$ *such that* t *is enabled.*

This thesis concentrates on the *well behaved* properties, that is, the properties of liveness and boundedness, and on well behaved Petri nets, that is, those that are both live and bounded.

2.3 Structural Properties

This section introduces a few definitions of properties that are *independent* of the initial marking, such properties are termed structural properties as they describe the Petri net independent of a particular initial marking.

Definition 2.21 Structurally Bounded [ES90]
An unmarked Petri net $N = (P, T, B, F)$ is said to be structurally bounded *if and only if for all possible initial markings M_0' the Petri net $N' = (P, T, B, F, M_0')$ is bounded.*

Definition 2.22 Structurally Live [ES90]
An unmarked Petri net $N = (P, T, B, F)$ is said to be structurally live *if and only if there exists an initial marking M_0' such that the Petri net $N' = (P, T, B, F, M_0')$ is live.*

2.4 Classes of Petri nets

Some of the results and transformations mentioned in this thesis pertain to subclasses of Petri nets. The following classes of Petri net feature quite prominently in the literature on transformations of Petri nets.

Definition 2.23 Ordinary Petri Net
A Petri net $N = (P, T, B, F)$ is called an ordinary *Petri net if and only if*

$$B(p,t) \in \{0,1\} \quad \forall p \in P \text{ and } \forall t \in T \text{ and}$$
$$F(p,t) \in \{0,1\} \quad \forall p \in P \text{ and } \forall t \in T.$$

The term "ordinary" is sometimes used to refer to Petri nets as defined in Definition 2.1 to distinguish them from Coloured Petri nets (CPN), Stochastic Petri nets (SPN) or Generalised Stochastic Petri nets (GSPN). However, this should not cause any confusion as these extensions to Petri nets are not discussed in this thesis.

Definition 2.24 State Machine
A state machine *is an ordinary Petri net $N = (P, T, B, F)$ such that $|\bullet t| = 1$ and $|t \bullet| = 1$ for all $t \in T$.*

This definition of a state machine is used in [Mur89], [Pet81] and [Rei85]. However, there is another often used definition; [BT87], who proves several liveness and safeness results

for the classes of Petri nets in this section, only requires that $|\bullet t| \leq 1$ and $|t \bullet| \leq 1$ for all $t \in T$. [BT87] call such a Petri net an S-net and the Petri net defined in Definition 2.24 an S-graph. State machines are thus also called *P-graphs* or *S-graphs*. Definition 2.24 is preferred in this thesis as it is used in this form in some of the Petri net transformation literature (see, for example, [Des90]). The definition of state machine (above) is also more intuitive as a finite state machine never has a "half" arc leaving a state or a "half" arc entering a state.

Definition 2.25 Marked Graph
A marked graph *is an ordinary Petri net* $N = (P, T, B, F)$ *such that* $|\bullet p| = 1$ *and* $|p \bullet| = 1$ *for all* $p \in P$.

As with state machines, there is also another often used alternate definition of a marked graph; [Mur89], [Pet81] and [Rei85] use the definition above, while [BT87] only requires that $|\bullet p| \leq 1$ and $|p \bullet| \leq 1$ for all $p \in P$. [BT87] defines T-nets and T-graphs analogously to S-nets and S-graphs, respectively. Definition 2.25 of a marked graph is preferred in this thesis for the same reasons that Definition 2.24 is preferred (unless stated otherwise). [Rei85] also requires that marked graphs are strongly connected. Marked graphs are also called *event graphs*, *T-graphs* or *decision free nets* (because there is no conflict of enabled transitions).

Definition 2.26 Free Choice Net
A free choice net *is an ordinary Petri net* $N = (P, T, B, F)$ *such that* $|p \bullet| \leq 1$ *or* $\bullet(p\bullet) = \{p\}$ *for all* $p \in P$.

Other classes of Petri nets that appear in the Petri net literature, such as *extended free choice nets* and *asymmetric choice nets* or *simple nets*, have not featured in the Petri net transformation literature and so are omitted from this discussion.

Chapter 3

Analysis Techniques

Petri nets are a powerful medium for describing the behaviour of systems. Some of the important advantages of Petri nets over other modelling techniques are:

- Graphical representation of the system being modelled.
- There is often a direct translation between the real world system being modelled and the structure in the Petri net model.
- The model can be as abstract or as detailed as required by the modeller.
- Analysis techniques have been devised to detect certain properties in the system being modelled (assuming the model is a correct representation of the system) or in the model itself assuming that the system being modelled is *well behaved.*

In this chapter the analysis techniques that have been developed for Petri nets are discussed. In summarising the analysis techniques and their results, this chapter introduces the problem that has been studied in this thesis: the complexity of Petri net transformations as they apply to the simplification of the analysis of Petri nets.

Two analysis techniques have been developed for Petri nets: reachability and coverability trees, and matrix equations. These techniques are presented in this chapter together with a discussion of their problems. In addition to these techniques, a number of results have been developed which characterize liveness and boundedness in certain classes of Petri nets; these results are also summarised in this chapter. Omitted proofs can be found in the references cited.

3.1 Reachability and Coverability Trees

The reachability set $R(M_0)$ of Petri net $N = (P, T, B, F, M_0)$ can be drawn as a directed graph with root M_0. The nodes of the graph are thus the markings in the reachability set and a directed edge connects those markings that are directly reachable from one another. For example, a directed edge exists in the reachability tree from node M_i to node M_j labeled t_k if $M_i \xrightarrow{t_k} M_j$.

Figure 3.1 depicts the reachability graph of the marked Petri net in Figure 2.1 on page 7.

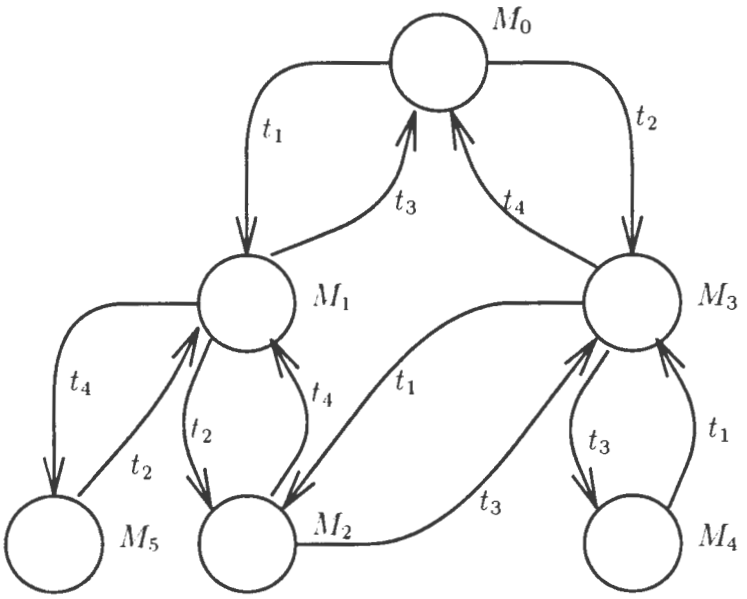


Figure 3.1: Reachability graph of Petri net

By removing back edges from the reachability graph and duplicating the nodes at the ends of these back edges as *duplicate nodes*, the reachability graph is transformed into a tree called the *reachability tree*. Nodes in which no transitions are enabled are called *terminal nodes*. Nodes which are neither duplicate nodes nor terminal nodes are termed *interior nodes*.

The reachability tree is generated by a procedure that is known as the *token playing game*: The initial marking is added as a node. For each of the transitions that are enabled at the initial marking a new marking is generated by simulating the firing of the corresponding transition. This new node and arc labeled with the transition that fired to cause the new marking, are added to the reachability tree. From this new marking, all enabled transitions are again considered, adding further nodes and arcs to the reachability tree. The procedure terminates by realising that certain nodes are duplicates and that it is not

necessary to further investigate the firing of enabled transitions from a duplicate marking.

The analysis of Petri nets using the reachability tree is performed by a walk of the tree. However, in the case of unbounded Petri nets the reachability tree is not finite and so this analysis technique cannot be used for unbounded Petri nets. To overcome this problem a finite representation of the reachability tree, called the *coverability tree*, is used.

The coverability tree is built in the same way as the reachability tree except that, in addition to detecting duplicate markings, the procedure detects cycles in the reachability tree that generate strictly covering markings which are not just duplicate markings. The places corresponding to these covered components of the new marking are unbounded. To represent an unbounded component in a marking a new symbol ω is used. In order to further generate the coverability tree it is necessary to define arithmetic rules for ω: $\forall k \in \mathbb{N}$,

$$\begin{aligned} \omega + k &= \omega \\ \omega - k &= \omega \\ k &< \omega \\ \omega &\leq \omega \end{aligned}$$

The algorithm for generating the coverability tree is described in [Pet81], [Mur89] and [BK93].

To illustrate the problem of using the coverability tree for Petri net analysis, consider the problem of deciding boundedness and liveness of a given a marked Petri net $N = (P, T, B, F, M_0)$. [Pet81] describes the work of Lipton who constructed a Petri net which could count to 2^{2^n} and then states that the existence of such a Petri net shows that the reachability problem requires at least exponential time and space. Further, [Pet81] states as an important corollary to this construction: "The Petri net which is constructed is bounded, since the number of tokens in any place cannot exceed 2^{2^n}. This means that any algorithm to determine boundedness of a Petri net must also require exponential time and space." The algorithm to generate the coverability tree, [Pet81] and [Mur89], must have at least an exponential worst case complexity. This complexity is in terms of the number of places and their interconnections to transitions in the case of Lipton. Also, [Rei85] mentions that it is decidable whether a Petri net is bounded or not with a space complexity of $2^{cs \log s}$, where s in this case is the sum $|P| + |T| + \sum_{p \in P} M_0(p)$ and c is some constant.

Thus without directly determining the space and time complexities of the algorithm to

generate the coverability tree it must at least have a worst case complexity of

$$2^{cS} \text{ where } S = |P| + |T| + \sum_{p \in P} \lceil \log_2 M_0(p) \rceil.$$

Generating the coverability tree decides whether the Petri net is bounded; if the symbol ω was used in the coverability tree then the Petri net is unbounded; otherwise the Petri net is bounded. Thus the space and time complexities to determine boundedness of the Petri net is at least 2^{cS}.

If the Petri net is not bounded then the coverability tree cannot be used to determine whether or not the Petri net is live as the ω symbol is a loss of information. For example, in a reachability problem in which it has to be decided whether a particular marking with an odd number of tokens in place p_0 is reachable and the only marking that covered this place had an ω symbol in this place then it would not be possible to determine whether this marking was reachable from the coverability tree as such information is lost by introducing the ω markings. A similar problem exists for liveness; [Pet81] gives an example of two Petri nets with the same coverability tree one of which can deadlock while the other cannot.

After deciding that the Petri net is bounded, the coverability tree, which now coincides with the reachability tree, can be used to test for liveness; [Baa88] describes an algorithm that splits a digraph into strongly connected components. The input to the algorithm is a graph $G = (V, E)$ (represented by linked adjacency lists) and the output is a list of the vertices in the strongly connected components. This algorithm has a time complexity of $O(\max(|V|, |E|))$ and a space complexity of $O(|V| + |E|)$ thus in terms of the original problem the time and space complexities are still 2^{cS}.

The problem of deciding whether the Petri net is live now reduces to determining whether or not each transition of T appears in all the strongly connected components as labels of arcs. At worst all the arcs of the coverability/reachability tree will have to be examined once with time and space complexity $O(|E|)$. Therefore in terms of the Petri net the complexities in space and time are still exponential. This intractably large coverability tree and hence reachability set is referred to as the *state space explosion problem.*

3.2 Matrix Equations

Given a Petri net $N = (P, T, B, F, M_0)$ in which a transition $t_i \in T$ is enabled at the marking $M' \in R(M_0)$ and $M' \xrightarrow{t_i} M'' \in R(M_0)$ then M'' is defined by

$$M'' = M' + Ce_i.$$

Ce_i is the column vector of C corresponding to the transition $t_i \in T$. Similarly for a transition t_j enabled at the new marking M'':

$$\begin{aligned} M''' &= M'' - Ce_j \\ &= M' + Ce_i + Ce_j \\ &= M' + C(e_i + e_j) \end{aligned}$$

Assume that $\sigma = t_{i_1} t_{i_2} \ldots t_{i_k}$ and that $M_0 \xrightarrow{\sigma} M_k$ then

$$\begin{aligned} M_k &= M_0 + C(e_{i_1} + e_{i_2} + \cdots + e_{i_k}) \\ M_k &= M_0 + Cf \end{aligned} \tag{3.1}$$

where $f = e_{i_1} + e_{i_2} + \cdots + e_{i_k}$ is called the *firing count vector* and whose components give the number of times each of the transitions in T fired in the sequence σ, that is,

$$f = \begin{bmatrix} \#(\sigma, t_1) \\ \#(\sigma, t_2) \\ \vdots \\ \#(\sigma, t_n) \end{bmatrix}.$$

Equation (3.1) is referred to as the *state equation* and can be used to verify certain invariant properties of all reachable markings; multiplying Equation (3.1) on the left by $v \in \mathbb{N}^m$ yields

$$v^T M_k = v^T M_0 + v^T C f. \tag{3.2}$$

If $v \in \mathbb{N}^m$, $v \neq O$, $f \neq O$ then $M_k = M_0$ implies that $v^T C = O$:

Definition 3.1 Place Invariants [Mur89]
Given a Petri net $N = (P, T, B, F)$. A vector $v \in \mathbb{N}^m$, $v \neq O$ such that $v^T C = O$ is called a place invariant *or simply a* P-invariant. *If v is a P-invariant then $\{p \in P : v(p) \neq 0\}$ is called the* support *of v. A support is* minimal *if no proper subset of it is also a support. If each element of v is nonnegative then v is termed a* positive P-invariant. *The Petri net N is said to be* covered *by positive P-invariants if and only if $\forall p_i \in P$ there exists a positive P-invariant v such that $v(p_i) \neq 0$.*

Definition 3.2 P-components [GSW80]
The subnet generated by the support of a P-invariant is called a P-component.

P-invariants give a sufficient condition for boundedness of Petri nets:

Theorem 3.1 [Rei85] *Let $N = (P, T, B, F, M_0)$ be a Petri net. If N is covered by positive P-invariants then N is bounded.*

From Equation (3.1) it can also be seen that if $M_k = M_0$ then $Cf = O$. In other words, a firing count vector that leaves the marking invariant:

Definition 3.3 Transition Invariant [Mur89]
Given a Petri net $N = (P, T, B, F)$, a vector $f \in \mathbb{N}^n$, $f \neq O$ such that $Cf = O$ is called a transition invariant *or simply a* T-invariant. *If f is a T-invariant then $\{t \in T : f(t) \neq 0\}$ is called the* support *of f.* Minimal supports *and* positive T-invariants *are also defined for T-invariants analogously to their P-invariant counterparts.*

Boundedness and liveness give a sufficient condition for the existence of positive T-invariants:

Theorem 3.2 [Rei85] *Let $N = (P, T, B, F, M_0)$ be a Petri net. If N is bounded and live then N is covered by positive T-invariants.*

Invariant analysis does not provide a method of deciding liveness in Petri nets. Furthermore, it does not give necessary conditions for boundedness in Petri nets; a Petri net may be bounded yet not covered by positive P-invariants.

Although it is not possible to characterize structural liveness for a general Petri net (see [Mur89]), structural boundedness can be characterized using matrices:

Theorem 3.3 [Mur89] *Let $N = (P, T, B, F)$ be a Petri net. N is structurally bounded if and only if there exists $v \in \mathbb{N}^{+m}$ such that $v^T C \leq O$.*

3.3 Classes of Petri Nets

Analysis techniques have also been established for classes of Petri nets which do not require the generation of the coverability tree. This section presents some of these results for the classes of Petri nets defined in Chapter 2.

Without generating the coverability tree or resorting to invariant analysis it is possible to decide whether a state machine is live and for a live state machine it is possible to decide whether it is bounded:

Theorem 3.4 [BT87] *Let $N = (P, T, B, F, M_0)$ be a state machine. N is live if and only if N is strongly connected and $M_0 \neq O$.*

Theorem 3.5 *Let $N = (P, T, B, F, M_0)$ be a state machine. N is bounded.*

Proof Since $|\bullet t| = 1$ and $|t \bullet| = 1$ for all $t \in T$, tokens can neither be created nor destroyed as transitions fire. □

Under the alternate definition of a state machine mentioned in Chapter 2, a state machine is bounded if it is strongly connected. This strongly connected condition just ensures that $|\bullet t| = 1$ and $|t \bullet| = 1$ for all $t \in T$ (assuming that state machines are connected).

Furthermore, for marked graphs, the following results characterize liveness and boundedness:

Theorem 3.6 [Mur89] *Let $N = (P, T, B, F, M_0)$ be a marked graph. N is live if and only if every simple cycle of N contains a place that is marked at M_0.*

Lemma 3.1 [Rei85] *Let $N = (P, T, B, F, M_0)$ be a marked graph and let $w = (p_0, p_1, \ldots, p_n)$ be the places of a cycle of N. Then for all markings $M \in R(M_0)$*

$$\sum_{i=0}^{n} M(p_i) = \sum_{i=0}^{n} M_0(p_i).$$

Theorem 3.7 *Let $N = (P, T, B, F, M_0)$ be a live marked graph. N is bounded if and only if N is covered by simple cycles, that is, for all $u \in P \cup T$ there exists a simple cycle containing u.*

Proof If N is covered by cycles then, by Lemma 3.1, N is bounded. If N is not covered by cycles then there must be an edge of the marked graph $t_i \rightarrow p_i \rightarrow t_{i+1}$ such that either t_i does not have any input places or t_{i+1} does not have any output places. If t_i has no input places then t_i can fire an arbitrary number of times in succession, thus p_i is not bounded, and so N is not bounded. If t_{i+1} has no output places then it can always empty p_i of any tokens placed on it. If the number of tokens that can be placed on p_i is finite then the t_{i+1} can remove them all and the marked graph is not live. If there is no limit to then number of tokens that can be placed on p_i then p_i cannot be bounded as t_{i+1} does not have to fire, so the marked graph is not bounded. □

Finally, for free choice nets, liveness is characterized by:

Theorem 3.8 [BT87] *Let $N = (P, T, B, F, M_0)$ be a free choice net. N is live if and only if every siphon of N contains a trap which is marked at M_0.*

In order to characterize boundedness in live free choice systems, the following definition and results are used:

Definition 3.4 Extended Free Choice Net [Mur89]
An extended free choice net $N = (P, T, B, F)$ *is an ordinary Petri net in which*

$$(p_1\bullet) \cap (p_2\bullet) \neq \emptyset \Longrightarrow p_1\bullet = p_2\bullet \text{ for all } p_1, p_2 \in P.$$

Theorem 3.9 [BD90] *Let $N = (P, T, B, F, M_0)$ be a live extended free choice net. N is k-bounded if and only if N is covered by strongly connected P-components which have no more than k tokens at M_0.*

Theorem 3.10 *Let $N = (P, T, B, F, M_0)$ be a live free choice net. N is bounded if and only if N is covered by strongly connected P-components.*

Proof Let $N = (P, T, B, F, M_0)$ be a live and bounded free choice net. N is an extended free choice net (extended free choice nets are a superclass of free choice nets), so N must be covered by P-components by Theorem 3.9. Now let $N = (P, T, B, F, M_0)$ be a live and unbounded free choice net. By Theorem 3.9 N is not covered by P-components since M_0 is always assumed to be finite. □

[BT87] also give results that characterise safeness in these classes of Petri nets.

Chapter 4

Computational Complexity

In discussing the computational complexity of Petri net transformations in Chapter 6, reference is made to certain notions of complexity and results. This chapter defines those notions of complexity that will be used later in this thesis together with some of the results that are required to classify certain problems according to their complexity. This chapter also motivates the use of Pascal as a reasonable computation model, it describes the standard encodings that will be used throughout this thesis to encode problem instances, and the data structures used to encode Petri nets. The discussion below assumes familiarity with some notions of formal languages from sources such as [HU79] or [HU69a].

4.1 Models of Computation

This thesis endeavours to give bounds on the worst case complexities of the decision problems on the applicability of certain Petri net transformations. This means that the problems can be considered from a language theoretic point of view; the applicability of a transformation corresponds to *deciding* whether a *word* over some *alphabet* describing an *instance* of the problem by a *(deterministic multitape) Turing machine* belongs to a particular language. That is, for all possible instances of the problem some Turing machine halts with an output symbol on the tape of *"yes"* (*accepting* the word) if the transformation is applicable and *"no"* if the transformation is not applicable. Such a Turing machine is termed an *algorithm* or *effective procedure* [HU79]. Under *Church's hypothesis* this formal definition of an algorithm should not conflict with the intuitive notion of an algorithm. However, from a programming point of view, the Turing machine is equivalent in computing power to the modern digital computer [HU79].

The measure of time and space complexity of such problems is usually expressed in terms of a worst case upper bound as a function of the length of the problem instance on the

tape presented to the Turing machine. If this length is n then $T(n)$ and $S(n)$ represent the worst case time and space complexities, respectively, for the decision problems; $T(n)$ is the maximum number of moves the Turing machine makes before entering a halt state and $S(n)$ is the maximum number of tape cells scanned before entering a halt state [HU79].

The Turing machine model is too clumsy to describe algorithms and in this thesis the programming language Pascal (level 0) is used to describe algorithms that are to be analysed [JW83]. That the Pascal model is *polynomially equivalent* to the Turing machine model (under logarithmic costs), in terms of the complexity measures $T(n)$ and $S(n)$, follows from the following facts:

- A Pascal compiler can produce code for a *random access machine* (RAM) or a *random access stored program machine* (RASP) in such a way that each Pascal statement produces a polynomially bounded (in the length of the statement) number of RAM or RASP instructions.
- The RAM and RASP models are polynomially related to the Turing machine model (under logarithmic cost) [AHU74].

In this thesis logarithmic costs are ignored and the complexity of algorithms is given in terms of the number of times the most significant operations are performed, usually the number of times loops are executed. The intent is to illustrate that the algorithm, and hence the problem, is polynomially bounded. Furthermore, procedures and functions will be assumed to be expanded in-line so that their complexity is also taken into account. [AHU74] use the above argument to motivate the use of a language called Pidgin Algol as a suitable model of computation equivalent to the Turing machine model in both power and complexity. The equivalence of Pidgin Algol and Pascal is obvious.

The complexity measures will be given in terms of their *asymptotic upper bounds* or O-notation (see, for example, [CLR90]). For the sequel, $T(n)$ and $S(n)$ of an algorithm will imply a high level programming language model of computation.

4.2 Reasonable Problem Encodings

An *abstract decision problem* is a binary relation mapping a set of problem *instances* onto the set of solutions {"yes", "no"}. Before an algorithm can be applied to an abstract problem, the problem needs to be *encoded*; the resulting problem instance is known as a *concrete decision problem.*

The complexity measures $T(n)$ and $S(n)$ above are measures of complexity of concrete problem instances as they are functions of the length of the problem instance on the tape. However, it is not the complexity of the *concrete* problem that is important, since it may depend on the particular encoding chosen and may not reflect the inherent complexity of the *abstract* problem. It is clear then that a suitable *standard encoding* is required. If an algorithm accepts a single number k as input then provided this number is not represented in *unary* then all other "reasonable" encodings (for example, binary, decimal or ASCII) are polynomially equivalent [CLR90].

In this thesis numbers are assumed to be represented in binary in the standard encoding; thus the number k requires $\lceil \log_2 k \rceil$ bits to represent in the standard encoding. Lists of numbers in the standard encoding will be encoded as the individual numbers separated by some delimiting tape symbol. The length of such a list of numbers will be taken to be the sum of the lengths of the individual numbers. Arrays will be encoded with their dimensions followed by their entries as lists of numbers. The space required for the dimensions can be ignored as this is never significant compared to the space required to encode the entries. Thus the length required to encode an array of numbers will be taken as the sum of the lengths of the entries of the array. The alphabet of the language that is used to describe problem instances is thus $\Sigma = \{0, 1\}$. If G is an abstract problem instance then $\langle G \rangle$ represents the concrete encoding of the problem in the standard encoding.

Of the popular representations of graphs and digraphs, *adjacency matrices* or *adjacency lists* (see, for example, [Baa88]), this thesis uses adjacency matrices to represent graphs for the following reasons:

- The incidence matrices can be thought of as adjacency matrices. In particular, for a Petri net $N = (P, T, B, F)$ the corresponding adjacency matrix, A, would be

$$A = \begin{bmatrix} O & B \\ F^T & O \end{bmatrix}.$$

- The conversion between the two representations can be performed in polynomial-time and are hence polynomially equivalent.

- The complexity of the sparse (few edges) and the dense case do not have to be separately given.

- The high level language model of computation chosen is only polynomially related to the Turing machine model (under logarithmic cost). In addition, the particular encoding chosen is one of several possible polynomially related suitable encodings

and so any further polynomial abstractions will not materially change the results (the set of polynomials are closed under function composition).

If $N = (P, T, B, F, M_0)$ is a marked Petri net describing an abstract problem then $\langle N \rangle = \langle B \rangle \langle F \rangle \langle M_0 \rangle$, that is, the encoding of the Petri net is the concatenation of the encodings of the individual matrices used to describe the Petri net.

4.3 Complexity Classes

In defining the complexity classes used in this thesis, it is necessary to define the smallest classes first, as the definitions of some of the larger classes use the notions of the smaller classes:

Definition 4.1 Complexity Class LOGSPACE [vEB90]
A language L is said to belong to the complexity class **LOGSPACE** *if and only if*

$$L = \{x \in \{0,1\}^* : \exists M : \{0,1\}^* \rightarrow \{\text{"yes"}\} : S_M(|x|) = O(k \log |x|)\},$$

where k is a constant and M is a Turing machine that recognizes strings in L (the class of languages are recursively enumerable and not necessarily recursive, that is, M need only accept *strings of a language belonging to the complexity class and not necessarily decide such strings). $*$ is the* Kleene closure *operator.*

Definition 4.2 Complexity Class $\mathcal{P}$ [CLR90]
A language L is said to belong to the complexity class $\mathcal{P}$ if and only if

$$L = \{x \subseteq \{0,1\}^* : \exists A : \{0,1\}^* \rightarrow \{\text{"yes"}, \text{"no"}\} : T_A(|x|) = O(|x|^c)\},$$

where A is an algorithm that decides strings in L and c is some constant. T_A is the time complexity measure of algorithm A.

The following result was used implicitly in Definition 4.2:

Theorem 4.1 [Bro89] *If a language L can be recognized by a Turing machine in polynomial-time, then there is a Turing machine that decides L in polynomial-time.*

The following additional definition is required in order to define the complexity class $\mathcal{NP}$.

Definition 4.3 Verification Algorithms and Certificates [CLR90]
A verification algorithm *is a two argument algorithm $A(x, y)$ where x is an input string and y is a binary string called a* certificate. *A* verifies *the input string x if there exists a certificate y such that $A(x, y) =$ "yes".*

Definition 4.4 Complexity Class $\mathcal{NP}$ [CLR90]
A language L is said to belong to the complexity class $\mathcal{NP}$ if and only if

$$L = \{x \subseteq \{0,1\}^* : \exists y : |y| = O(|x|^c) : A(x, y) = \text{"yes"}\},$$

where y is a certificate, A is a polynomial-time verification algorithm and c is some constant. A is said to verify *language L in polynomial-time.*

It is often useful to be able to say whether a particular problem is at least as hard as any problem in a particular class. The following notion of reduction is used in the definition of *hardness* for a complexity class:

Definition 4.5 Log-space and Polynomial-time Reductions
A problem X is said to be log-space reducible *(*polynomial-time reducible*) to a problem Y, written $X \preceq_{\mathbf{LOGSPACE}} Y$ ($X \preceq_{\mathcal{P}} Y$) if and only if there exists $f : \{0,1\}^* \to \{0,1\}^*$ computable in logarithmic bounded space (polynomially bounded time) mapping instances of X onto instances of Y such that if an algorithm $A : \{0,1\}^* \to \{$"yes", "no"$\}$ decides Y then $A \circ f$ decides X. If the complexity of the reduction f is such that for any problems X and Y, $X \preceq_f Y$, and $X \in \mathcal{C}$ implies $Y \in \mathcal{C}$, for some complexity class $\mathcal{C}$, then f is said to be* compatible *with $\mathcal{C}$.*

Definition 4.6 Hardness for a Class Under a Class of Reductions [Joh90]
Given a problem X and a complexity class $\mathcal{C}$. If $Y \preceq_{\mathbf{LOGSPACE}} X$ for all $Y \in \mathcal{C}$ then X is said to be $\mathcal{C}$-hard under log-space reductions. *Similarly, if $Y \preceq_{\mathcal{P}} X$ for all $Y \in \mathcal{C}$ then X is said to be* $\mathcal{C}$-hard under polynomial-time reductions.

This notion of hardness is used, in turn, in the definition of complete classes:

Definition 4.7 Complexity Class $\mathcal{NPC}$ [CLR90]
A language $L \subseteq \{0,1\}^*$ is said to be $\mathcal{NP}$-complete or $\mathcal{NPC}$ if and only if L satisfies the following two conditions:

1. *$L \in \mathcal{NP}$, and*

2. *L is $\mathcal{NP}$-hard ($\mathcal{NPH}$) under polynomial-time reductions.*

The function f is termed a reduction function *and the corresponding algorithm a* reduction algorithm.

A problem is defined as *tractable* if it is in $\mathcal{P}$ (see, for example, [GJ79]). Since it is not known whether $\mathcal{P} = \mathcal{NP}$ it is not possible to decide whether a problem in $\mathcal{NP}$ is tractable unless it can also be shown to be in $\mathcal{P}$ ($\mathcal{P} \subseteq \mathcal{NP}$). However, many researchers believe that $\mathcal{P} \neq \mathcal{NP}$. Strong circumstantial evidence of the intractability of the problem can be found in the fact that many problems, none of which have a known polynomial-time algorithm, have been shown to belong to $\mathcal{NPC}$. It is for this reason that the time complexity measure $T(n)$ is used in this thesis as a means of deciding the tractability of a problem.

Lipton showed that the reachability problem required at least exponential time and space [Pet81]. The following complexity classes, of exponential worst case complexities, are required to understand the complexity of the reachability problem.

Definition 4.8 Complexity Class EXPTIME [vEB90]
A language L is said to belong to the complexity class **EXPTIME** *if and only if*

$$L = \{x \in \{0,1\}^* : \exists M : \{0,1\}^* - \{\text{"yes"}\} : T_M(|x|) = O(k2^{|x|^c})\},$$

where k and c are constants and M is a Turing machine that recognizes strings in L.

Definition 4.9 Complexity Class EXPSPACE [vEB90]
A language L is said to belong to the complexity class **EXPSPACE** *if and only if*

$$L = \{x \in \{0,1\}^* : \exists M : \{0,1\}^* - \{\text{"yes"}\} : S_M(|x|) = O(k2^{|x|^c})\},$$

where k and c are constants and M is a Turing machine that recognizes strings in L.

Lipton's results actually show that the reachability problem is **EXPSPACE**-hard under log-space reductions [JLL77].

The complexity hierarchy for the complexity classes mentioned above, excluding hard and complete classes, is as follows [vEB90]:

$$\textbf{LOGSPACE} \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \textbf{EXPTIME} \subseteq \textbf{EXPSPACE}.$$

Under the assumption that $\mathcal{P} \neq \mathcal{NP}$, the containment of $\mathcal{P}$ in its superclasses is proper. Furthermore, since hardness of a problem with respect to a given complexity class gives

a lower bound on the worst case complexity (for example, a problem that is $\mathcal{NPH}$ is at least as hard as any problem in $\mathcal{NP}$), it follows that $\mathcal{P}$ and $\mathcal{NPH}$, or **EXPSPACE**-hard are disjoint.

The following notions and results concerning complementary languages and complementary complexity classes are useful in proving certain problems intractable:

Definition 4.10 Complement of a Language [Joh90]
If L is a language over some alphabet Σ, then its complementary language *is $\mathcal{C}o$-L = $\Sigma^* - L$.*

The complexity class $\mathcal{P}$ is closed under complement. This well known result follows from Theorem 4.1:

Theorem 4.2 *If L is a language in $\mathcal{P}$, then $\mathcal{C}o$-L is also in $\mathcal{P}$.*

In this thesis a number of problems are shown to be $\mathcal{C}o$-$\mathcal{NPH}$ and $\mathcal{C}o$-**EXPSPACE**-hard. This is sufficient to prove the intractability of such problems, since such a problem cannot be in $\mathcal{P}$ according to Theorem 4.2 and the fact that both $\mathcal{NPH}$ and **EXPSPACE**-hard are disjoint from $\mathcal{P}$.

4.4 Complexity of Certain Useful Problems

In order to prove the complexity bounds for certain problems it is often possible to find a polynomial-time reduction to the problem from some other problem of known complexity thus showing that the new problem is at least as complex as a known problem. Alternatively, it may be possible to show that two problems, one of known complexity and the other of unknown complexity are *polynomially equivalent* (polynomial-time transformations exist from either problem to the other) thus establishing that the problems have the same complexity. Also, for proving problems to be $\mathcal{NPH}$ it is sufficient to show a polynomial-time transformation from any $\mathcal{NPC}$ problem to the problem of interest [CLR90]. In all the cases above it would be useful to have a collection of problems of known complexity. This section introduces certain such useful problems of known complexity. Those theorems that include proofs are due to this author except where noted otherwise. Proofs of the theorems that are merely stated can be found in the references indicated.

4.4.1 Three Conjunctive Normal Form Satisfiability

This problem concerns the existence of a truth assignment to a boolean expression such that the boolean expression is satisfied:

Definition 4.11 Three Conjunctive Normal Form Satisfiability Problem [GJ79] *Let $X = \{x_1, \ldots, x_h\}$ be a set of boolean variables and $J = \bigwedge_{i=1}^{l}(\alpha_1^i \vee \alpha_2^i \vee \alpha_3^i)$ a boolean formula, where $\alpha_j^i = x_k$ or $\neg x_k, x_k \in X$. The problem of deciding whether an assignment to the variables in X exists, such that J is satisfied is called the* three conjunctive normal form satisfiability problem *or 3-CNF-SAT.*

Theorem 4.3 [GJ79] *Deciding whether there exists a truth assignment satisfying a given three conjunctive normal form boolean expression is $\mathcal{NPC}$.*

4.4.2 Integer Linear Inequalities and Equations

A number of problems can be formulated as systems of integer linear inequalities or as systems of linear equations that require integer, nonnegative integer or $\{0, 1\}$ solutions, depending on the particular problem. A number of such problems are dealt with in this thesis. The complexity of such problems is thus useful as it enables one to determine the complexity of other problems. The complexity of the general cases of these problems is well established. However, some problems can be formulated as restrictions of these problems; for example, the homogeneous cases of some of these problems are also used in this thesis. Cited below are some results concerning the complexity of these problems and, following that, proofs of the complexity of the restricted problems used in this thesis.

The *linear programming problem*, or LP, is defined as follows:

Definition 4.12 Linear Programming Problem
Given a matrix $A \in \mathbb{Q}^{m \times n}$ and a column vector $b \in \mathbb{Q}^m$ does there exist an $x \in \mathbb{Q}^n$ such that $Ax \leq b$?

The *integer linear programming problem*, or ILP, is defined as follows:

Definition 4.13 Integer Linear Programming Problem
Given a matrix $A \in \mathbb{Q}^{m \times n}$ and a column vector $b \in \mathbb{Q}^m$ does there exist an $x \in \mathbb{N}^n$ such that $Ax \leq b$? The $\{0, 1\}$ integer linear programming problem, *or ZERO-ONE ILP, is defined as for the ILP problem except that the solution vector x is required to belong to $\{0, 1\}^n$.*

The *homogeneous integer linear programming problem*, or HILP, is defined as follows:

Definition 4.14 Homogeneous Integer Linear Programming Problem
Given a matrix $A \in \mathbb{Q}^{m \times n}$ does there exist an $x \in \mathbb{N}^n$ such that $Ax \leq O$ and $x \neq O$? The $\{0,1\}$ homogeneous integer linear programming problem, *or ZERO-ONE HILP, is defined as in the HILP case except that the solution vector x is required to belong to $\{0,1\}^n$.*

In all cases the entries of A and b can be specified as pairs of numbers, a numerator and a denominator, both specified as binary numbers. The length of an instance of the above problems is taken to be the sum of the length of the entries of A and b (in the case of ILP). Note that all the entries of A and b can be multiplied by the product of all the denominators and the result will be a matrix and a vector with integer entries and this new problem instance will have a solution if and only if the original problem instance had a solution (both problems admit the same solutions). Furthermore, this transformation is a polynomial-time transformation and the increase in problem size is polynomially bounded (the size is at most squared). The sequel assumes that the entries of A and b are integers [vEB79].

Theorem 4.4 [NW88] *Deciding whether there exists a solution to a given LP problem is in $\mathcal{P}$.*

In the following sense, the LP problem is among the hardest problems in $\mathcal{P}$:

Theorem 4.5 [DLR79] *The LP problem is $\mathcal{P}$-hard under* **LOGSPACE** *reductions.*

Theorem 4.4 and Theorem 4.5 together show that the LP problem is $\mathcal{P}$-complete under **LOGSPACE** reductions.

Theorem 4.6 *Deciding whether a solution exists to a given ZERO-ONE ILP problem is $\mathcal{NPC}$.*

The following result shows that all problems of systems of linear equations and inequalities over $\mathbb{Z}$, $\mathbb{N}$ or $\{0,1\}$ are in $\mathcal{NP}$:

Theorem 4.7 [vzGS78] *Given $A' \in \mathbb{Z}^{m \times n}$, $b' \in \mathbb{Z}^m$, $C' \in \mathbb{Z}^{p \times n}$ and $d' \in \mathbb{Z}^p$, then the problem of deciding whether there exists an $x \in \mathbb{Z}^n$ (or $\{0,1\}^n$) such that $A'x = b'$ and $C'x \geq d'$ is in $\mathcal{NP}$.*

For example, the HILP problem can be seen to be $\mathcal{NP}$ by letting $A' = O$, $b' = O$, $d' = O$ and

$$C' = \begin{bmatrix} I_n \\ -A \end{bmatrix}.$$

The following proof is part of the proof of Theorem 4.8 from [HU79]:

Proof The problem is in $\mathcal{NP}$ by Theorem 4.7. To show the problem is $\mathcal{NPH}$: Assume an instance of 3-CNF-SAT and the notation from Definition 4.11. Construct an instance of ZERO-ONE ILP as follows: Corresponding to each of the clauses $\alpha_1^i \vee \alpha_2^i \vee \alpha_3^i$ $(i = 1, \ldots, l)$ of J, introduce the constraint

$$\alpha_1^i + \alpha_2^i + \alpha_3^i \geq 1.$$

This ensures that each clause evaluates to true. Also, to ensure that for each $x_i \in X$ when $x_i = 0$ then $\neg x_i = 1$ and vice versa, add the constraints

$$\begin{aligned} x_i + \neg x_i &\geq 1 \\ -x_i - \neg x_i &\geq -1 \end{aligned}$$

The resultant ZERO-ONE ILP problem has a solution if and only if J is satisfiable with some truth assignment. The problem is thus $\mathcal{NPH}$ and hence $\mathcal{NPC}$. □

Theorem 4.8 [HU79] *Deciding whether a solution exists to a given ILP problem is $\mathcal{NPC}$.*

The proof in this thesis that the ZERO-ONE HILP problem is $\mathcal{NPC}$ requires the complexity of a *restricted homogeneous* $\{0, 1\}$ *linear programming problem*, which is defined as follows:

Definition 4.15 The ZERO-ONE SHILP Problem
Given a matrix $A \in \mathbb{Z}^{m \times n}$ does there exist an $x \in \mathbb{N}^n$ such that $Ax \leq O$ and $x_1 = 1$ where x_1 is the first component of x?

Theorem 4.9 *Deciding whether a solution exists to a given ZERO-ONE SHILP problem is $\mathcal{NPC}$.*

Proof The problem is in $\mathcal{NP}$ by Theorem 4.7. To show that the problem is $\mathcal{NPH}$: Given an instance of the ZERO-ONE ILP problem, $Ax \leq b$, introduce a new scalar variable,

y. The given instance of the ZERO-ONE ILP problem has a solution if and only if the following instance of the ZERO-ONE SHILP problem

$$[-b|A]\begin{bmatrix} y \\ x \end{bmatrix} \leq O \text{ such that } y = 1$$

has a solution. The problem is thus NP-hard and this completes the proof that the problem is NP-complete. □

Theorem 4.10 *Deciding whether a solution to a given ZERO-ONE HILP problem exists is $\mathcal{NPC}$.*

Proof The problem is in $\mathcal{NP}$ by Theorem 4.7. To show that the problem is $\mathcal{NPH}$ a polynomial-time reduction from the problem proved to be $\mathcal{NPC}$ in Theorem 4.9 is used: Given an instance of ZERO-ONE SHILP, $Ax \leq O$, $x_1 = 1$ and $A \in \mathbb{Z}^{m \times n}$, construct an instance of the ZERO-ONE HILP problem, $A'x \leq O$, $A' \in \mathbb{Z}^{(m+n-1)\times n}$, by adding the constraints

$$\begin{aligned} -x_1 + x_2 &\leq 0 \\ -x_1 + x_3 &\leq 0 \\ &\vdots \\ -x_1 + x_n &\leq 0 \end{aligned}$$

to $Ax \leq O$. This ZERO-ONE HILP problem has a nontrivial solution $x \in \{0,1\}^n$ if and only if the given instance of ZERO-ONE SHILP has a solution, that is, with $x_1 = 1$ since $x_1 = 0 \Rightarrow x_2 = 0, x_3 = 0, \ldots, x_n = 0$ which is a trivial solution. Thus the ZERO-ONE HILP problem is $\mathcal{NPH}$. This completes the proof that the ZERO-ONE HILP problem is $\mathcal{NPC}$. □

Theorem 4.11 **[WW86]** *Deciding whether a solution exists to a given LP problem is in $\mathcal{P}$.*

Theorem 4.12 *Deciding whether a solution exists to a given HILP problem is in $\mathcal{P}$. If any component of a solution were constrained by a positive lower bound then the problem is still in $\mathcal{P}$.*

This simple proof was pointed out by David S. Johnson:

Proof Given an instance of HILP, $Ax \leq O$, the problem can be formulated as an LP problem by adding the constraints

$$\begin{aligned} x_1 + x_2 + \cdots + x_n &\geq 1 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \\ &\vdots \\ x_n &\geq 0. \end{aligned}$$

A solution to the HILP problem exists if and only if a solution to the constructed LP problem exists, because the homogeneity of the problem allows any rational solution to be scaled to an integer solution. Since the scaling involves multiplying by positive integers, any component satisfying a positive lower bound will satisfy the constraint after scaling. □

The complexity of the homogeneous system changes if the system is required to be bounded. The complexity result in this case follows from a polynomial-time reduction from a restricted knapsack problem which is also $\mathcal{NPC}$:

Definition 4.16 The Knapsack Problem [GJ79]
Given a vector $a \in \mathbb{N}^n$ and $B, K \in \mathbb{N}$, $B \geq K$, does there exist a vector $x \in \{0,1\}^n$ such that $a \cdot x \leq B$ and $a \cdot x \geq K$?

The following result cited in [GJ79]:

Theorem 4.13 *The knapsack problem in Definition 4.16 is $\mathcal{NPC}$.*

Definition 4.17 A Special Bounded Homogeneous Integer Linear Programming Problem (SBHILP)
Given a matrix $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{N}$ and a component index k, does there exist an $x \in \mathbb{N}^n$ such that $Ax \geq O$, $Ax \leq be$ (e is the vector with each component equal to one) and $x_k \geq 1$?

Theorem 4.14 *The SBHILP problem is $\mathcal{NPC}$.*

Proof The SBHILP problem is in $\mathcal{NP}$ by Theorem 4.7. The proof that the problem is $\mathcal{NPH}$ follows from a reduction from the knapsack problem: Consider an instance of the knapsack problem in Definition 4.16: $a \in \mathbb{N}^n$ and $B, K \in \mathbb{N}$. The system

$$\begin{aligned} a \cdot x &\leq B \\ a \cdot x &\geq K \end{aligned}$$

where $x \in \{0,1\}^n$ can be rewritten

$$\begin{aligned} a \cdot x - Ky &\leq B - K \\ a \cdot x - Ky &\geq 0 \end{aligned}$$

where $y = 1$ and $x \in \{0,1\}^n$. This system can be augmented with additional constraints to get an equivalent system with solutions $x \in \mathbb{N}^n$:

$$\begin{aligned} a \cdot x - Ky &\leq B - K \\ (B-K)x_i &\leq B - K \text{ for } i = 1,2,\ldots,n \\ (B-K)y &\leq B - K \\ a \cdot x - Ky &\geq 0 \\ (B-K)x_i &\geq 0 \text{ for } i = 1,2,\ldots,n \\ (B-K)y &\geq 0 \end{aligned}$$

where $y = 1$ and $x \in \{0,1\}^n$. Since $y \in \{0,1\}$, the requirement that $y = 1$ can be written $y \geq 1$. Thus an instance of the SBHILP problem has been created which has a solution if and only if the given knapsack problem has a solution. Thus SBHILP is $\mathcal{NPH}$ and hence $\mathcal{NPC}$. □

Also of interest are problems involving systems of linear *equations*. The *integer programming problem*, or IP, is defined as follows:

Definition 4.18 Integer Programming Problem
Given a matrix $A \in \mathbb{Z}^{m \times n}$ and a column vector $b \in \mathbb{Z}^m$ does there exist an $x \in \mathbb{N}^n$ such that $Ax = b$? The $\{0,1\}$ integer programming problem, *or ZERO-ONE IP, is defined as in the IP case except that the solution vector x is required to belong to $\{0,1\}^n$.*

Theorem 4.15 [Sah74] *Deciding whether there exists a solution to a given IP or ZERO-ONE IP problem is $\mathcal{NPC}$.*

If the IP problem allowed $x \in \mathbb{Z}^n$ (or $x \in \mathbb{Q}^n$) then deciding whether a solution exists is $\mathcal{P}$:

Theorem 4.16 [Knu69] *Given a matrix $A \in \mathbb{Z}^{m \times n}$ and a column vector $b \in \mathbb{Z}^m$, deciding whether there exists a vector $x \in \mathbb{Z}^n$ such that $Ax = b$ is $\mathcal{P}$.*

Table 4.1, adapted from [WW86], summaries these results; the left hand column gives the domain for the solution vector x, the top row shows the problems.

	$Ax \leq b$	$Ax \geq O$ $Ax \leq b_0 e$	$Ax = b$	$a \cdot x = b_0$	$Ax \leq 0$	$a \cdot x \leq b_0$
$\mathbb{Q}^n$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
$\mathbb{Z}^n$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$	$\mathcal{P}$
$\mathbb{N}^n$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{P}$	$\mathcal{P}$
$\{0,1\}^n$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{NPC}$	$\mathcal{P}$

Table 4.1: Complexity of integer linear inequalities and equations

4.4.3 Petri net Reachability, Coverability and Liveness

The problems listed in this section and their complexities will be useful when proving problems regarding the dynamic behaviour of Petri nets.

Definition 4.19 Reachability Problem [Pet81]
Given a Petri net $N = (P, T, B, F, M_0)$ and a marking M, is $M \in R(M_0)$?

Definition 4.20 Submarking Reachability (SMR) Problem [Pet81]
Given a Petri net $N = (P, T, B, F, M_0)$, a subset $P' \subseteq P$ and a marking M', does there exists a marking $M'' \in R(M_0)$ such that $M''(p) = M'(p)$ for all $p \in P'$?

Definition 4.21 Zero Reachability (ZR) Problem [Pet81]
Given a Petri net $N = (P, T, B, F, M_0)$, does there exist a marking $M \in R(M_0)$ such that $M(p) = 0$ for all $p \in P$?

Definition 4.22 Single Place Zero Reachability (SPZR) Problem [Pet81]
Given a Petri net $N = (P, T, B, F, M_0)$ and a place $p \in P$, does there exist a marking $M \in R(M_0)$ such that $M(p) = 0$?

Definition 4.23 The Liveness Problem [Pet81]
Given a Petri net $N = (P, T, B, F, M_0)$. For all transitions $t \in T$, is t live?

Definition 4.24 Single Transition Liveness (STL) Problem [Pet81]
Given a Petri net $N = (P, T, B, F, M_0)$ and a transition $t' \in T$, is t' live?

Theorem 4.17 [Pet81] *The reachability problem, the SMR problem, the ZR problem, the SPZR problem, the liveness problem and the STL problem are all polynomially equivalent.*

The following result of Lipton is cited in [JLL77]:

Theorem 4.18 *The reachability problem is* **EXPSPACE**-*hard.*

Since the reachability problem is **EXPSPACE**-hard so are all the problems of Theorem 4.17.

Definition 4.25 The Coverability Problem [Pet81]
Given a Petri net $N = (P, T, B, F, M_0)$ *and a marking* M'. *Does there exist a marking* $M \in R(M_0)$ *such that* $M \geq M'$?

Definition 4.26 Submarking Coverability Problem
Given a Petri net $N = (P, T, B, F, M_0)$, *a marking* M' *and a subset* $P' \subseteq P$. *Does there exist a marking* $M \in R(M_0)$ *such that* $M(p) \geq M'(p)$ *for all* $p \in P'$?

The following result, due to Lipton, is cited in [JLL77]:

Theorem 4.19 *The coverability problem is* **EXPSPACE**-*hard.*

Theorem 4.20 *The SMC problem is* **EXPSPACE**-*hard.*

Proof Given an instance of the coverability problem, construct an instance of SMC by taking $P' = P$. □

Finally, the decidability of the reachability, coverability and liveness problems are established by the following two results:

Theorem 4.21 [Pet81] *The coverability problem is decidable.*

Theorem 4.22 [May84] *The reachability problem is decidable.*

Chapter 5

Petri Net Transformations

The liveness and boundedness problems for Petri nets were introduced in Chapter 1. The discussion in Chapter 4 has shown that the analysis of these properties in the general Petri net is expensive. Petri net transformations have been proposed which preserve certain properties (for example, some subset of properties defined in Chapter 2) and which may help in the analysis of the Petri net to determine such properties. *Reduction transformations* reduce the size of the Petri net and, possibly, the size of the coverability or reachability tree, thus reducing the complexity of any analysis using coverability or reachability trees. *Synthesis transformations* can be used to synthesize a Petri net model by adding to a simple Petri net, in which the desired properties hold, additional places, transitions and arcs producing a model in which the properties are guaranteed to hold and thus no analysis is needed to determine such properties.

Definition 5.1 Transformation Rule
Consider a Petri net $N = (P, T, B, F, M_0)$. *Let* S_N *be the set of all Petri nets. A* transformation rule $\mathcal{R}$ *is a rewrite rule together with a predicate* $\mathbf{P} : S_N \rightarrow \{true, false\}$ *that produces a new Petri net* $N' = (P', T', B', F', M_0')$, *written* $N \xrightarrow{\mathcal{R}} N'$, *provided* $\mathbf{P}(N) = true$. *The transformation rule describes how the Petri* N' *is derived from* N. *The predicate* $\mathbf{P}$ *is called the* application condition *for the transformation rule* $\mathcal{R}$. *The Petri net* N *is called the* source *and* N' *the* target.

Although the definition above specifies marked Petri nets, transformation rules can also be defined on unmarked Petri nets.

Definition 5.2 Property Preserving Transformation
Let $\mathcal{R}$ *be a transformation rule,* $N = (P, T, B, F, M_0)$, $N' = (P', T', B', F', M_0')$ *be Petri*

nets such that $N \xrightarrow{\mathcal{R}} N'$ *and let* $\mathcal{C}$ *be a class of Petri nets characterised by a set of properties* $\mathcal{C}_P$. *The transformation rule* $\mathcal{R}$ *is said to be* property preserving with respect to $\mathcal{C}_P$ *if* $N \in \mathcal{C} \iff N' \in \mathcal{C}$. *Such a transformation rule is said to be* strongly sound *with respect to* $\mathcal{C}_P$. *If* $N \in \mathcal{C} \implies N' \in \mathcal{C}$ *then the transformation is said to be* sound *with respect to* $\mathcal{C}_P$ *(according to [Esp91]).*

The bi-implication (strong soundness) in the above definition is required in order for the transformation to be able to decide properties when used as a reduction transformation. This is because the properties are not known to hold in either the target or source Petri nets and an analysis of the properties in the one must hold in the other. For a transformation to be used as a synthesis transformation it need only be sound with respect to its preserved properties. This is because the desired properties are known to hold in the source Petri net and so it is only required that this imply that they hold in the target Petri net.

A reduction transformation can be reversed and used as a synthesis transformation and a synthesis transformation can be reversed and used as a reduction transformation if the properties are strongly sound with respect to the synthesis transformation. In this thesis a transformation is considered to be a reduction transformation if it reduces the size of the Petri net (either in the number of places, transitions, arcs or arc-weights) and a synthesis transformation if it increases the size of the Petri net. The direction of the transformation will be taken such that the application condition predicate is defined on the source Petri net. The application conditions of certain Petri net transformations that appear in the literature appear to specify application conditions on the target Petri net; these are, however, merely the rules specifying how the target Petri net is derived (see, for example, Esparza's refinement rules in Definition 5.18). To classify a transformation rule as "reduction" or "synthesis" based on the specification of its application conditions does not mean that the reverse transformation does not have an application condition based on the reversed transformation's source Petri net. This can be seen, for example, in the refinement of places using the S-transformation (Section 5.2.5) and merging of places (Section 5.3.2). This consideration of transformations allows the transformations in the literature to be treated as proposed by their respective authors and in certain cases both the reduction and synthesis transformations are studied.

Synthesis transformations often start with a particular Petri net in which the properties of liveness and boundedness are known to hold. The smallest such Petri nets have only one place and one transition, anything smaller would be either an isolated place or an isolated transition.

Definition 5.3 Atomic Net
A Petri net $N = (\{p_1\}, \{t_1\}, B, F)$ *where* $B, F \in \mathbb{N}^{1\times 1}$ *is called an* atomic net. *If the place* p_1 *is marked then the marked Petri net is also called an atomic net.*

Certain classes of reduction transformations aim to reduce a given Petri net into a Petri net in which it is a straightforward problem to decide certain properties. The atomic net is an example of such a Petri net in which liveness and boundedness analysis is straightforward.

This chapter surveys the transformations that have been proposed in the literature. Only transformations that preserve both liveness and boundedness ("well-behavedness") have been considered. For example, transformation rules $\mathcal{R}_1$ and $\mathcal{R}_2$ (which, together, generate a class of Petri nets called *state machine decomposable* nets) from [Esp90] and transformation rule TA4 (addition of non-restricting nets) from [Ber86a] have not been considered. In addition, the synthesis work of [YC88] considers rather specialised connections to restricted Petri nets, and has not been included in this thesis. Apart from this "filtering", all the transformations that the author has come across in the literature are included in this thesis. In the following sections reduction transformations are surveyed. These are followed by synthesis transformations and then finally, transformations that completely reduce classes of Petri net. [Mur89] describes six simple transformation rules that preserve liveness, boundedness and safeness, however, these are special cases of other transformations and hence are not included in this survey in their simple forms.

5.1 Reduction Transformations

The transformations in this section are due to [Ber86a], [Ber86b] and [BRV80]. These transformations make localised changes to the structure of the Petri net and sometimes also to the initial marking. Other transformations in [Ber86a], which Berthelot calls *addition of nets* are synthesis transformations (see Section 5.2).

The Petri net properties that are preserved by these transformations can be split into three groups [Ber86a]:

1. **General reachability.** These are the properties that describe the set of all reachable markings:
   - Safety,
   - Boundedness and
   - S-invariant covering.

2. **Specific reachability.** These properties describe the reachability of special markings:
   - Proper termination,
   - Home state and
   - unavoidable state.
3. **Transition liveness.** These properties concern the liveness of transitions:
   - Pseudo liveness,
   - Quasi-liveness and
   - Liveness.

Furthermore, [Ber86a] splits application conditions into two groups:

1. **Structural conditions.** These application conditions depend only on the (static) structure of the Petri net and not on the dynamic behaviour of the Petri net.
2. **Behavioural conditions.** These application conditions depend both on the dynamic behaviour and on the particular initial marking (as well as the structure) of the Petri net.

The application conditions considered here concern themselves as much as possible with the structure of the Petri net but some need a partial knowledge of the dynamic behaviour and as such are marking dependent.

According to [Ber86a] the "classes" of transformation are:

- Place transformations (see Sections 5.1.1, 5.1.2 and 5.1.3),
- Fusion of transitions (see Sections 5.1.4, 5.1.5 and 5.1.6) and
- Addition of Petri nets (see Section 5.2).

5.1.1 Simplification of Redundant Places

If a place is such that its marking can never inhibit the firing of any transition connected to it then it is a redundant place. This transformation removes redundancies with regard to a place by removing arcs from transitions surrounding the place. If all arcs are removed then the place can be removed from the Petri net. Such places are said to be *redundant* which [Ber86a] defines in terms of the application conditions of this transformation:

Definition 5.4 Redundant Place. *A place $p' \in P$ is said to be* redundant *if and only if there exists an m-vector $v \in \mathbb{Z}^m$ such that the following four conditions apply:*

1. $$v(p) \begin{cases} > 0 & \text{if } p = p' \text{ and} \\ \leq 0 & \text{if } p \neq p'. \end{cases} \tag{5.1}$$

2. $$\exists b \in \mathbb{N} : v^T M_0 = b. \tag{5.2}$$

3. $$v^T B \leq b e^T \tag{5.3}$$

   where e is the n-vector with a one for each component.

4. $$v^T (B - F) \leq O^T. \tag{5.4}$$

These conditions are a reformulation, by the author, in linear algebra of the conditions given in [Ber86a]. This linear algebra formulation allows the problem to be compared to (integer) linear programming problems. Definition 5.4 is the definition of *structurally* redundant places in [Ber86a] probably because it does not depend on the evolution of the Petri net. However, it does depend on the initial marking and therefore should be considered a behav'oural property and not a structural property.

After having identified the redundant place the transformation can be performed by removing all edges in $\bullet p'$ and $p'\bullet$, then for every transition, $t_i \in T$, an edge with weight

$$w_i = -v^T C e_i \tag{5.5}$$

is added from t to p'. If all the weights w_i are zero then the place p' is isolated and can be removed.

Table 5.1 from [Ber86a] lists properties which remain invariant under simplification of redundant places. In this table and all the other tables that list properties preserved by transformations, the second column gives the soundness of the transformation with respect to the property; $\Longleftrightarrow$ indicates that the transformation is strongly sound, $\Longrightarrow$ indicates that the transformation is sound, and $\Longleftarrow$ indicates that the reverse transformation is sound. Furthermore, the third column gives additional conditions that must be satisfied for the property to be sound.

Figure 5.1 from [Ber86a] shows a part of a Petri net that has a redundant place, p_3 in N. N' is the same part of the Petri net after the transformation of simplifying redundant places. In this example all the weights are zero and the place p_3 has been removed.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longrightarrow$	
S-invariant Covering	$\Longleftrightarrow$	$\Leftarrow \forall t_i \in T, w_i = 0$
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	$\Rightarrow \forall t_i \in T, w_i = 0$
Unavoidable state	$\Longleftrightarrow$	$\Rightarrow \forall t_i \in T, w_i = 0$
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.1: Properties invariant under simplification of redundant places

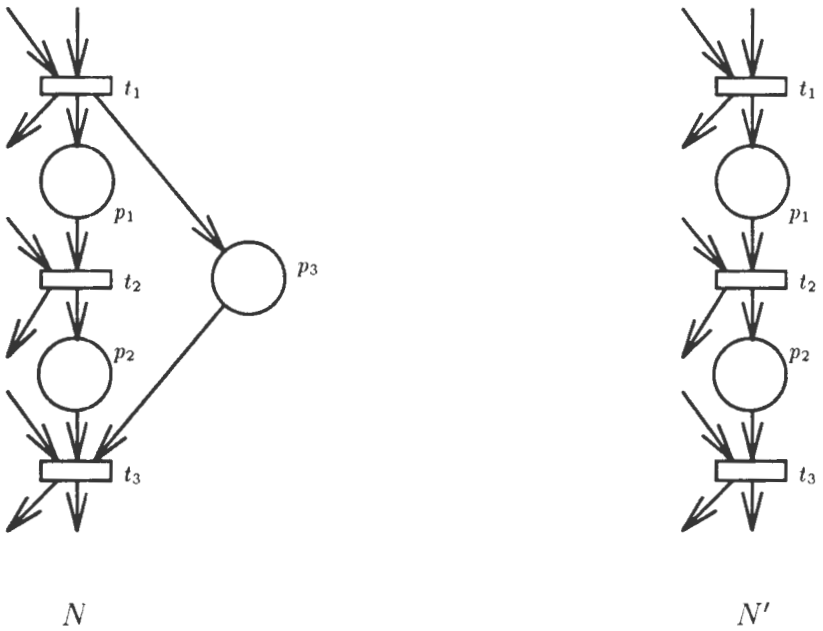


Figure 5.1: Simplification of redundant places

5.1.2 Fusion of Doubled Places

When one combines two places into a single place and is able to distinguish to which of the original two places tokens in the combined place belonged (by markings on sets of other places) then the two places are doubled. [Ber86a] defines doubled places in terms of the application conditions of this transformation:

Definition 5.5 Doubled Places [Ber86a]
Two places $p, p' \in P, p \neq p'$ *are* doubled *if and only if the following application conditions hold:*

1. $\exists I_1, I_2, \subseteq P : p \in I_1$ *and* $\forall t \in T : p \in {}^\bullet t \Longrightarrow \exists q \in I_2 : q \in {}^\bullet t$ (5.6)
and an m-vector $v \in \mathbb{Z}^m$ *such that*

$$v(p) \begin{cases} > 0 & \text{if } p \in I_1, \\ < 0 & \text{if } p \in I_2 \text{ and} \\ = 0 & \text{otherwise.} \end{cases}$$

$$\begin{aligned} v^T M_0 &\leq 0. \\ v^T B &\geq O^T. \\ v^T (F - B) &\leq O^T. \end{aligned}$$

2. $\exists I_1', I_2', \subseteq P : p' \in I_1'$ *and* $\forall t \in T : p' \in {}^\bullet t \Longrightarrow \exists q \in I_2' : q \in {}^\bullet t$ (5.7)
and an m-vector $v' \in \mathbb{Z}^m$ *such that*

$$v'(p) \begin{cases} > 0 & \text{if } p \in I_1', \\ < 0 & \text{if } p \in I_2' \text{ and} \\ = 0 & \text{otherwise.} \end{cases}$$

$$\begin{aligned} v'^T M_0 &\leq 0. \\ v'^T B &\geq O^T. \\ v'^T (F - B) &\leq O^T. \end{aligned}$$

3. $I_2 \cap I_2' = \emptyset.$ (5.8)

4. $\forall M \in R(PN, M_0) :$ (5.9)
$\sum_{q \in I_2} M(q) > 0 \Longrightarrow \sum_{q \in I_2'} M(q) = 0$ *and*
$\sum_{q \in I_2'} M(q) > 0 \Longrightarrow \sum_{q \in I_2} M(q) = 0.$

5. $\forall t \in T :$ (5.10)
$B(p, t) > 0 \Longrightarrow B(p', t) = 0$ *and*
$B(p', t) > 0 \Longrightarrow B(p, t) = 0.$

After having identified two places as being doubled places the transformation can be performed. This is done by fusing the two doubled places, that is, places p and p' are removed and a single place, p'', is inserted into the Petri net such that $\bullet p'' = \bullet p \cup \bullet p'$ and $p''\bullet = p\bullet \cup p'\bullet$.

Table 5.2 from [Ber86a] lists properties which remain invariant under fusion of doubled places.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftrightarrow$	
S-invariant Covering	$\Longleftrightarrow$	$\Rightarrow v(p) = v(p')$
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	
Unavoidable state	$\Longleftrightarrow$	
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.2: Properties invariant under fusion of doubled places

Figure 5.2 from [Ber86a] shows a part of a Petri net N that contains doubled places p and p' and the same part of the Petri net N' after the transformation of fusing the doubled places.

5.1.3 Fusion of Equivalent Places

If the set of markings reachable after the firing of one transition is the same for two different places because of the markings of those places then the places are said to be equivalent [Ber86a]:

Definition 5.6 Equivalent Places [Ber86a]
Let $N = (P, T, B, F, M_0)$ be a marked Petri net. Two places $p_1, p_2 \in P$ are equivalent *if and only if $\exists t_1, t_2 \in T$, $t_1 \neq t_2$ such that the following conditions hold:*

1. $B(p_1, t_1) = 1$ *and* $B(p_2, t_2) = 1$.
2. $B(p_2, t_1) = 0$ *and* $B(p_1, t_2) = 0$.
3. $\forall t \in T \setminus \{t_1, t_2\}$, $B(p_1, t) = 0$ *and* $B(p_2, t) = 0$.
4. $\forall q \in P \setminus \{p_1, p_2\}$, $B(q, t_1) = B(q, t_2)$ *and* $F(q, t_1) = F(q, t_2)$.

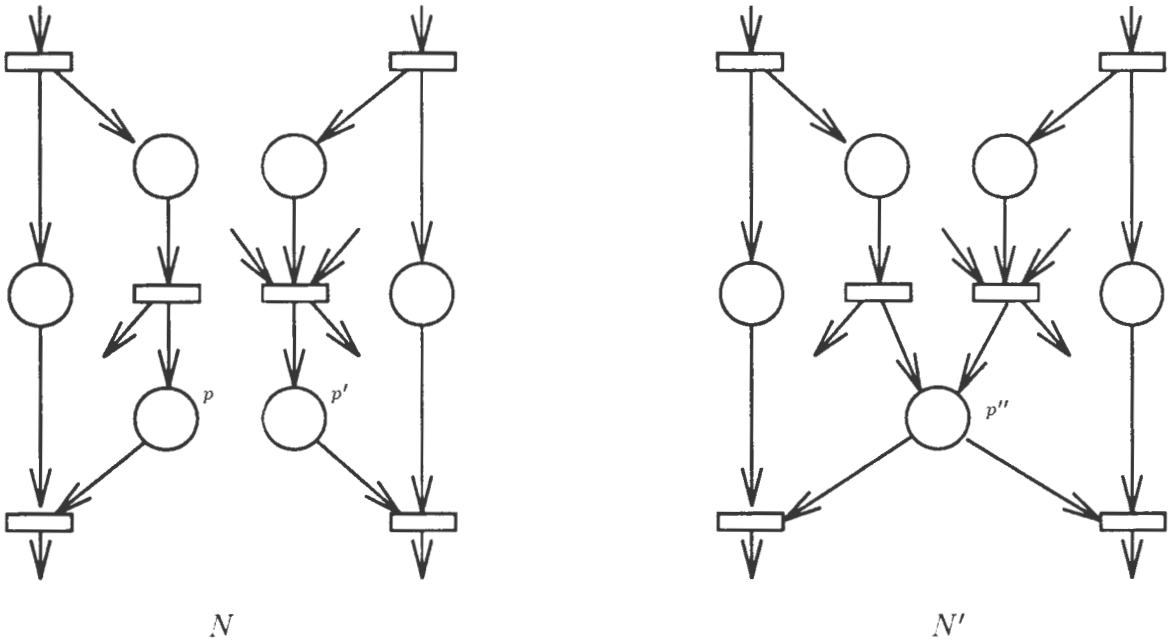


Figure 5.2: Fusion of doubled places

5. $\exists t \in T : p_1 \in t\bullet$ *and* $\exists t \in T : p_2 \in t\bullet$.

After having established that the places p_1 and p_2 are equivalent, the transformation is performed by fusing places p_1 and p_2 as described above in Section 5.1.2. Transitions t_1 and t_2 are *identical transitions* and can subsequently be fused (see the algorithm in Section 5.1.4 for fusion of transitions).

Table 5.3 from [Ber86a] lists properties which remain invariant under fusion of equivalent places.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftarrow$	
S-invariant Covering	$\Longleftrightarrow$	$\Rightarrow v(p_1) = v(p_2)$
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	$\Leftarrow M_h(p_1) = 0$ and $M_h(p_2) = 0$
Unavoidable state	$\Longleftrightarrow$	$\Leftarrow M_u(p_1) = 0$ and $M_u(p_2) = 0$
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.3: Properties invariant under fusion of equivalent places

Figure 5.3 from [Ber86a] shows a part of a Petri net N containing equivalent places, p_1 and p_2 and the same part of the Petri net N' after fusion of the equivalent places to form the place p_0.

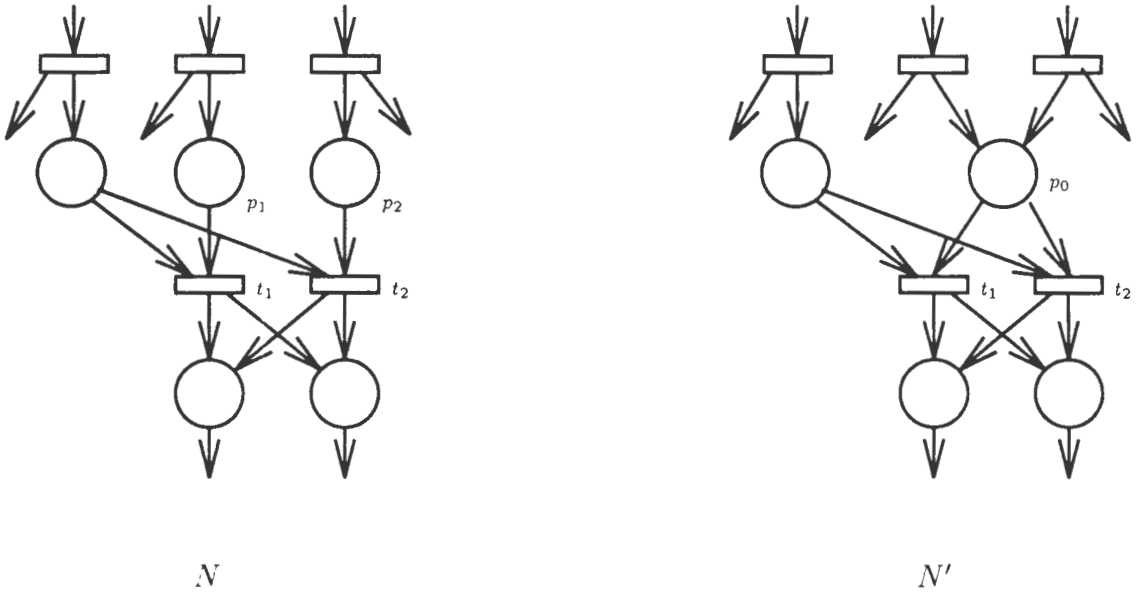


Figure 5.3: Fusion of equivalent places

5.1.4 Post-Fusion of Transitions

Sets of transitions T_F and T_H can be fused if all the transitions in T_F have a single place p as input, and the place p has as only inputs the transitions of T_H. More precisely, [Ber86a] defines structural post-fusion in terms of the application conditions of the transformation:

Definition 5.7 Structural Post-Fusion [Ber86a]
Let $N = (P, T, B, F, M_0)$ be a marked Petri net. A subset $T_F \subseteq T$ is said to be structurally post-fusible *with a subset $T_H \subseteq T$ if and only if $\exists p \in P$ and $m \in \mathbb{N}^+$ such that the following conditions hold:*

1. $\forall f \in T_F, \forall q \in P, B(q,f) = \begin{cases} m & \text{if } q = p, \\ 0 & \text{if } q \neq p \text{ and} \end{cases}$

   $\forall f \in T_F, F(p,f) = 0.$

2. $\exists f \in T_F : \exists q \in P : F(q,f) > 0.$

3. $\forall h \in T_H, B(p,h) = 0$ *and* $\exists k_h \in \mathbb{N} : F(p,h) = mk_h.$

$$4. \quad \forall t \in T \setminus (T_H \cup T_F), B(p,t) = 0 \text{ and } F(p,t) = 0. \tag{5.11}$$

Once the sets T_F and T_H are found, such that T_F is post-fusible with T_H, the transformed Petri net is obtained by performing the following procedure: Let $N = (P, T, B, F, M_0)$ with $T = T_F \cup T_H \cup T_0$ where T_F and T_H are structurally post-fusible. The Petri net $N' = (P', T', B', F', M_0')$ can be constructed by fusing the sets $T_F \subseteq T$ and $T_H \subseteq T$ of post-fusible transitions:

Algorithm 5.1 Fusion of Transitions

1. $P' \longleftarrow P$.
2. $T_F' \longleftarrow T_F$ and for each $t_h \in T_H$ and for all $p \in P$, either one of the following conditions hold:
   (a) $t_h \bullet \cap \bullet T_F = \emptyset$ and $\bullet t_h \cap T_F \bullet = \emptyset$ then for all $t_f \in T_F$:
      i. $B'(p, t_f') \longleftarrow B(p, t_h) + B(p, t_f)\ \forall t_h \in T_H$ and $\forall t_f \in T_F$.
      ii. $F'(t_f', p) \longleftarrow F(t_f, p) + F(t_h, p)\ \forall t_h \in T_H$ and $\forall t_f \in T_F$.
   (b) $t_h \bullet \cap \bullet T_F \neq \emptyset$ then for all $t_f \in T_F$:
      i. $B'(p, t_f') \longleftarrow B(p, t_h) + \max\{0, B(p, t_f) - F(t_h, p)\}\ \forall t_h \in T_H$ and $\forall t_f \in T_F$.
      ii. $F'(t_f', p) \longleftarrow F(t_f, p) + \max\{0, F(t_h, p) - B(p, t_f)\}\ \forall t_h \in T_H$ and $\forall t_f \in T_F$.
   (c) $\bullet t_h \cap T_F \bullet \neq \emptyset$ then for all $t_f \in T_F$:
      i. $B'(p, t_f') \longleftarrow B(p, t_f) + \max\{0, B(p, t_h) - F(t_f, p)\}\ \forall t_h \in T_H$ and $\forall t_f \in T_F$.
      ii. $F'(t_f', p) \longleftarrow F(t_h, p) + \max\{0, F(t_f, p) - B(p, t_h)\}\ \forall t_h \in T_H$ and $\forall t_f \in T_F$.
3. $T' \longleftarrow T_0' \cup T_F'$, where
   (a) $T_0' \longleftarrow T_0$.
   (b) $B'(s', t') \longleftarrow B(s, t)\ \forall s' \in P'$ and $\forall t' \in T_0'$ together with the corresponding $s \in P$ and $t \in T_0$.
   (c) $F'(s', t') \longleftarrow F(s, t)\ \forall s' \in P'$ and $\forall t' \in T_0'$ together with the corresponding $s \in P$ and $t \in T_0$.
4. $M_0' \longleftarrow M_0$.

Algorithm 5.1 is not only used for the application condition of post-fusion but is a general procedure for the fusion of transitions [Ber86b].

Table 5.4 from [Ber86a] lists properties which remain invariant under fusion of post-fusible transitions.

Figure 5.4 from [Ber86a] shows a part of a Petri net N containing sets of post-fusible transitions $T_H = \{t_h\}$, $T_F = \{t_{f_1}, t_{f_2}\}$ with respect to the place p. The figure also shows part of the Petri net, N', after performing the transformation which resulted in the removal of the place p and transition t_h, and the replacement of transitions t_{f_1} and t_{f_2} by t'_{f_1} and t'_{f_2}, respectively.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftrightarrow$	$\Leftarrow$ If no multiple arcs exist.
S-invariant Covering	$\Longleftrightarrow$	
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	
Unavoidable state	$\Longleftrightarrow$	
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.4: Properties invariant under fusion of post-fusible transitions

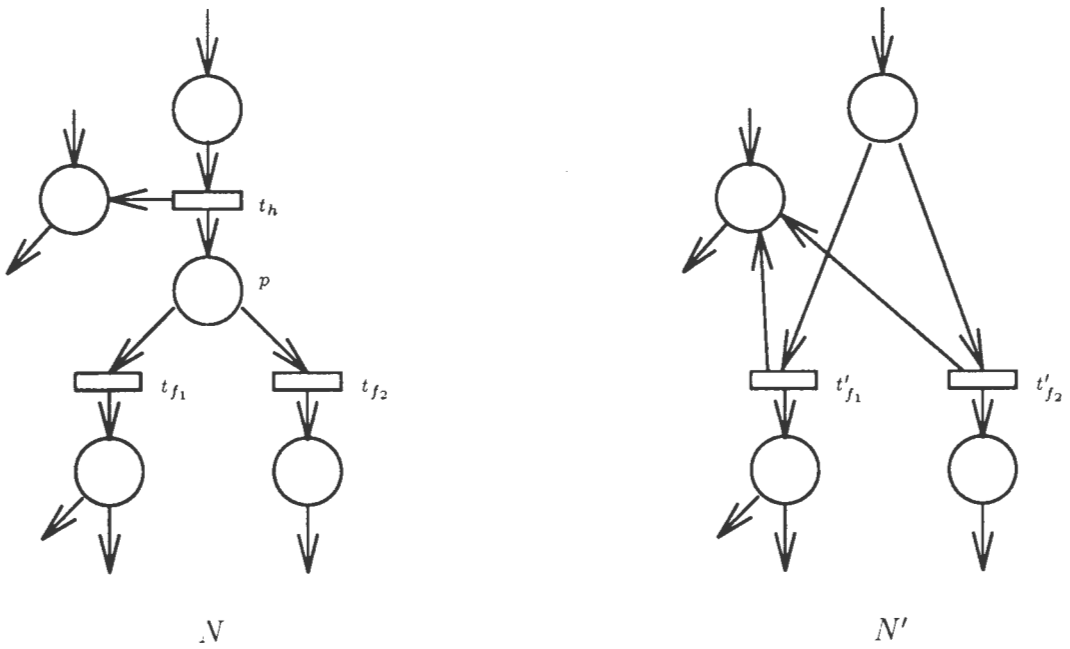


Figure 5.4: Post-fusion of transitions

5.1.5 Pre-Fusion of Transitions

As with post-fusion, pre-fusion identifies transitions that can be fused. However, in this case, if a transition h has a place p as its only output and it is one of the, possible many, inputs of transitions in a set of transitions T_F then T_F is said to be pre-fusible with h. More precisely, [Ber86a] defines "pre-fusible" in terms of the application conditions of the transformation:

Definition 5.8 Pre-Fusion [Ber86a]
Let $N = (P, T, B, F, M_0)$ be a marked Petri net. A subset $T_F \subseteq T$ is said to be pre-fusible *with a transition $h \in T$ if and only if $\exists p \in P$ such that the following conditions hold:*

1. $$\forall s \in P, F(s,h) = \begin{cases} 1 & \text{if } s = p, \\ 0 & \text{if } s \neq p \text{ and} \end{cases} \tag{5.12}$$

   $$B(p,h) = 0. \tag{5.13}$$

2. $\exists s \in P : B(s,h) > 0.$

3. $\forall f \in T_F,\ B(p,f) = 1 \text{ and } F(p,f) = 0.$ (5.14)

4. $\forall t \in T \setminus T_F \cup \{h\},\ B(p,t) = 0 \text{ and } F(p,t) = 0.$ (5.15)

5. $M_0(p) = 0.$ (5.16)

6. $\forall q \in P, \forall t \in T \setminus \{h\},\ B(q,h) \neq 0 \Rightarrow B(q,t) = 0.$

Once the set T_F of transitions and the transition h have been found, such that T_F is pre-fusible with h, the transformation can be performed in the same manner as in Section 5.1.4 by letting the set $T_H = \{h\}$.

Table 5.5 from [Ber86a] lists properties which remain invariant under fusion of pre-fusible transitions.

Figure 5.5 from [Ber86a] shows a part of a Petri net N containing sets of pre-fusible transitions with $T_H = \{h\}$, $T_F = \{t_{f_1}, t_{f_2}\}$ and p, and the same part of the Petri net N' after pre-fusion of the transitions resulting in the removal of place p and transition h, and the replacement of transitions t_{f_1} and t_{f_2} by t'_{f_1} and t'_{f_2}, respectively.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftrightarrow$	
S-invariant Covering	$\Longleftrightarrow$	
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	
Unavoidable state	$\Longleftrightarrow$	
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.5: Properties invariant under fusion of pre-fusible transitions

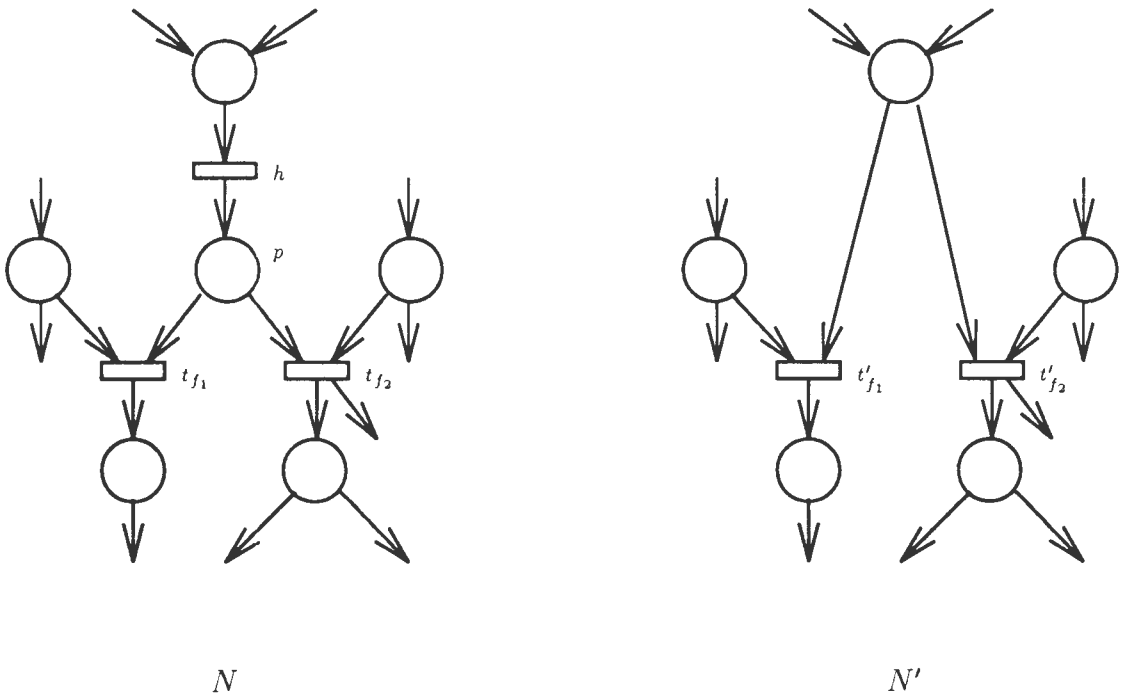


Figure 5.5: Pre-fusion of transitions

5.1.6 Lateral Fusion of Transitions

The fusion of laterally fusible transitions is similar to the case of pre-fusible transitions except that the fused transitions now have symmetric roles in that they either both precede a common transition or they both follow a common transition [Ber86a].

Definition 5.9 Lateral Fusion [Ber86a]
Let $N = (P, T, B, F, M_0)$ be a marked Petri net. Two transitions $t_1, t_2 \in T$ are laterally fusible *if and only if there exists two places $p_1, p_2 \in P$ and a transition $t_c \in T$ such that either:*

1. *t_c precedes t_1 and t_2 and the following conditions hold:*

   (a) $F(p_1, t_c) = 1,\ F(p_2, t_c) = 1$ *and* (5.17)

   $\forall t \in T \setminus \{t_c\},\ F(p_1, t) = 0$ *and* $F(p_2, t) = 0.$ (5.18)

   (b) $\exists s_1, s_2 \in P : B(s_1, t_i) > 0$ *and* $F(s_2, t_i) > 0$ *for* $i = 1, 2.$ (5.19)

   (c) $B(p_1, t_1) = 1$ *and* $\forall t \in T \setminus \{t_1\},\ B(p_1, t) = 0.$ (5.20)

   (d) $B(p_2, t_2) = 1$ *and* $\forall t \in T \setminus \{t_2\},\ B(p_2, t) = 0.$ (5.21)

   (e) *for* $i, j = 1, 2,\ i \neq j,$ (5.22)

   $$F(s, t_i) > 0, s \in P \Rightarrow B(q, t_j) = \begin{cases} 1 & \text{if } q = p_j \text{ and} \\ 0 & \text{if } q \neq p_j. \end{cases}$$

   (f) $M_0(p_1) = M_0(p_2).$ (5.23)

   (g) $\forall q \in P,\ B(q, t_i) > 0 \Rightarrow B(q, t) = 0, t \neq t_i$ *for* $i = 1, 2.$ (5.24)

2. *Or, t_c follows p_1 and p_2 and the following conditions hold:*

   (a) $B(p_1, t_c) = 1,\ B(p_2, t_c) = 1$ *and*

   $\forall t \in T \setminus \{t_c\},\ B(p_1, t) = 0,$ *and* $B(p_2, t) = 0.$

   (b) $\exists s_1, s_2 \in P : F(s_1, t_i) > 0$ *and* $B(s_2, t_i) > 0$ *for* $i = 1, 2.$

   (c) $F(p_1, t_1) = 1$ *and* $\forall t \in T \setminus \{t_1\},\ F(p_1, t) = 0.$

   (d) $F(p_2, t_2) = 1$ *and* $\forall t \in T \setminus \{t_2\},\ F(p_2, t) = 0.$

   (e) *for* $i, j = 1, 2,\ i \neq j,$

   $$B(s, t_i) > 0, s \in P \Rightarrow F(q, t_j) = \begin{cases} 1 & \text{if } q = p_j \text{ and} \\ 0 & \text{if } q \neq p_j. \end{cases}$$

   (f) $M_0(p_1) = M_0(p_2).$

(g) $\forall q \in P,\ F(q,t_i) > 0 \Rightarrow F(q,t) = 0, t \neq t_i$ *for* $i = 1,2$.

Once laterally fusible transitions t_1 and t_2 have been found then the fusion of t_1 and t_2 can be performed using Algorithm 5.1.

Table 5.6 from [Ber86a] lists properties which remain invariant under fusion of laterally fusible transitions.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftrightarrow$	
S-invariant Covering	$\Longleftrightarrow$	
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	
Unavoidable state	$\Longleftrightarrow$	
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.6: Properties invariant under fusion of laterally fusible transitions

Figure 5.6 from [Ber86a] shows a part of a Petri net, N, containing two laterally fusible transitions, t_1 and t_2, and the same part of the Petri net, N', with fused transition t_{12}.

5.2 Synthesis Transformations

The transformations in this section can be used to build Petri net models that are guaranteed to have certain properties. These transformations are due to [Ber86a], [Val79], [SM83] [Esp90, ES90] and [SX92].

The synthesis transformations can be further split into the following broad classes:

1. **Net Additions.** These transformations are performed by adding one Petri net to another by fusing together places or transitions. The application conditions apply to the constituent Petri nets (see Sections 5.2.1, 5.2.2 and 5.2.3).

2. **Element Refinement.** These transformations perform a stepwise refinement of Petri nets. This is done by refining transitions or places of a Petri net by replacing the transition or place with a Petri net (see Sections 5.2.4, 5.2.5 and 5.2.6).

3. **Module Decomposition.** This method decomposes a system into subsystems or *modules*, where the modules are modelled by marked graphs. These modules are

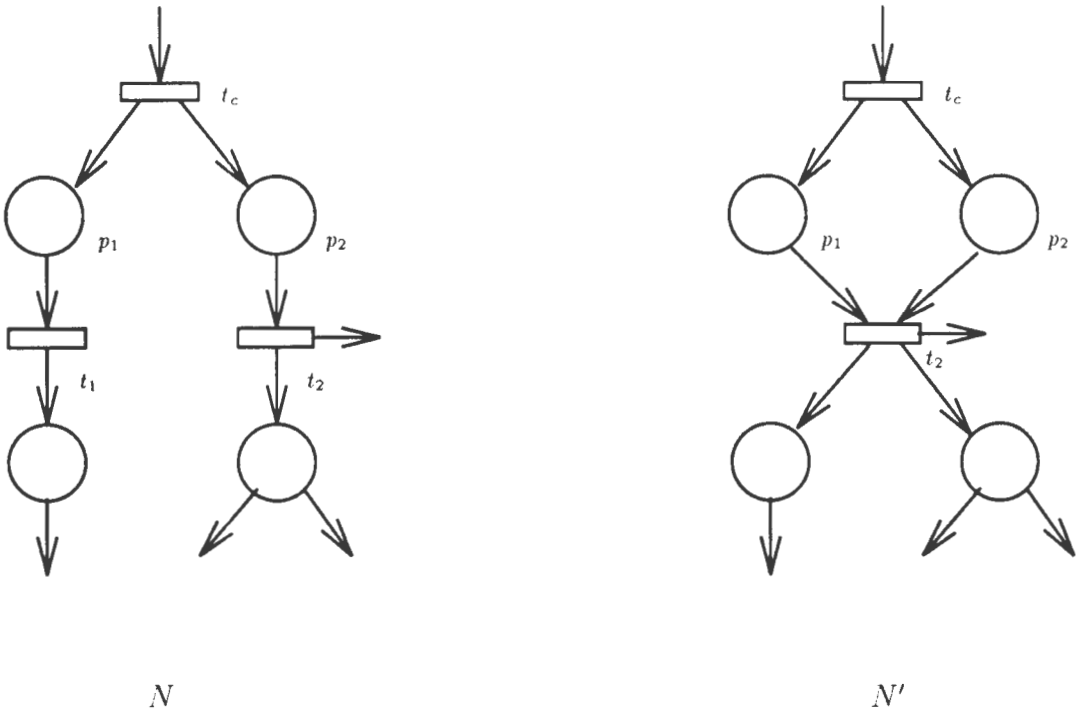


Figure 5.6: Lateral fusion of transitions

integrated to form the model of the system. Finally, the modules can be simplified without changing the model's behaviour using a "reduction" transformation (see Section 5.2.7).

5.2.1 Addition of a Derivable Subnet

This transformation duplicates a subnet of a Petri net. This refines the process by adding an alternate path along which tokens can flow. The tokens which take this alternate route are removed from the original Petri net, flow through the added subnet and then are returned to the original Petri net. This transformation does not change the dynamic behaviour of the Petri net as each firing sequence in the transformed Petri net has a corresponding firing sequence in the original Petri net (replacing any transitions in the added subnet with the corresponding transitions of the subnet in the original Petri net) and each firing sequence in the original Petri net is still a firing sequence in the transformed Petri net [Ber86a].

[Ber86a] defines a derivable subnet in terms of the application conditions that must be satisfied before this transformation can be applied:

Definition 5.10 Derivable Subnet [Ber86a]
Let $N = (P, T, B, F, M_0)$ be a marked Petri net. A subnet $N_s = (P_s, T_s, B_s, F_s)$ *of N is* derivable *if and only if the following conditions hold:*

1. *N_s is an* open subnet *of N, that is, P_s contains all the places of P which are connected to transitions of T_s.*
2. *N_s is a marked graph (according to the alternate definition of marked graphs — see Chapter 2).*
3. *N_s is connected and contains no circuit.*
4. *$\exists t_1 \in T_s$ such that there is a path from t_1 to every other transition of N_s.*
5. *N_s is not* re-entrant*; for all markings $M \in R(M_0)$ reached by N, every path included in N_s (a marked graph) contains at most one token.*
6. *P_s can be partitioned into:*
   - $P_E = \{p \in P_s : p \notin T_s\bullet\}$.
   - $P_S = \{p \in P_s : p \notin \bullet T_s\}$.
   - $P_I = \bullet T_s \cap T_s\bullet$.
7. *N_s can be emptied by firing transitions in T_s and non-conflicting transitions of T.*
8. $(T \setminus T_s)\bullet \cap P_I = \emptyset$.

After having identified a derivable subnet the addition of the subnet transformation is completed in two steps (let N_d be the duplicate of the derivable subnet):

1. The inputs of N_d are set to the inputs of N_s and the outputs of N_d are set to the outputs of N_s, i.e. $\bullet N_d = \bullet N_s$ and $N_d\bullet = N_s\bullet$.
2. N_d is added to N by fusing inputs and outputs of N_s and N_d leaving the internal places distinct.

Table 5.7 from [Ber86a] lists properties which remain invariant under addition of a derivative net.

Figure 5.7 shows a part of a Petri net N with a derivable subnet and the same part of the Petri net N' after duplicating the subnet.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftrightarrow$	
S-invariant Covering	$\Longleftrightarrow$	
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	
Unavoidable state	$\Longleftrightarrow$	
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.7: Properties invariant under addition of a derivative net

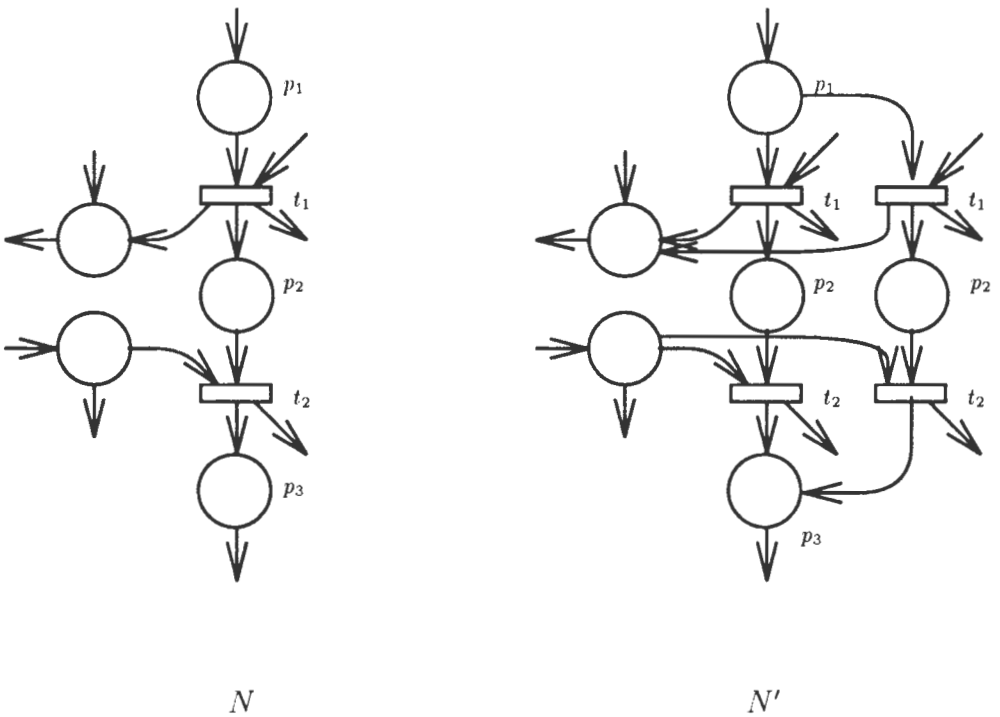


Figure 5.7: Addition of a derivable subnet

5.2.2 Regulation of Laterally Fusible Transitions

The application conditions of this transformation are the same as those for fusion of laterally fusible transitions (see Section 5.1.6, page 51). The difference between this transformation, and the fusion of laterally fusible transitions, is that instead of having a single transition firing, being the fused transition, the firings between the two transitions are alternated by the inclusion of a circuit between the two transitions that contains one token.

Definition 5.11 Regulated Transitions [Ber86a]
Two transitions t_1 and t_2 may be regulated *if and only if they are laterally fusible transitions. The regulation is established by adding two places p_a and p_b and arcs forming the circuit*

$$t_1 \longrightarrow p_a \longrightarrow t_2 \longrightarrow p_b \longrightarrow t_1$$

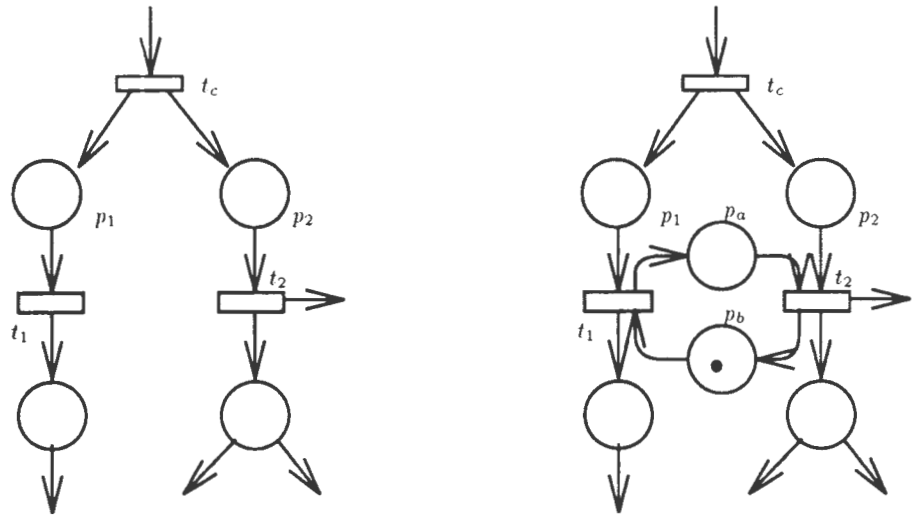
containing exactly one token.

Table 5.8 from [Ber86a] lists properties which remain invariant when two laterally fusible transitions are regulated.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftrightarrow$	
S-invariant Covering	$\Longleftrightarrow$	
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftrightarrow$	
Unavoidable state	$\Longleftrightarrow$	
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.8: Properties invariant under regulation of laterally fusible transitions

Figure 5.8 shows a part of a Petri net N with laterally fusible transitions t_1 and t_2. Figure 5.8 also shows the same part of the Petri net N' after transitions t_1 and t_2 have been regulated by adding the places p_a and p_b, and the arcs to form the circuit consisting of these transitions and places.



N N'

Figure 5.8: Regulation of laterally fusible transitions

5.2.3 Regulation of Identical Transitions

Identical transitions are those that have the same input places and output places. The behaviour does not change if identical transitions in a Petri net are regulated since it does not matter which of the identical transitions fire as the same markings are still reachable.

Definition 5.12 Identical Transitions [Ber86a]
Given a Petri net $N = (P, T, B, F)$*, a set of transitions* $\{t_1, t_2, \ldots, t_n\} \subseteq T$ *are said to be* identical *if*

$$\begin{aligned} B(p,t_1) &= B(p,t_2) = \cdots = B(p,t_n) \text{ and} \\ F(p,t_1) &= F(p,t_2) = \cdots = F(p,t_n) \end{aligned}$$

for all $p \in P$.

The transformation is performed by adding a set of places $p_1, p_2, \ldots, p_n$ to the Petri net N and a circuit

$$p_1 \rightarrow t_1 \rightarrow p_2 \rightarrow t_2 \rightarrow \cdots \rightarrow t_n \rightarrow p_1$$

containing a non-zero number of tokens.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Safe	$\Longleftrightarrow$	
S-invariant Covering	$\Longleftrightarrow$	
Proper Termination	$\Longleftrightarrow$	
Home State	$\Longleftarrow$	
Unavoidable state	$\Longleftarrow$	
Pseudo-Live	$\Longleftrightarrow$	
Quasi-Live	$\Longleftarrow$	
Live	$\Longleftrightarrow$	

Table 5.9: Properties invariant under regulation of identical transitions

Table 5.9 from [Ber86a] lists properties that remain invariant under regulation of identical transitions.

Figure 5.9 shows a part of a Petri net N with identical transitions t_1 and t_2. Also shown is the same part of the Petri net N' after adding the places p_1 and p_2 corresponding to each of the identical transitions t_1 and t_2 and the arcs to form a circuit of the identical transitions and the added places. This circuit must also contain a non-zero number of tokens (not shown).

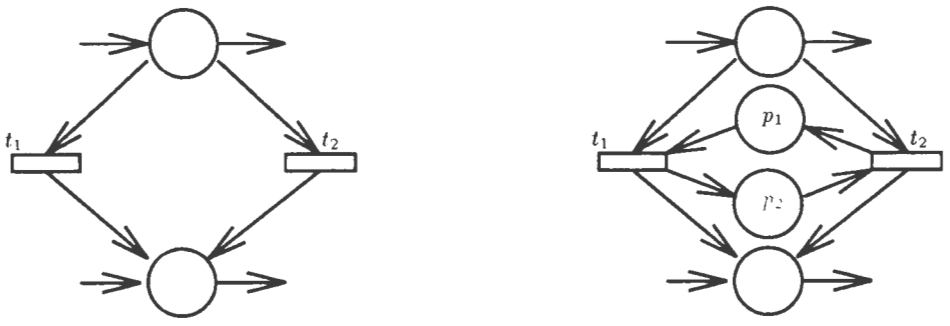


Figure 5.9: Regulation of identical transitions

5.2.4 Stepwise Refinement of Transitions

This refinement transformation is performed by replacing a transition of a Petri net, in which certain conditions hold, by another Petri net, for which certain other conditions

also hold such that the resultant Petri net is live and bounded. This transformation has its origins in [Val79] and was generalised by [SM83]:

Definition 5.13 k**-enabled Transitions [SM83]**
A Petri net $N = (P, T, B, F, M_0)$ *has a transition* $t \in T$ *which is said to be* k-enabled *for* $k \in \Pi^+$ *in* N *if and only if there exists a marking* $M \in R(M_0)$ *such that*

$$kB(p,t) \leq M(p), \forall p \in P.$$

Definition 5.14 Block and Associated Petri Net [SM83]
A Petri net $N = (P, T, B, F, M_0)$ *with two distinct transitions* $t_{in}, t_{out} \in T$ *is called a* block. *The Petri net* $B(N, t_{in}, t_{out}, k) = (P \cup \{p_0\}, T, B_B, F_B, M_{0B})$ *is called the* associated Petri net *of the block* N, *and is defined as follows:*

1. $$B_B(p,t) = \begin{cases} 1 & t = t_{in}, p = p_0, \\ 0 & t \neq t_{in}, p = p_0 \text{ and} \\ B(p,t) & p \in P. \end{cases}$$

2. $$F_B(p,t) = \begin{cases} 1 & t = t_{out}, p = p_0, \\ 0 & t \neq t_{out}, p = p_0 \text{ and} \\ F(p,t) & p \in P. \end{cases}$$

3. $$M_{0B}(p) = \begin{cases} k & p = p_0 \text{ and} \\ M_0(p) & p \in P \end{cases}$$

   where $k \in \Pi^+$.

The place $p_0 \notin P$ *is called the* idle place.

Definition 5.15 k**-well-behaved [SM83]**
A Petri net $N = (P, T, B, F, M_0)$ *is said to be* k-well-behaved *with respect to two distinct transitions* $t_{in}, t_{out} \in T$ *if and only if the following conditions hold:*

1. t_{in} *is live in* $B(N, t_{in}, t_{out}, k)$.
2. *For each* $\sigma_1 \in L(B(N, t_{in}, t_{out}, k))$ *such that* $\#(\sigma_1, t_{in}) > \#(\sigma_1, t_{out})$, *there exists* $\sigma_2 \in (T \setminus \{t_{in}\})^+$ *such that* $\sigma_1\sigma_2 \in L(B(N, t_{in}, t_{out}, k))$ *and* $\#(\sigma_1, t_{in}) = \#(\sigma_1\sigma_2, t_{out})$.
3. $\#(\sigma, t_{in}) \geq \#(\sigma, t_{out})$ *for any* $\sigma \in L(B(N, t_{in}, t_{out}, k))$.

[SM83] shows that for $n \in \mathbb{N}^+$, if a Petri net N is $(n+1)$-well-behaved with respect to t_{in} and t_{out}, then N is n-well-behaved.

Let $N = (P, T, B, F, M_0)$ and $N' = (P', T', B', F', M_0')$ be Petri nets such that $P \cap P' = \emptyset$ and $T \cap T' = \emptyset$. Assume that for some $k \in \mathbb{N}^+$, a transition $t_0 \in T$ is *not* (k+1)-enabled in N and that N' is k-well-behaved with respect to two distinct transitions $t_{in}, t_{out} \in T'$. The transformation of refining transition t_0 yields

$$N'' = \mathrm{TR}(N, N', t_0, t_{in}, t_{out}) = (P'', T'', B'', F'', M_0'')$$

defined as follows:

1. $P'' = P \cup P'$.

2. $T'' = (T \cup T'') \setminus \{t_0\}$.

3. $$B''(p,t) = \begin{cases} 0 & t \in T \setminus \{t_0\}, p \in P', \\ 0 & t \in T' \setminus \{t_{in}\}, p \in P, \\ B(p,t) & t \in T \setminus \{t_0\}, p \in P, \\ B'(p,t) & t \in T', p \in P' \text{ and} \\ B(p,t_0) & t = t_{in}, p \in P. \end{cases}$$

4. $$F''(p,t) = \begin{cases} 0 & t \in T \setminus \{t_0\}, p \in P', \\ 0 & t \in T' \setminus \{t_{out}\}, p \in P, \\ F(p,t) & t \in T \setminus \{t_0\}, p \in P, \\ F'(p,t) & t \in T', p \in P' \text{ and} \\ F(p,t_0) & t = t_{out}, p \in P. \end{cases}$$

5. $$M_0''(p) = \begin{cases} M_0(p) & p \in P \text{ and} \\ M_0'(p) & p \in P'. \end{cases}$$

Table 5.10 lists properties that remain invariant under refinement of transitions [SM83].

Figure 5.10 from [SM83] shows a Petri net N with a transition t_0 that is 3-enabled and *not* 4-enabled. Figure 5.11 shows the Petri net N' which is a block with respect to transitions t_{in} and t_{out} and is 3-well-behaved and *not* 4-well-behaved. Figure 5.12 shows the Petri net $B(N', t_{in}, t_{out}, 3)$ associated with the block and Figure 5.13 shows the Petri net N'' which is the result of refining transition t_0 in N using N'.

5.2.5 Stepwise Refinement of Places

This transformation allows a place to be refined into a Petri net by introducing transitions. The reverse of this transformation, a reduction, could thus be used to ignore certain activities or events [SM83].

Property		Conditions
Bounded	$\Longleftrightarrow$	$\Rightarrow \forall p \in P'$: p is bounded in $B(N', t_{\text{in}}, t_{\text{out}}, k)$
Safe	$\Longleftrightarrow$	$\Rightarrow B(N', t_{\text{in}}, t_{\text{out}}, 1)$ is safe
Live	$\Longleftrightarrow$	$\Rightarrow \forall M \in R(M_0) : t_0$ is $k-$enabled in (P, T, B, F, M) and $B(N', t_{\text{in}}, t_{\text{out}}, k)$ is live

Table 5.10: Properties invariant under stepwise refinement of transitions

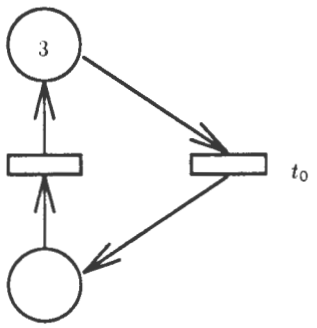


Figure 5.10: Petri net N in which t_0 is 3-enabled

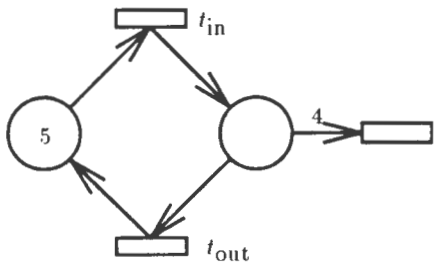


Figure 5.11: Block N' which is 3-well-behaved and not 4-well-behaved

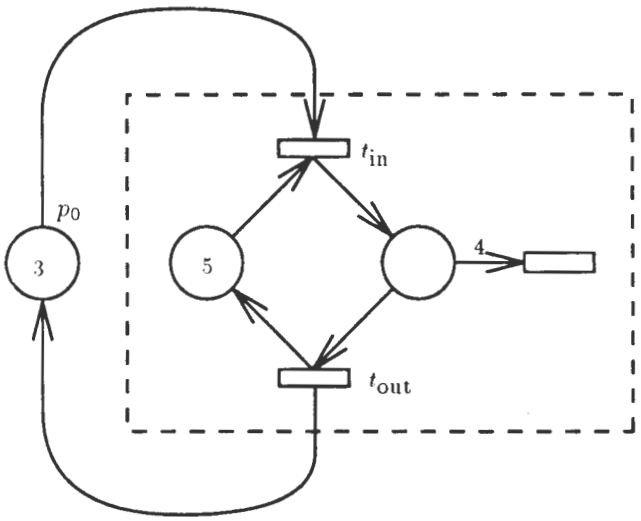


Figure 5.12: Associated Petri net $B(N', t_{\text{in}}, t_{\text{out}}, 3)$ of block N'

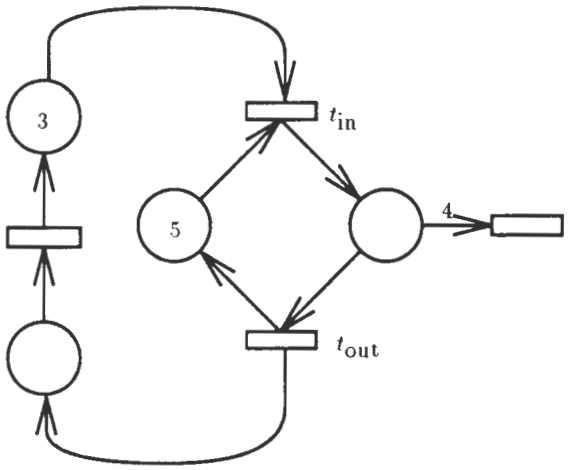


Figure 5.13: Petri net N'' produced by refining t_0 in N using N'

Definition 5.16 S-transformation [SM83]
Let $N = (P, T, B, F, M_0)$ *be a marked Petri net with some place* $p_0 \in P$. *The Petri net* $N' = (P', T', B', F', M_0')$ *is derived from* N *using the* S-transformation *where*

1. $P' = (P \setminus \{p_0\}) \cup \{p_{01}, p_{02}\}$,

2. $T' = T \cup \{t_0\}$,

3. $$B'(p,t) = \begin{cases} B(p_0,t) & p = p_{02} \text{ and } t \neq t_0, \\ 1 & p = p_{01} \text{ and } t = t_0, \\ 0 & p \neq p_{01} \text{ and } t = t_0 \text{ and} \\ B(p,t) & \text{otherwise.} \end{cases}$$

4. $$F'(p,t) = \begin{cases} F(p_0,t) & p = p_{01} \text{ and } t \neq t_0, \\ 1 & p = p_{02} \text{ and } t = t_0, \\ 0 & p \neq p_{02} \text{ and } t = t_0 \text{ and} \\ F(p,t) & \text{otherwise.} \end{cases}$$

5. $$M_0'(p) = \begin{cases} M_0(p) & p \in P \setminus \{p_0\}, \\ M_0(p_0) & p = p_{01} \text{ and} \\ 0 & p = p_{02}. \end{cases}$$

Table 5.11 lists properties that remain invariant under the S-transformation [SM83].

Property		Conditions
Bounded	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	$\Rightarrow \bullet p_0 \neq \emptyset$

Table 5.11: Properties invariant under S-transformation

Figure 5.14 shows a subnet N of a Petri net N and the place p_0 and a subnet N' of the same Petri net after applying the S-transformation to the place p_0 [SM83]. The transition t_0 introduced by the transformation can now be refined further by using the transition refinement transformation described in Section 5.2.4. The preservation of properties is transitive so Table 5.10 and Table 5.11 can be used to determine which properties are preserved by the combined transformations (and the conditions for the preservation).

5.2.6 Generating Classes of Nets From Kits of Rules

[Esp90] presented his rules in the form of *kits* which are classes of rules. The name of the class of rules or kit is due to the class of Petri nets that it generates (although the Petri

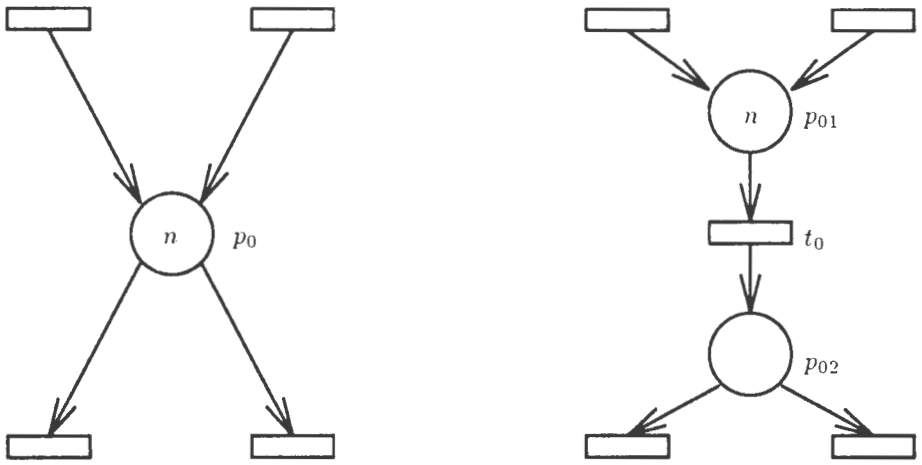


Figure 5.14: Refinement of place p_0

nets produced by a kit need not be restricted to the class). Of the kits in [Esp90], it is only the *SL&SB kit* which preserves liveness and boundedness. The SL&SB kit produces only *structurally live* and *structurally bounded* Petri nets. In this stepwise refinement technique the atomic net is not marked and nothing is said of the actual markings which ensure that the system is live and bounded, only that such a marking exists. This transformation is also described in [ES90].

The following definitions are used by [Esp90] to define the transformations of the SL&SB kit.

Definition 5.17 Refinement Kits and Classes [Esp90]
A refinement kit *is a set* $\{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ *of transformations, called* refinement transformations. *The class of Petri nets generated by* $\{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$, *denoted* $\mathcal{N}(\mathcal{R}_1, \ldots, \mathcal{R}_k)$, *is the smallest class of Petri nets given by:*

- $N_0 \in \mathcal{N}(\mathcal{R}_1, \ldots, \mathcal{R}_k)$, *where* N_0 *is an atomic net in which the arc weights are one.*
- *If* $N \in \mathcal{N}(\mathcal{R}_1, \ldots, \mathcal{R}_k)$ *and* $N \xrightarrow{\mathcal{R}_i} N''$ *where* $\mathcal{R}_i \in \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ *then* $N' \in \mathcal{N}(\mathcal{R}_1, \ldots, \mathcal{R}_k)$.

Definition 5.18 SL&SB Kit Rules
Let $N = (P, T, B, F)$ *and* $N' = (P', T', B', F')$ *be two unmarked Petri nets. The SL&SB kit comprises two refinement rules* $\mathcal{R}_3$ *and* $\mathcal{R}_4$ *([Esp90] numbered these transformation rules and did not name them).*

1. **Refinement Rule $\mathcal{R}_3$.** $N \xrightarrow{\mathcal{R}_3} N'$ *if and only if the following conditions hold:*

   (a) $P' = P \cup \{p_0\}$, $p_0 \notin P$ *and* $T' = T$.

   (b)
   $$B'(p,t) = B(p,t)\ \forall p \in P, \forall t \in T$$
   $$F'(p,t) = F(p,t)\ \forall p \in P, \forall t \in T$$
   and
   $$\exists t' \in T : B'(p_0,t') \neq 0$$
   $$\exists t'' \in T : F'(p_0,t'') \neq 0$$

   (c) The row in the incidence matrix corresponding to the place p_0 is a linear combination of the rows corresponding to the other places of N.

2. **Refinement Rule $\mathcal{R}_4$.** $N \xrightarrow{\mathcal{R}_4} N'$ *if and only if* $\exists p \in P$ *and a P-graph* $N'' = (P'', T'', B'', F'') \subseteq N'$ *such that the following conditions hold:*

   (a) $N = N_0$ *or* $\forall t \in {}^\bullet p, |t^\bullet| > 1$ *and* $\forall t \in p^\bullet, |{}^\bullet t| > 1$.

   (b) N' is obtained by replacing p in N by N''.

   (c) $\forall p'' \in P''$, *there exists a path* $(p''_{in}, \ldots, p'', \ldots, p''_{out})$ *in* N'', *where* p''_{in} *and* p''_{out} *are a way-in place (has an input transition outside of N'') and a way-out place (has an output transition outside of N'') of N'', respectively.*

   (d) $\forall p'' \in P''$ *and for all way-out places* $p''_{out} \in P''$ *there exists and path* $(p'', \ldots, p''_{out})$ *in* N''.

The refinement rule $\mathcal{R}_4$ always operates on places added to the Petri net by the refinement rule $\mathcal{R}_3$ except when $\mathcal{R}_4$ is applied to the atomic net N_0 [Esp90].

Table 5.12 lists properties that remain invariant under the refinement rules of the SL&SB kit [Esp90].

Property		Conditions
Structurally Bounded	$\Longrightarrow$	
Structurally Live	$\Longrightarrow$	

Table 5.12: Properties invariant under the refinement rules of the SL&SB kit

A rule is said to be *local* if it can be determined whether the application conditions hold by examining a small environment of a node of the graph. In the SL&SB kit the refinement rule $\mathcal{R}_4$ is a local rule while the refinement rule $\mathcal{R}_3$ is a *non-local* rule. It has been conjectured that it is not possible to generate precisely the class of strongly connected graphs using local rules alone. Since live and bounded Petri nets are strongly connected

[Mur89], it may not be possible to synthesize live and bounded Petri nets using only local rules [Esp90].

Figure 5.15 shows a Petri net which is refined in Figure 5.16 using refinement rule $\mathcal{R}_4$ and which is further refined in Figure 5.17 using refinement rule $\mathcal{R}_3$ [Esp90].

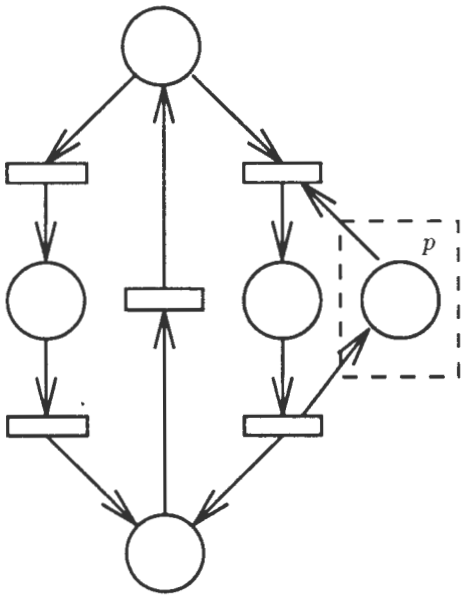


Figure 5.15: Petri net before refinement using an SL&SB refinement rule

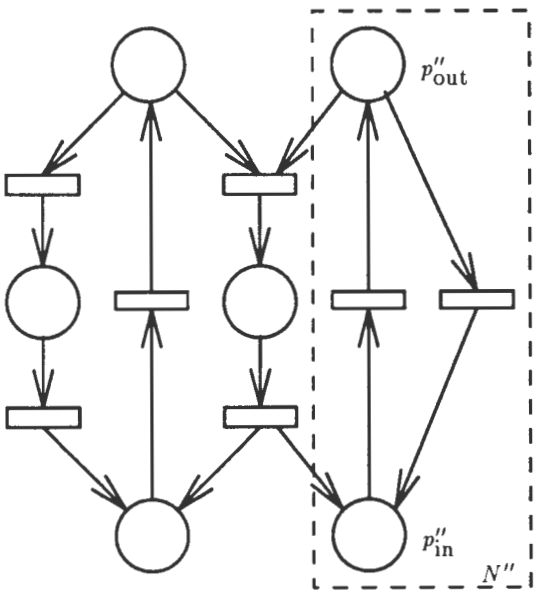


Figure 5.16: Petri net after application of refinement rule $\mathcal{R}_4$ of the SL&SB kit

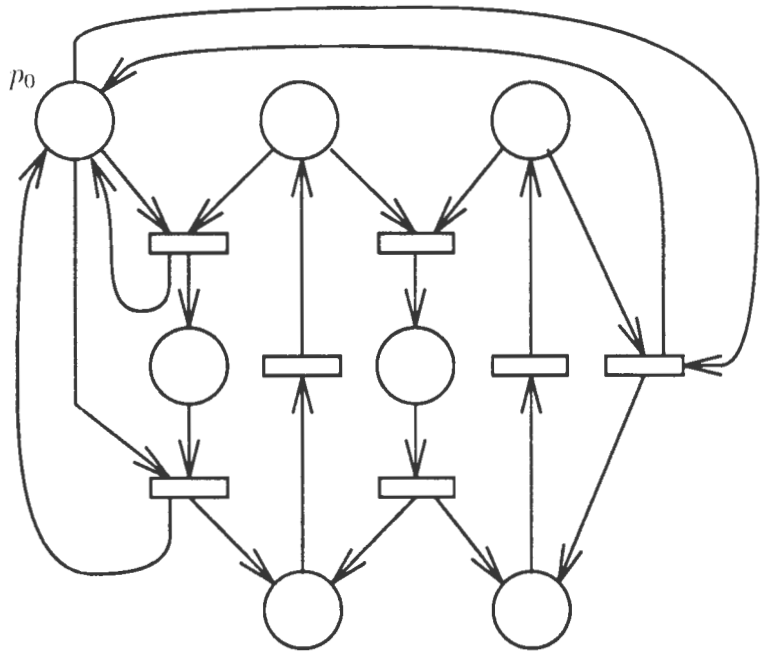


Figure 5.17: Petri net after application of refinement rule $\mathcal{R}_3$ of the SL&SB kit

[Esp90] also introduces a refinement rule $\mathcal{R}_5$ which is a restriction of $\mathcal{R}_3$, and this rule is used to define a new kit that generates exactly the structurally live and structurally bounded free choice nets:

Definition 5.19 The Free Choice Kit
Let $N = (P, T, B, F)$ *and* $N' = (P', T', B', F')$ *be two unmarked Petri nets in* $\mathcal{N}(\mathcal{R}_4, \mathcal{R}_5)$ $\{\mathcal{R}_4, \mathcal{R}_5\}$ *is called the* free choice kit *where* $\mathcal{R}_4$ *is as defined in the SL&SB kit and* $\mathcal{R}_5$ *is defined as follows:* $N \xrightarrow{\mathcal{R}_5} N'$ *if and only if* $N \xrightarrow{\mathcal{R}_3} N'$ *and the place* p_0 *introduced by* $\mathcal{R}_3$ *is such that* $|p_0 \bullet| = 1$.

5.2.7 Event Graph Module Decomposition

The placement of this work as a synthesis transformations needs some clarification. The method described here builds models of systems from components which can be modeled by marked graphs and in this sense is a synthesis transformation. However, the method also abstracts the individual marked graphs by a transformation which is actually a reduction transformation.

Event graphs or marked graphs are an important sub-class of Petri nets which can model synchronization and in which liveness and boundedness characterisation is straightforward (Chapter 3).

Definition 5.20 Augmented Event Graph [SX92]
An augmented event graph *is composed of:*

1. *An event graph.*
2. *A set of places which make tokens available to the event graph. These are called* input places.
3. *A set of places which remove tokens from the event graph. These are called* output places.

A transition of the event graph which follows an input place is called an input transition. *A transition of the event graph which precedes an output place is called an* output transition. *The transitions of the event graph excluding the input transitions and output transitions are called* internal transitions.

The models considered are those that use Petri nets in which basic modules are integrated to form the model of the system and in which each module can be modelled by an augmented event graph. The integration is performed by merging the input and output places of the modules [SX92].

Definition 5.21 Token Distance [Mur89]
The token distance $d(t_i, t_j)$ *between two transitions in a marked graph* $N = (P, T, B, F, M_0)$ $(t_i, t_j \in T)$ *is the minimum token content among all possible paths from* t_i *to* t_j *at the marking* M_0. *If no path exists between* t_i *and* t_j *then the token distance is defined as* ∞.

The reduction transformation is performed on each event graph module by replacing them with their minimal representations:

Definition 5.22 Minimal Representation [SX92]
Given a marked event graph module $N = (P, T, B, F, M_0)$. *Let* $T_{in} \subseteq T$, $T_{out} \subseteq T$ *denote the sets of input and output transitions, respectively. The* minimal representation $N' = (P', T', B', F', M_0')$ *of* N *is derived as follows:*

- *For all* $t_1 \in T_{in}$ *and for all* $t_2 \in T_{in} \cup T_{out}$ *there is a place* $p_{12} \in P'$ *connecting* t_1 *to* t_2 *in* N' *if and only if there exists a path from* t_1 *to* t_2 *in* N.
- *The number of tokens in* p_{12} *is equal to the token distance from* t_1 *to* t_2 *in the event graph* N.

Property		Conditions
Bounded	$\Longleftrightarrow$	
Live	$\Longleftrightarrow$	

Table 5.13: Properties preserved by replacing event graph modules by their minimal representations

Table 5.13 lists properties which remain invariant when the event graph modules are replaced by their minimal representations [SX92].

Figure 5.18 is an example of a Petri net model with modules (details omitted) in which input and output places and transitions are shown. Figure 5.19 shows the reduced Petri net model after replacing the event graph modules by their minimal representations.

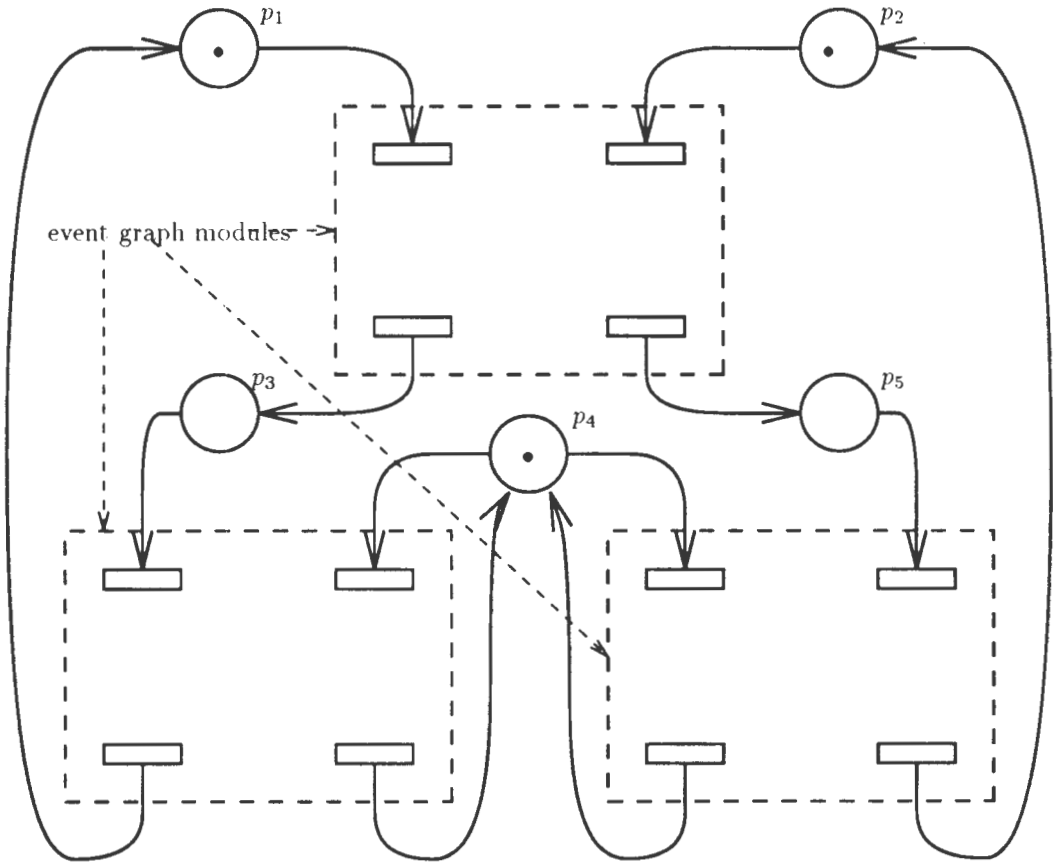


Figure 5.18: Petri net with integrated event graph modules

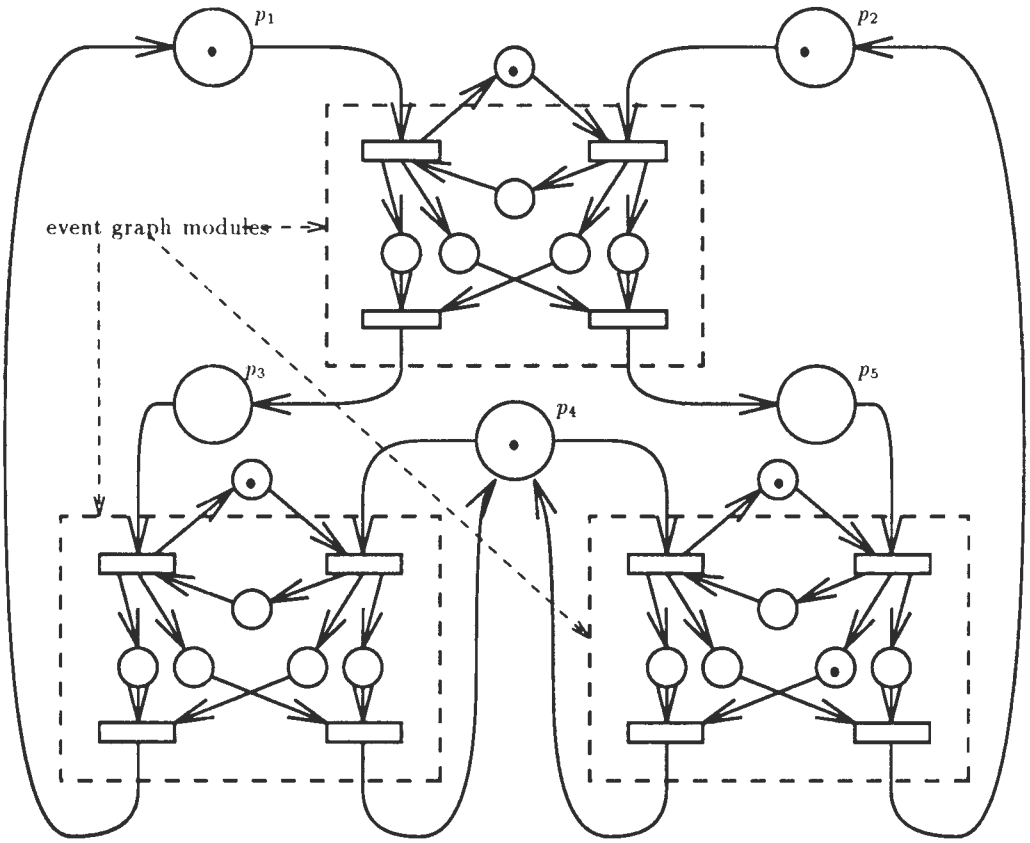


Figure 5.19: Petri net with integrated minimal representation event graph modules

5.3 Complete Reduction of Classes of Petri nets

The kits of synthesis rules presented in [Esp90] (Section 5.2.6) generated the classes of structurally live and structurally bounded Petri nets and structurally live and structurally bounded free choice nets. Such refinement kits would have to be used in software systems where the operator guides the construction of the Petri net, unless some goal could be communicated to the software tool, for example, a formal specification in some other language. In the absence of such goals it would be useful if a kit of rules could *reduce* a constructed Petri net, possibly restricted to a class of Petri net, to a suitable small Petri net or a subclass of Petri net in which the analysis was straight forward. This section surveys the proposals of [Des90] and [Esp91], both of which are based on free choice nets.

[Des90] presents a set of reduction rules that can reduce live and *safe* free choice Petri nets without frozen tokens (defined below) and shows that these rules can always reduce such a Petri net to either a marked graph or a state machine. [Esp91] presents a set of reduction rules that reduces precisely the class of live and bounded free choice nets to an atomic net. Unlike the synthesis transformations of Section 5.2.6 these sets of reduction rules are not restricted to structural properties but are defined on marked Petri nets. Furthermore, the classes of Petri nets in this section are characterized by these sets of rules so their failure to reduce a given Petri net to one of the target classes of Petri net means that the corresponding properties did not hold for the original Petri net.

Some of the transformations in this section are special cases of the other reduction transformations (covered earlier).

The importance of free choice nets can be seen in their power to model certain programming systems [Pet81].

5.3.1 Well-behaved Free Choice Systems

[Des90] describes four local rules which reduce well-behaved free choice systems to a system that has no concurrency or a system that has no choice, the idea being that the analysis of such reduced systems is simpler. In the sense of the application conditions either concurrency or choice is strictly local and for this reason these rules, which are local, can remove one of these properties in the process of reduction. These two "classes" are not disjoint and determining whether a particular system belongs in any of these classes can be done without having to apply the reduction rules [Des90].

The reduction transformations of this section and the results are applicable to live and safe free choice systems without frozen tokens:

Definition 5.23 Frozen Tokens

Consider a safe Petri net $N = (P, T, B, F, M_0)$ initially marked with some initial marking M_0. If there exists an infinite firing sequence $\sigma \in L(N)$ such that for some place $p \in P$ where p is marked at M_0 and marked for the whole of the firing sequence σ then the token on p is said to be a frozen token.

Definition 5.24 Well-Behaved System [Des90]

A well-behaved system *is a live and safe free choice net with an initial marking void of any frozen tokens.*

Definition 5.25 Well-formed Net [Des90]

A Petri net $N = (P, T, B, F)$ is termed a well-formed net *if there exists an initial marking M_0 such that the Petri net N together with the initial marking M_0 is a well-behaved system.*

[Des90] shows that under his set of rules a system is well-behaved if and only if the reduced system is well-behaved. A Petri net is well-formed if and only if the reduced system is well-formed. Furthermore, [Des90] shows that a well-behaved system can be reduced by his set of rules if and only if the well-formed Petri net can be reduced.

Also of importance is the fact that each well-formed net can be reduced to either a state machine or a marked graph [Des90]. Results concern to which type of Petri net (state machine or marked graph) the well-formed Petri net can be reduced. Each well-formed Petri net is covered by P-components and by T-components (since a well-formed Petri net is also a well-formed free choice Petri net). This is used to characterise the reduced Petri net. A well-formed Petri net can be reduced to a marked graph if and only if its T-components have a non-empty intersection. In addition, a well-formed Petri net can be reduced to a state machine if and only if its P-components have a non-empty intersection [Des90].

The Place Reduction

Consider a marked free choice Petri net $N = (P, T, B, F, M_0)$ with distinct places $p_1, p_2 \in P$. If for all $t \in T$, $B(p_1, t) = B(p_2, t)$ and $F(p_1, t) = F(p_2, t)$, and $M_0(p_1) \neq 0 \iff M_0(p_2) \neq 0$, then p_1 and p_2 are "co-redundant" according to [Ber86a] in the strong sense (the redundant place can be completely removed as its simplification isolates it, see Section 5.1.1) [Des90].

The above conditions are the application conditions of the place reduction (or P-reduction) transformation, which is performed by removing one of the places p_1 or p_2.

In addition to the resultant system being well-behaved if N is well-behaved, the other properties shown to be preserved by [Ber86a] for redundant places are also preserved by the P-reduction.

Figure 5.20 from [Des90] shows a part of a Petri net N in which the P-reduction is applicable on the left and on the right is the part of the Petri net N' after the transformation has been applied.

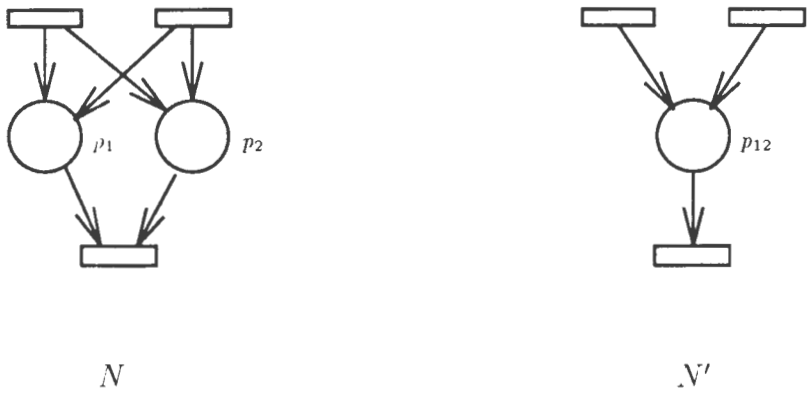


Figure 5.20: Applying the P-reduction

The Transition Reduction

For a marked free choice net $N = (P, T, B, F, M_0)$, if two distinct transitions $t_1, t_2 \in T$ are identical, that is, for all $p \in P$ $B(p, t_1) = B(p, t_2)$ and $F(p, t_1) = F(p, t_2)$, then for all $M \in R(M_0)$ such that $M \xrightarrow{t_1} M'$ it is true that $M \xrightarrow{t_2} M'$ and vice versa. These are the application conditions of the transition reduction (or T-reduction) transformation [Des90].

Ignoring the exact actions represented by such transitions allows one of them to be removed as they both result in the same state change in the system.

This transformation is also called fusion of parallel transitions and when applied to the general Petri net preserves liveness, boundedness and safeness [Mur89]. The T-reduction is also a special case lateral fusion of transitions (Section 5.1.6).

Figure 5.21 from [Des90] shows a part of a Petri net N in which the T-reduction is applicable on the left and on the right is the part of the Petri net N' after the transformation has been applied.

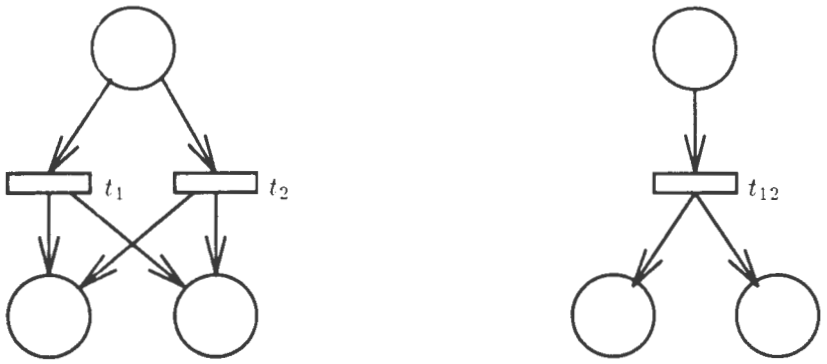


Figure 5.21: Applying the T-reduction

The Flow Reduction

The application rules for the flow reduction (or F-reduction) are as follows: Let $N = (P, T, B, F, M_0)$ be a marked free choice net, then the F-reduction is applicable if there exists a place $p \in P$ and a transition $t \in T$ such that t is the only output of p and p is the only input of t, p is not an output of t and there is no contact situation with respect to t (contact situations are those that threaten the safety of the Petri net) [Des90].

The F-reduction is performed by removing the place p and the transition t. The inputs to the place p are duplicated as inputs to all the output places of the transition t. All the outputs of the transition t are duplicated as outputs of all the transitions that are inputs to the place p. The new initial marking is obtained as follows: If p is marked in M_0 then each of the outputs of t are marked in the modified Petri net and if p is not marked at M_0 then the markings of all the other places remain unchanged.

The F-reduction is a special case of post-fusion of transitions [Ber86a], and so all the properties preserved by that transformation are inherited by the F-reduction.

Figure 5.22 from [Des90] shows a part of a Petri net N in which the F-reduction is applicable on the left and on the right is the part of the Petri net N' after the transformation has been applied.

The "A" Reduction

The name of this transformation is somewhat of a misnomer as it does not reduce the size of the free choice net. Although [Des90] showed that this transformation preserves the

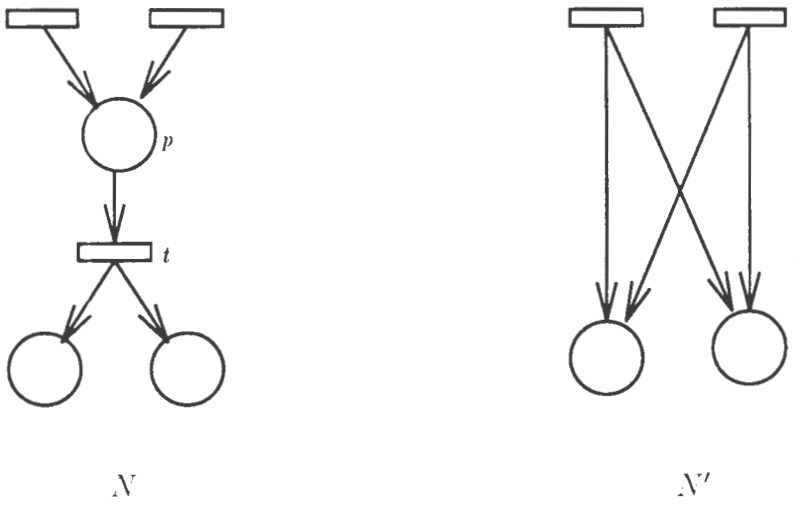


Figure 5.22: Applying the F-reduction

well-behavedness property, he states that it is an open problem whether A-reductions are necessary (to reduce a system to a state machine or a marked graph). This transformation can, however, reduce the number of T-components [Des90].

For the application conditions of the A-reduction, consider a marked free choice net $N = (P, T, B, F, M_0)$ which is P-reduced (the P-reduction cannot be applied). If there exists transitions $t_1, t_2 \in T$ such that $\bullet t_2 \subseteq t_1 \bullet$ and $|\bullet t_2| > 1$ then the A-reduction is applicable [Des90].

The transformation is applied by removing the arcs from t_1 to the input places of t_2 and adding arcs from t_1 to the output places of t_2. The A-reduction does not change any markings.

Figure 5.23 from [Des90] shows a part of a Petri net N in which the A-reduction is applicable on the left and on the right is the part of the Petri net N' after the transformation has been applied.

5.3.2 Reduction of Live and Bounded Free Choice Nets

The subclass of Petri nets that these transformation rules completely reduces is a 'relaxation' on the class of well-formed nets (Section 5.3.1) in that frozen tokens are permitted and the net need not be safe. This relaxation increases the modelling power of the Petri nets allowed in the subclass and the reduction rules are also more powerful in the sense that complete reducibility (to an atomic net) is always possible. Since a well-behaved system is a live and bounded free choice net, these rules will also completely reduce well-behaved

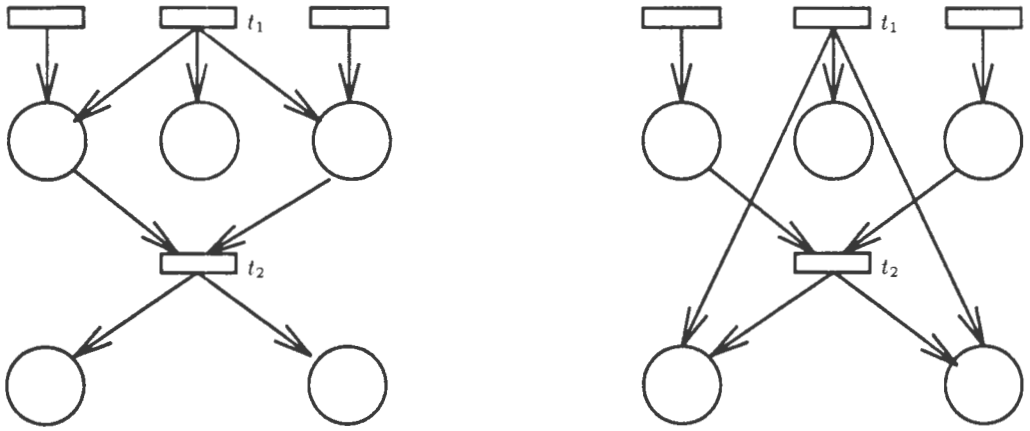


Figure 5.23: Applying the A-reduction

systems.

Definition 5.26 Well-formed Free Choice Nets [Esp91]
A Petri net $N = (P, T, B, F)$ *is a* well-formed free choice net *(WFFC net) if and only if there exists an initial marking* M_0 *such that the Petri net* N *together with the initial marking* M_0 *is a live and bounded free choice system.*

The above definition states that the Petri net must have the free choice structural property, be structurally live and structurally bounded in order to be a well-formed free choice net. Furthermore, it states that live and bounded free choice (LBFC) systems can be characterised in terms of their underlying WFFC net: A Petri net $N = (P, T, B, F, M_0)$ is LBFC system if and only if the underlying unmarked Petri net is a WFFC net and that every P-component is marked at M_0, (Theorem 3.10).

[Esp91]'s rules are organised into *kits* and it is these kits which reduce the subclass of Petri nets to an atomic net.

Merging Places — Rule 1

For a marked free choice net $N = (P, T, B, F, M_0)$ if there exists $t_1 \in T$ such that $|\bullet t_1| = |t_1 \bullet| = 1$, $\bullet(\bullet t_1) \neq \emptyset$ and $(\bullet t_1)\bullet = \{t_1\}$ then $p_1, p_2 \in P$ can be merged where $\{p_1\} = \bullet t_1$ and $\{p_2\} = t_1 \bullet$.

The result of applying this reduction transformation is a new marked free choice net $N' = (P', T', B', F', M_0')$ where $P' = (P \setminus \{p_1, p_2\}) \cup \{p_{12}\}$ and p_{12} is a new place created

as a result of merging places p_1 and p_2, $T' = T \setminus \{t_1\}$ and all input arcs to p_1 in N are inputs arcs to p_{12} in N' and all output arcs of p_2 in N are output arcs of p_{12} in N'. The new initial marking is obtained by placing on p_{12} the sum of the initial markings on p_1 and p_2. The initial marking of all the other places remains the same [Esp91].

This reduction transformation is also called *fusion of series transitions* and preserves, in both directions, liveness, safeness and boundedness in the general Petri net [Mur89].

Figure 5.24 shows a part of a Petri net N in which the merging of places is applicable on the left and on the right is the part of the Petri net N' after the transformation has been applied.

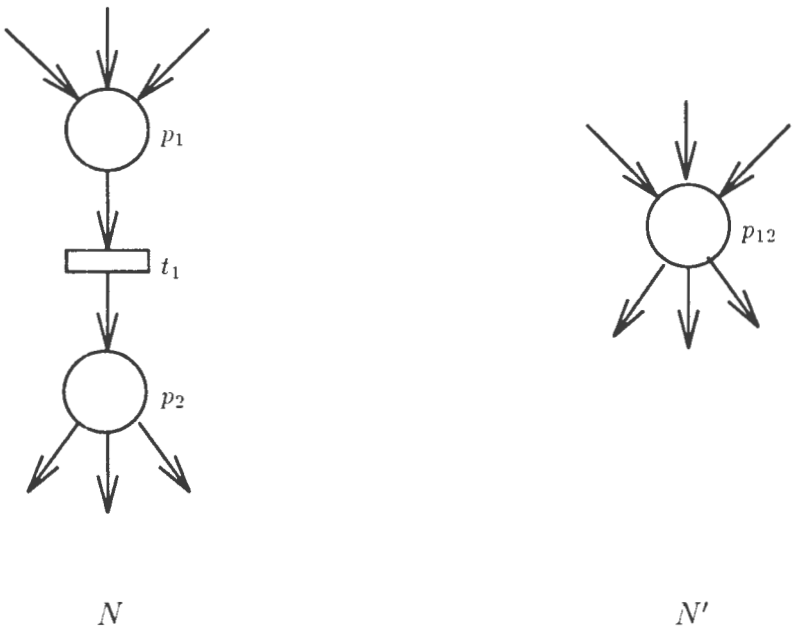


Figure 5.24: Applying the transformation to merge places

Merging Transitions — Rule 2

For a marked free choice net $N = (P, T, B, F, M_0$ if there exists $p_1 \in P$ such that $|\bullet p_1| = |p_1 \bullet| = 1$, $(p_1\bullet)\bullet \neq \emptyset$ and $\bullet(p_1\bullet) = \{p_1\}$ the $t_1, t_2 \in T$ can be merged where $\{t_1\} = \bullet p_1$ and $\{t_2\} = p_1\bullet$ [Esp91].

The result of applying this reduction transformation is a new marked, free choice net $N' = (P', T', B', F', M_0')$ where $P' = P \setminus \{p_1\}$, $T' = (T \setminus \{t_1, t_2\}) \cup \{t_{12}\}$ where t_{12} is a new transition created as a result of merging transitions t_1 and t_2. Further, all inputs to t_1 in N are inputs to t_{12} in N' and all outputs of t_2 in N are outputs of t_{12} in N'. The

new initial marking obtained by adding the initial marking on p_1 to all the output places of t_{12} in N'' [Esp91].

This reduction transformation is also called *fusion of series places* and preserves (in both directions) liveness, safeness and boundedness [Mur89]. This reduction transformation is also a restriction of post-fusion of transitions [Ber86a], and as such all the properties preserved by that transformation are also preserved by this transformation.

Figure 5.25 shows a part of a Petri net N in which the merging of transitions is applicable on the left and on the right is the part of the Petri net N' after the transformation has been applied.

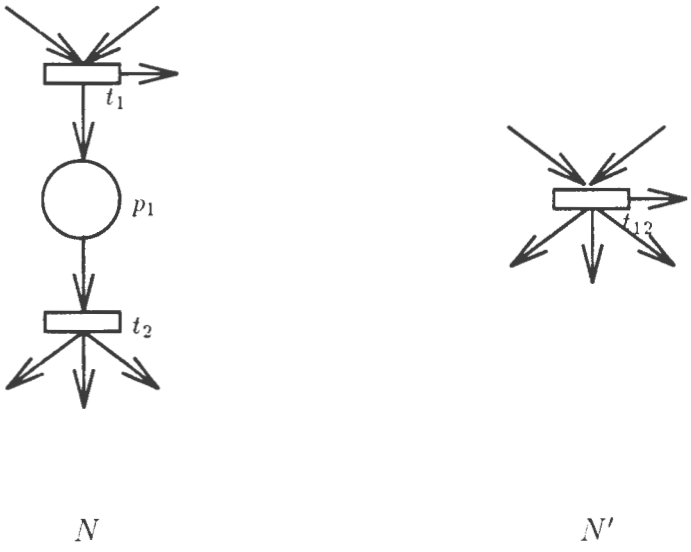


Figure 5.25: Applying the transformation to merge transitions

Linear Dependent Places — Rule 3

Given a marked free choice net $N = (P, T, B, F, M_0)$ in which every P-component is marked at M_0, a place $p_1 \in P$, which is a non-negative linear combination of the other places in P, is called a *linear dependent place* (that is, the row of the incidence matrix $C = F - B$ corresponding to p_1 is a non-negative linear combination of the other rows) [Esp91].

The modified net is obtained by deleting p_1 from the net and removing all arcs incident on p_1. The initial marking is obtained from M_0 by ignoring p_1 [Esp91].

[Esp91] proves that this transformation preserves structural boundedness and the free choice property when applied (in both directions) to the general Petri net. As for the

liveness property, [Esp91] shows that if the Petri net after the transformation is structurally live, then the original Petri net is structurally live, but that the reverse is not true (that is, this rule is not strongly sound with respect to liveness). This means that the rule can be used for reduction but not synthesis if liveness is to be preserved.

This transformation is similar to [Ber86a]'s simplification of redundant places (in the case of total redundancy) in the sense that the application conditions of linear dependent places implies the last application condition of simplification of redundant places (the application condition restricting the incidence matrix).

Figure 5.26 shows a part of a Petri net N in which a place is linear dependent on the left and on the right is the part of the Petri net N' after the transformation has been applied.

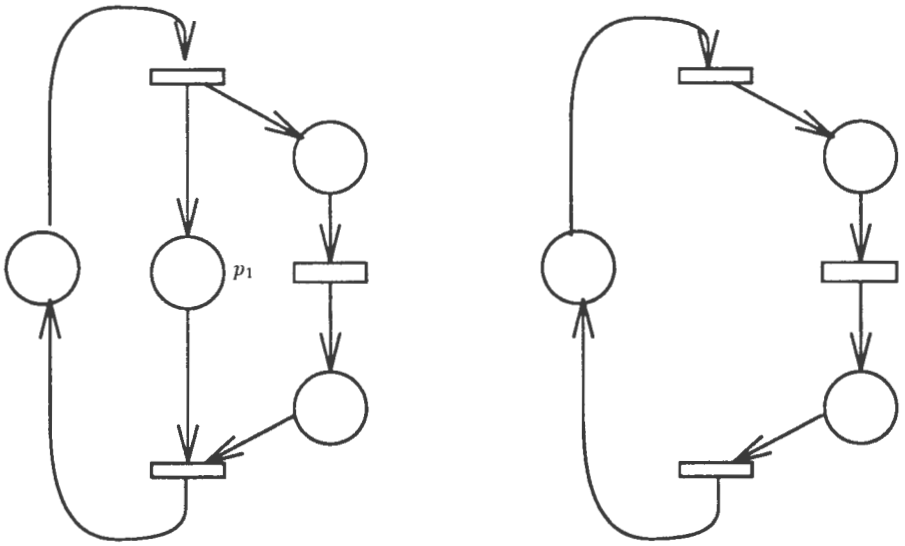


Figure 5.26: Applying the linear dependent places transformation

Linear Dependent Transitions — Rule 4

Given a marked free choice net $N = (P, T, B, F, M_0)$, then a transition $t_1 \in T$ which is a non-negative linear combination of the other transitions in T is a *linear dependent transition* (that is, the column of the incidence matrix $C = F - B$ corresponding to t_1 is a non-negative linear combination of the other transitions) [Esp91].

The modified Petri net is obtained by deleting the transition t_1 from the Petri net N

together with any arcs incident on t_1. The initial marking remains unchanged [Esp91].

[Esp91] shows the soundness of this transformation rule by showing that the structural rules corresponding to removal of linear dependent places and removal of linear dependent transitions are related in the following way: If the structural linear dependent place rule is applicable in a Petri net $N = (P, T, B, F)$ then the structural linear dependent transition rule is applicable in the reverse dual $N^{rd} = (T, P, B^T, F^T)$, and vice versa. Furthermore, the result of applying the structural linear dependent place reduction to a place of a Petri net N, is the same as applying the structural linear dependent transition rule to the corresponding transition of N^{rd}, and then taking the reverse dual. Similarly, for the result of the structural linear dependent transition reduction. With this property [Esp91] was able to show that the linear dependent transition rule applied in either direction preserves together the liveness, boundedness and the free choice property of a Petri net (that is, the reduction rule is strongly sound with respect to live and bounded free choice nets).

Figure 5.27 shows a part of a Petri net N in which a transition is linear dependent on the left and on the right is the part of the Petri net N' after the transformation has been applied.

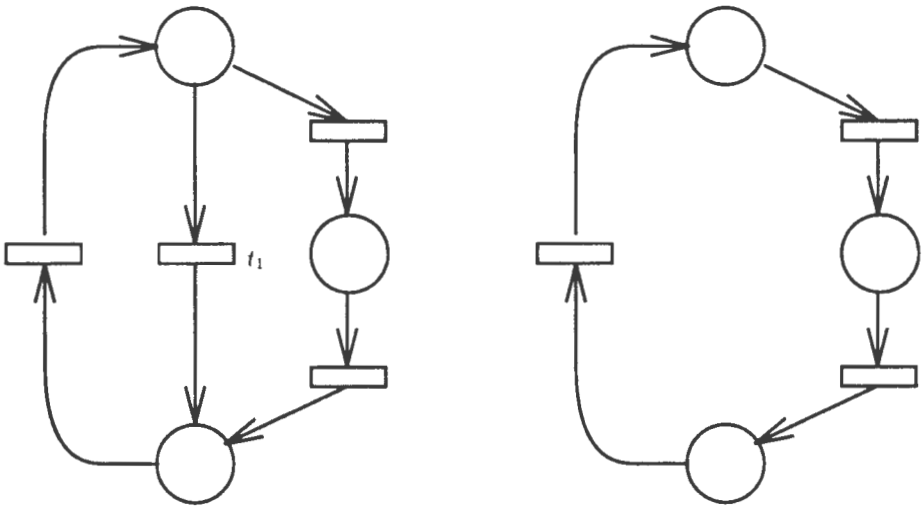


Figure 5.27: Applying the linear dependent transition transformation

Kits to Reduce Live and Bounded Free Choice Nets

[Esp91] showed that the class of Petri nets reduced to an atomic net by the kit of rules comprising Rule 1, Rule 3 and Rule 4 is precisely the class of live and bounded free choice nets. In addition, he showed that the kit of rules comprising Rule 2, Rule 3 and Rule 4 reduces precisely the same class.

Chapter 6

Algorithms and Complexity of Transformations

Algorithms are considered in this chapter as a means of providing complexity bounds for proving the tractability of deciding the applicability of the respective Petri net transformations. Intractability results are based on showing reductions from certain *hard* problems. The tractability of some of the decision problems is proved by illustrating a compatible reduction from a tractable problem. In the case of reduction transformations, it is interesting to consider, in addition to the complexity of deciding the application conditions, the extent of the decreases in the size of the coverability tree which are possible. This makes any analysis requiring the inspection of the coverability tree easier. This is not necessary for synthesis transformations as the generated Petri nets are guaranteed to have the desired properties.

The formal correctness of the algorithms in this chapter has not been established. They have, however, been recoded as C functions and tested on small examples.

6.1 Reduction Transformations

Examples of Petri nets are used to illustrate the possible reductions that can be achieved with reduction transformations. These examples are designed to maximize the reduction in the state space using a particular reduction transformation. Fortunately, it is often the minimum Petri nets satisfying the application conditions that yield the larger reductions in the size of the state space. This is because the part of the net affected by the transformation, and possibly removed, forms a larger proportion of the Petri net for smaller Petri nets than it does for larger ones. The examples used to illustrate the possible reductions

are created by considering such minimal Petri nets satisfying the application conditions. Further, suitable initial markings are chosen to accentuate the reductions in the size of the reachability set.

The *power* of any set of Petri net reduction transformations is limited when used to decide one of the properties liveness or boundedness alone:

Theorem 6.1 *Let $C_{\mathcal{P}}$ be the set of all Petri nets for which liveness (boundedness) can be decided in polynomial-time. No set of reduction transformations which preserve liveness (boundedness) and which can be applied in polynomial-time, can reduce a general Petri net to some member of $C_{\mathcal{P}}$. If $C_{\mathcal{P}}$ is the set of all Petri nets in which liveness and boundedness together can be decided in polynomial-time, then no set of reduction transformations which preserve liveness and boundedness and can be applied in polynomial-time, can reduce a general Petri net to some member of $C_{\mathcal{P}}$.*

Proof For liveness: Assume that there exists such a set of reduction transformations. Since deciding liveness in a member of $C_{\mathcal{P}}$ is in $\mathcal{P}$, this implies an algorithm to decide liveness of any Petri net in polynomial-time. However, this problem is **EXPSPACE**-hard [JLL77].

For boundedness: In this case any such set of reduction transformations would contradict the fact that any algorithm to decide boundedness requires exponential-time [Pet81].

The decision regarding whether boundedness and liveness hold together in a general Petri net is intractable; L pton showed that reachability in bounded Petri nets requires exponential space [Pet81]. Thus, by Theorem 4.17 (page 35) liveness in bounded Petri nets requires exponential space. □

Other properties, in addition to liveness and boundedness, will have to be required to hold in order for the combined properties to be determined in the general Petri net, for example, requiring some structural restriction to hold as well as liveness and boundedness. This, however, amounts to defining a class of Petri nets in which liveness and boundedness analysis can be performed in polynomial-time. For this reason it is important to understand the complexity of deciding the applicability of the transformations, and what reduction in the size of the reachability set can be achieved.

The algorithms in this section assume the environment of the following fragment of a Pascal program:

Algorithm 6.1

```
{ Context assumed by all the algorithms }

program transformation(input, output);

   const lambda           = 0; { used as a null marker }
         first_place      = 1; { first place in given Petri net }
         first_transition = 1; { first transition in given Petri net }
         last_place       =  ; { number of places in given Petri net }
         last_transition  =  ; { number of transitions in the given Petri net }

   type  place_range      = first_place .. last_place;
         place_index      = lambda .. last_place;
         transition_range = first_transition .. last_transition;
         transition_index = lambda .. last_transition;
         naturals         = 0 .. maxint;

         incidence_matrices = array [place_range, transition_range] of naturals;
         markings           = array [place_range] of naturals;


   begin { transformation }

   end. { transformation }
```

6.1.1 Simplification of Redundant Places

In order to show that deciding the application conditions for the transformation that simplifies redundant places (Definition 5.4 on page 41) is tractable, the problem is rearranged to show that it can be stated as an instance of the linear programming problem: Substituting in Inequality (5.3) the value for b from Equation (5.2) gives the matrix inequality $v^T B \leq v^T M_0 e^T$ or

$$v^T(B - M_0 e^T) \leq O^T \tag{6.1}$$

Using Inequalities (6.1) and (5.4), and Equation (5.2) gives an augmented matrix

$$D = [(B - M_0 e^T)|(B - F)|(-M_0)]$$

and the matrix inequality

$$v^T D \leq O^T.$$

Define an m-vector u such that

$$u(p) = \begin{cases} v(p) & \text{if } p = p' \text{ and} \\ -v(p) & \text{if } p \neq p'. \end{cases}$$

and a matrix $A = (a_{ij})$ with the same dimensions as $D = (d_{ij})$, that is $|P| \times (2|T| + 1)$, such that

$$a_{ij} = \begin{cases} d_{ij} & \text{if row } i \text{ corresponds to place } p' \text{ and} \\ -d_{ij} & \text{otherwise.} \end{cases}$$

The problem of deciding whether the place p' is redundant reduces to whether there exists a nonnegative solution to the matrix inequality

$$u^T A \leq O^T \text{ where } u(p) \begin{cases} \geq 1 & \text{if } p = p' \text{ and} \\ \geq 0 & \text{if } p \neq p'. \end{cases} \tag{6.2}$$

Theorem 6.2 *Consider a given Petri net $N = (P, T, B, F, M_0)$. The problem of deciding whether a given place $p' \in P$ is redundant is in $\mathcal{P}$.*

Proof By Theorem 4.12 (page 32) deciding whether there exists a vector u satisfying Inequality (6.2) is in $\mathcal{P}$. □

To find such a vector u to perform the transformation, a linear programming algorithm could be used, specifying a suitable optimization function (see, for example, [NW88]) followed by a scaling of the vector entries to natural numbers. Alternatively, the all integer algorithm of Gomory (see, for example, [Hu69b]) could be used, but, this algorithm, based on the simplex method, is exponential.

It is possible that there might be bounds imposed on the arc-weights of the resultant Petri net. For example, if the resultant Petri net were required to be an ordinary Petri net (only arc-weights of one allowed), or if it is required that redundant places be isolated and removed from the Petri net (arc-weights of zero). By Theorem 4.14 (page 33), deciding whether there exists a u satisfying Equation (5.5) and Condition (5.4) such that for all $t_i \in T$: $w_i \leq b$ for some given $b \in \mathbb{N}$ is $\mathcal{NPC}$.

The functional dependence between the changes of markings of the simplified place and the other places imposed by Equation (5.5) ensures that this transformation does not reduce the size of the state space [Ber86a]. However, it may be possible to perform further reductions because of the simplification of a redundant place.

6.1.2 Fusion of Doubled Places

The following discussion centres around the subproblems concerning the application conditions of doubled places. Whether the whole problem is intractable depends on whether the subproblems are restricted by their inter-dependence.

The application conditions for doubled places (Definition 5.5 on page 43) are a combination of structural and behavioural conditions. Condition (5.6) and Condition (5.7) depend on the initial marking and the structure of the Petri net. Since Condition (5.6) and Condition (5.7) define I_2 and I_2', Condition (5.8) also depends on the initial marking and the structure of the Petri net. Condition (5.9) is similar to the submarking reachability problem (Definition 4.20 on page 35). Finally, Condition (5.10) depends only on the structure of the Petri net and can be checked simply by inspecting the rows of the backward incidence matrix corresponding to the places p and p'.

The following problem concerns Condition (5.9):

Definition 6.1 Mutual Exclusion of Markings (MEM) Problem
Given a Petri net $N = (P, T, B, F, M_0)$ and disjoint subsets of places $Q, Q' \subseteq P$, does there exist a marking $M \in R(M_0)$ such that $\sum_{q \in Q} M(q) \neq 0$ and $\sum_{q \in Q'} M(q) \neq 0$?

Theorem 6.3 *Given a Petri net $N = (P, T, B, F, M_0)$, and subsets $I_2, I_2' \subseteq P$ with $I_2 \cap I_2' = \emptyset$. Deciding whether application Condition (5.9) holds is Co-$\mathcal{NPH}$. In other words, deciding the MEM problem is $\mathcal{NPH}$.*

Figure 6.1 illustrates the construction in the following proof:

Proof Let $J = \bigwedge_{i=1}^{l}(\alpha_1^i \vee \alpha_2^i \vee \alpha_3^i)$ where $\alpha_j^i = x_k$ or $\neg x_k, x_k \in X$ be an instance of the 3-CNF-SAT problem. Construct a Petri net and an instance of (5.9) such that J is satisfiable if and only if Condition (5.9) does not hold for two sets of places I_2 and I_2' in the constructed Petri net. This can be done as follows: Corresponding to each $x_k \in X$ construct a place s_{x_k} each of which is initially marked with one token. Corresponding to each place s_{x_k} construct transitions t_{x_k} and $t_{\neg x_k}$ such that s_{x_k} is an input to both t_{x_k} and $t_{\neg x_k}$ and s_{x_k} is not an input to any other transition. Firing t_{x_k} or $t_{\neg x_k}$ determines an assignment of the variable x_k. This assignment is represented by a token on a place p_{x_k} or $p_{\neg x_k}$, respectively. Therefore corresponding to each transition t_{x_k} construct a place p_{x_k}, initially not marked, such that t_{x_k} is the only input to p_{x_k} and t_{x_k} is not an input to any other place and to each transition $t_{\neg x_k}$ corresponds a place $p_{\neg x_k}$, initially not marked, such that $t_{\neg x_k}$ is the only input to $p_{\neg x_k}$ and $t_{\neg x_k}$ is not an input to any other place. For each of the α_j^i construct a transition $t_{\alpha_j^i}$ such that if $\alpha_j^i = x_k$ for some $x_k \in X$ then $t_{\alpha_j^i}$ has an input from and an output to the place p_{x_k} and if $\alpha_j^i = \neg x_k$ for some $x_k \in X$ then $t_{\alpha_j^i}$ has an input from and an output to the place $p_{\neg x_k}$. Corresponding to each clause $l^i = \alpha_1^i \vee \alpha_2^i \vee \alpha_3^i$ construct a place p_i, initially not marked, such that each place p_i has an input from each of the transitions $t_{\alpha_1^i}$, $t_{\alpha_2^i}$ and $t_{\alpha_3^i}$ and these places have no other inputs. The place p_i represents the value of the clause l^i. The transition t_a as depicted in

Figure 6.1 has as inputs each of the places p_i and as output a single place p_a, initially not marked, which represents the value of the conjunction of all the clauses l^i. To represent the subset I_2', construct an additional place p_b which is isolated and has an initial marking of one token. Figure 6.1 illustrates the construction for the 3-CNF-SAT expression

$$J_0 = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4).$$

Defining $I_2 = \{p_a\}$ and $I_2' = \{p_b\}$ we get an instance of (5.9) and the formula J is satisfiable if and only if Condition (5.9) does not hold for the constructed Petri net. Since 3-CNF-SAT is $\mathcal{NPC}$ this implies that deciding whether Condition (5.9) fails is $\mathcal{NPH}$ and the result follows. □

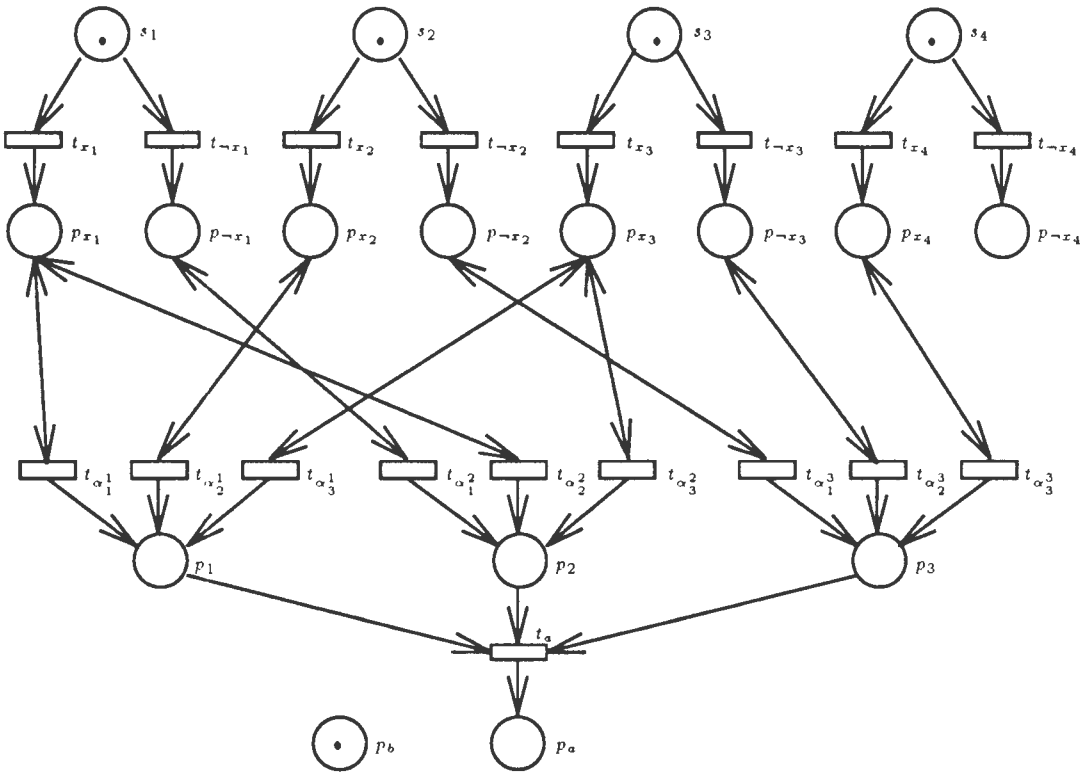


Figure 6.1: Petri net construction from a 3-CNF-SAT instance

It is possible to state a much stronger result about Condition (5.9) and the MEM problem.

Theorem 6.4 *The MEM problem is* **EXPSPACE**-*hard.*

The result follows by showing that the SMC problem can be reduced to the MEM problem in polynomial-time. Figure 6.2 illustrates this construction:

Proof Consider a given instance of the SMC problem; $N = (P, T, B, F, M_0)$, a marking M' and a subset $P' \subseteq P$. Construct a transition t' such that for each $p' \in P'$ there is an arc with weight $M'(p')$ from p' to t'. Construct places q_1 and q_2 such that q_1 has as its only input t' with arc-weight one and q_2 has no inputs. The new place q_1 is initially not marked and the new place q_2 is marked with one token. Neither q_1 nor q_2 have any outputs. With respect to the MEM problem, let $Q = \{q_1\}$ and $Q' = \{q_2\}$. Then the MEM problem holds if and only if there exists a marking $M \in R(M_0)$ such that $M(p) \geq M'(p)$ for all $p \in P'$. Thus SMC $\preceq_{\mathcal{P}}$ MEM and the MEM problem is **EXPSPACE**-hard. □

The above proof could just as well have used a reduction from the coverability problem.

Theorem 6.4 shows that deciding whether Condition (5.9) holds is *Co*-**EXPSPACE**-hard.

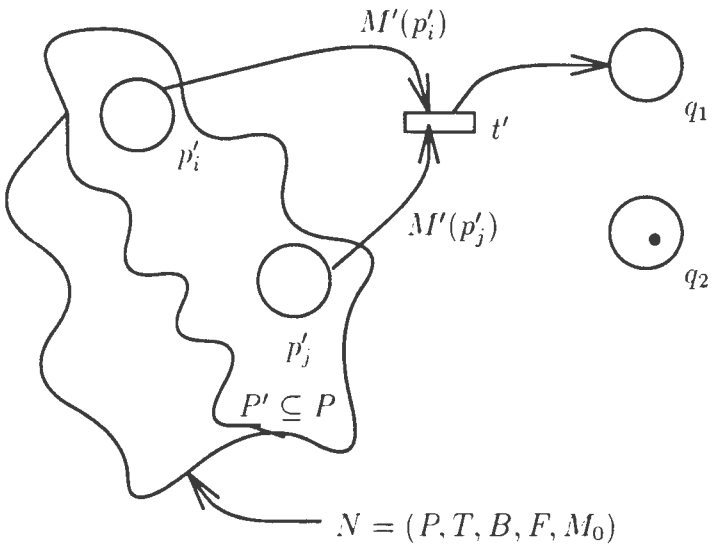


Figure 6.2: Reduction of the SMC problem to the MEM problem

By Theorem 6.4 deciding whether Condition (5.9) alone fails is **EXPSPACE**-hard.

The problem of deciding whether all conditions apply, that is, whether the two places are doubled, can also be shown to be intractable. The following result is used in the proof of the intractablility of deciding whether two places are doubled:

Theorem 6.5 *Consider a Petri net $N = (P, T, B, F, M_0)$ with doubled places $p, p' \in P$. Then for all $M \in R(M_0)$, $M(p) > 0 \Longrightarrow M(p') = 0$ and $M(p') > 0 \Longrightarrow M(p) = 0$.*

Proof By considering the firing rule, that is if $M \xrightarrow{t_i} M'$, then $M' = M + Ce_i$, and, by Condition (5.6), $v^T M_0 \leq 0$ and $v^T(F - B) \leq O^T$, it follows that $v^T M \leq 0$ for all

$M \in R(M_0)$. This can be rewritten

$$\sum_{q \in I_1} v(q)M(q) - \sum_{q \in I_2} v(q)M(q) \leq 0.$$

Assume that $M(p) > 0$ for some marking $M \in R(M_0)$, then $\sum_{q \in I_1} v(q)M(q) > 0$. Hence, $\sum_{q \in I_2} v(q)M(q) < 0$ or $\sum_{q \in I_2} M(q) > 0$. Thus, by Condition (5.9), $\sum_{q \in I_2'} M(q) = 0$. However, it follows from Condition (5.7), that for all $M \in R(M_0)$,

$$\sum_{q \in I_1'} v'(q)M(q) + \sum_{q \in I_2'} v'(q)M(q) \leq 0.$$

Hence $\sum_{q \in I_1'} v'(q)M(q) \leq 0$. However, $v'(q) > 0$ for all $q \in I_1'$, and hence $\sum_{q \in I_1'} M(q) = 0$. Since $p' \in I_1'$, $M(p') = 0$. Similarly, if $M(p') > 0$, then $M(p) = 0$. □

Theorem 6.6 *Consider a given Petri net $N = (P, T, B, F, M_0)$. Deciding whether two places $p, p' \in P$ are doubled is Co-***EXPSPACE***-hard.*

Figure 6.3 illustrates the construction used in the proof showing a reduction from the coverability problem.

Proof Consider a given instance of the coverability problem: A Petri net $N = (P, T, B, F, M_0)$ and a marking M'. Construct a transition t_0 with input arcs from each place $p_i \in P$ with arc-weight $M'(p_i)$, and no other inputs. For the rest of the construction, all the constructed arcs have an arc-weight of one. Construct a place p_0 which is the only output place of transition t_0. Construct transitions t_1 and t_2 which are the only outputs of p_0. Transitions t_1 and t_2 have no other inputs. Construct places p_1 and p_2 which are the only outputs of t_1. Places p_1 and p_2 have no other inputs. Construct transition t_3 which only has inputs from p_1 and p_2. Transition t_3 is the only output of places p_1 and p_2. Construct places p_3 and p_4 which are the only outputs of t_2. Places p_3 and p_4 have no other inputs. Construct transition t_4 which only has inputs from p_3 and p_4. Places p_3 and p_4 have no other outputs. Construct place p_5 which only has as inputs t_3 and t_4. Neither transition t_3 nor transition t_4 have any other outputs. Construct transition t_5 which has as only input p_5. Place p_5 has no other outputs. Construct place p_6 which only has as input t_5. Transition t_5 has no other outputs. Construct transition t_6 which only has as input p_6 and output p_0. Place p_0 has no other inputs and place p_6 has no other outputs. At the initial marking, p_6 is marked with exactly one token, and no other constructed places are marked. With respect to Definition 5.5 let $I_1 = \{p_2\}$, $I_2 = \{p_1\}$, $I_1' = \{p_3\}$, $I_2' = \{p_4\}$, $p = p_2$, $p' = p_3$,

$$v(s) = \begin{cases} 1 & \text{if } s = p_2, \\ -1 & \text{if } s = p_1 \text{ and} \\ 0 & \text{otherwise, and} \end{cases}$$

$$v'(s) = \begin{cases} 1 & \text{if } s = p_3, \\ -1 & \text{if } s = p_4 \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

By considering the initial marking as well as the backward and forward incidence matrices of the Petri net N with the additional places and transitions, it is easy to verify that v satisfies Condition (5.6), v' satisfies Condition (5.7), $p = p_2$ and $p' = p_3$ satisfy Condition (5.10). Since $I_2 \cap I_2' = \emptyset$, p_2 and p_3 are doubled if and only if Condition (5.9) is satisfied. If t_0 does not ever fire then Condition (5.9) is satisfied. However, if t_0 can fire then there exists a marking in the reachability set of the Petri net N augmented with the construction such that places p_2 and p_3 are simultaneously marked, and hence by Theorem 6.5 places p_2 and p_3 cannot be doubled. Transition t_0 will fire if and only if there exists a marking M, covering M', in the reachability set of the original Petri net N. Thus, places p_2 and p_3 are doubled if and only if the given instance of the coverability problem has no solution. The result follows from Theorem 4.19 (page 36). □

The fusion of doubled places does not change the size of the reachability set. To see this, one only has to realise that the combined place p'' has a dual role, that of the doubled places p and p'. These roles cannot be confused in N'. By Condition (5.10), p and p' do not share output transitions. Thus, the fact that the roles of p'' cannot be confused follows from Condition (5.9) and the fact that p and p' cannot be simultaneously marked in all markings reachable by N (Theorem 6.5).

6.1.3 Fusion of Equivalent Places

Algorithm 6.2 finds all pairs of equivalent places (Definition 5.6 on page 44) in a given Petri net:

Algorithm 6.2

```
{ Algorithm to find all pairs of equivalent places in a given Petri net }

procedure find_equiv(B, F : incidence_matrices);

   var p1, p2, p3 : place_range;
       t1, t2, t3 : transition_range;
       p1nz, p2nz : boolean;
       equivalent : array [ place_range, place_range ] of boolean;

   begin { find_equiv }

      for p1 := first_place to last_place do
```

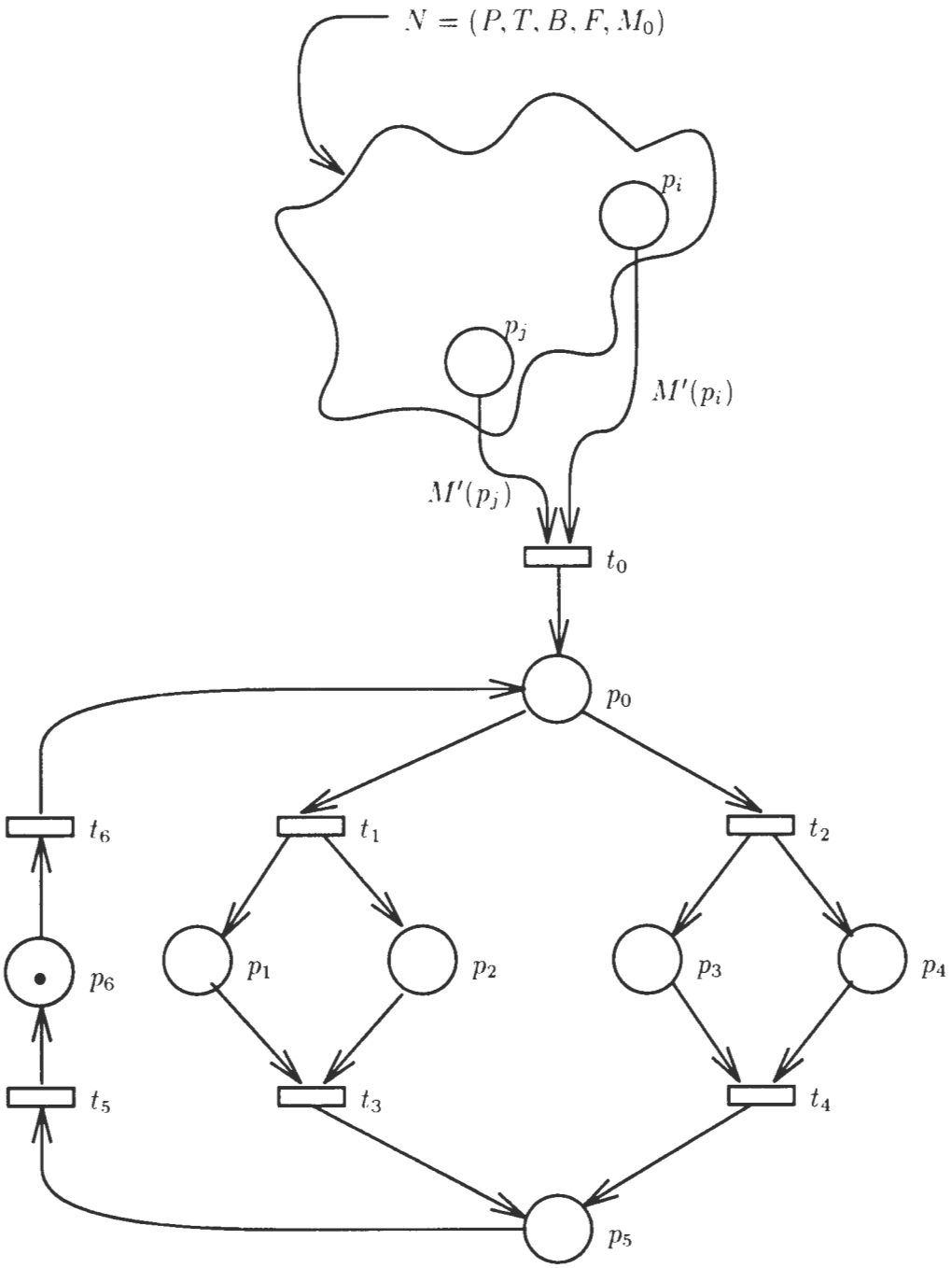


Figure 6.3: Reduction of the coverability problem to doubled places

```
      for p2 := first_place to last_place do
         if p1 = p2 then equivalent[p1, p2] := false
         else begin
            equivalent[p1, p2] := false;
            for t1 := first_transition to last_transition do
               for t2 := t1+1 to last_transition do
                  if not equivalent[p1, p2] then begin
                     equivalent[p1, p2] := (B[p1, t1] = 1) and
                                           (B[p2, t2] = 1) and
                                           (B[p2, t1] = 0) and
                                           (B[p1, t2] = 0);
                     if equivalent[p1, p2] then
                        for t3 : = first_transition to last_transition do
                           if (t3 <> t1) and (t3 <> t2) and
                              ((B[p1, t3] <> 0) or
                               (B[p2, t3] <> 0)) then
                              equivalent[p1, p2] := false;
                     if equivalent[p1, p2] then
                        for p3 := first_place to last_place do
                           if (p3 <> p1) and (p3 <> p2) and
                              ((B[p3, t1] <> B[p3, t2]) or
                               (F[p3, t1] <> F[p3, t2])) then
                              equivalent[p1, p2] := false
                     if equivalent[p1, p2] then begin
                        p1nz := false; p2nz := false;
                        for t3 : = first_transition to last_transition do
                           begin
                           if F[p1, t3] <> 0 then p1nz := true;
                           if F[p2, t3] <> 0 then p2nz := true
                           end;
                        equivalent[p1, p2] := p1nz and p2nz
                        end
                     end
               end

   end; { find_equiv }
```

Counting the maximum number of times the loops in Algorithm 6.2 can be executed in terms of the Petri net $N = (P, T, B, F)$, the complexity of finding all pairs of redundant places is

$$\begin{aligned} T(N) &= O(|P|^2|T|^2(|P| + 2|T|)) \\ &= O(|P|^3|T|^2 + 2|P|^2|T|^3). \end{aligned} \tag{6.3}$$

If N is encoded such that $|\langle N \rangle| = n$, then in this case the input only contains the backward and forward incidence matrices, and the values of $|P|$ and $|T|$ are largest when the entries of B and F occupy the minimum number of bits, that is, one bit each. This gives $|P||T| = O(n)$ irrespective of the actual dimensions of B and F. If the term with the highest power in $T(N)$ has the same power for $|P|$ and $|T|$, then $T(n)$ can be found by substituting the product $|P||T|$ for n in this term and ignoring all the other terms. If the term with

the highest power in $T(N)$ is not the same for $|P|$ and $|T|$, then the largest value of $T(N)$ is realised when the factor with lower power is taken to be some small constant. This constant can usually be taken to be one, but some of the transformations specify a minimum number of places or transitions greater than one. In Equation (6.3) there is a different factor with a highest power in each of the terms and these powers are equal. In this case it does not matter which factor is chosen to be constant. The largest value of $T(N)$ is realised when the term with the largest coefficient has its lower power factor replaced by some constant k. The factor with the highest power is then $O(n \div (2k))$. From Equation (6.3), the complexity in terms of the input length n is

$$\begin{aligned} T(n) &= O(k^3(\frac{n}{2k})^2 + 2k^2(\frac{n}{2k})^3) \\ &= O(n^3). \end{aligned}$$

Notice that even if M_0 were encoded in the input, then at worst this amounts to $2|T| + 1$ rows of $|P|$ bits, rather than $2|T|$ rows of $|P|$ bits, and so does not change the asymptotic complexities.

To illustrate the reduction in the size of the reachability tree that is possible by fusion of equivalent places, consider the Petri net $N = (P, T, B, F, M_0)$ where $P = \{p_0, p_1, p_2\}$, $T = \{t_0, t_1, t_2\}$,

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

$$F = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

and $M_0{}^T = [k, 0, 0]$. In this Petri net the places p_1 and p_2 are equivalent and the result of fusion is the Petri net $N' = (P', T', B', F', M_0')$ where $P' = \{p_0, p_1\}$, $T' = \{t_0, t_1\}$,

$$B' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix},$$

$$F' = \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

and $M_0'{}^T = [k, 0]$. By generating the reachability tree for different values of k, the reachability set size of N can be verified to be

$$S_N(k) = |R_N(M_0)| = \frac{2(k+1)^3 + 3(k+1)^2 + k + 1}{6}$$

and that of N' is

$$S_{N'}(k) = |R_{N'}(M'_0)| = (k+1)^2.$$

By considering the ratio

$$\begin{aligned} \frac{S_{N'}(k-1)}{S_N(k-1)} &= \frac{6k^2}{2k^3+3k^2+k} \\ &= \frac{3}{k+\frac{3}{2}+\frac{1}{2k}} \end{aligned}$$

one sees that for large k the size of the reachability set is reduced by a factor of $k/3 = O(k)$ after fusion of the equivalent places.

6.1.4 Post-Fusion of Transitions

Consider a given Petri net $N = (P, T, B, F)$. Algorithm 6.3 finds all possible candidates for the place $p \in P$ and the subsets $T_F, T_H \subseteq T$ that satisfy Condition (5.11) (Definition 5.7 on page 46):

Algorithm 6.3

```
{ Algorithm to find all possible candidate sets for structural
  post-fusion of transitions and their corresponding place. }

procedure post_fusion_candidates(B, F : incidence_matrices);

   type candidates = record
                        T_F : set of transition_range;
                        T_H : set of transition_range;
                     end;

   var p         : place_range;
       t         : transition_range;
       candidate : array [ place_range ] of candidates;
       skipping  : boolean;

   begin { post_fusion_candidates }

      for p := first_place to last_place do begin
         candidate[p].T_F := [];
         candidate[p].T_H := [];
         for t := first_transition to last_transition do begin
            if t = first_transition then skipping := false;
            if not skipping then
               if (B[p, t] > 0) and  (F[p, t] = 0) then
                  candidate[p].T_F := candidate[p].T_F + [t]
               else
```

```
                if (B[p, t] = 0) and (F[p, t] > 0) then
                   candidate[p].T_H := candidate[p].T_H + [t]
                else
                   if (B[p, t] > 0) and (F[p ,t] > 0) then
                      skipping := true
             end
          end

   end; { post_fusion_candidates }
```

Once the candidate sets of transitions have been found, they can be verified using Algorithm 6.4 which checks whether the given sets of transitions are post-fusible with respect to the given place:

Algorithm 6.4

```
{ Algorithm to test whether given sets of transitions are post-fusible
  with respect to a given place. }

procedure post_fusion_test(B, F     : incidence_matrices;
                           T_F, T_H : set of transition_range;
                           p        : place_range);

   var q            : place_range;
       t, tg        : transition_range;
       post_fusible : boolean;
       positive     : boolean;

   begin { post_fusion_test }

      if T_F <> [] then begin
         tg := first_transition;
         post_fusible := true;
         positive := false;
         for t := first_transition to last_transition do
            if (not (tg in T_F)) and (t in T_F) then tg := t;
         for q := first_place to last_place do
            for t := first_transition to last_transition do
               if (t in T_F) and (q = p) and post_fusible then
                  post_fusible := (B[q, t] = B[p, tg]) and
                                  (F[q, t] = 0)
               else
                  if (not (t in T_F)) and (q = p) and post_fusible then
                     post_fusible := (B[q, t] = 0) and
                                     (F[q, t] mod B[p, tg] = 0)
                  else
                     if (t in T_F) and (q <> p) and post_fusible then begin
                        post_fusible := (B[q, t] = 0);
                        if F[q, t] > 0 then positive := true
                        end;
         post_fusible := post_fusible and positive
```

```
        end
     else
        post_fusible := false

end; { post_fusion_test }
```

In terms of the Petri net N, the complexity of Algorithm 6.3 is

$$T(N) = O(|P||T|)$$

and that of Algorithm 6.4 is

$$T(N) = O(|T| + |P||T|).$$

Thus the complexity of finding all sets of post-fusible transitions is

$$\begin{aligned} T(N) &= O(|P||T|(|T| + |P||T|)) \\ &= O(|P|^2|T|^2) \end{aligned} \tag{6.4}$$

If N is encoded such that $|\langle N \rangle| = n$, then since $|P||T| = O(n)$, Equation (6.4) gives the complexity in terms of the encoding length as

$$T(n) = O(n^2).$$

Structural post-fusion of transitions can give even greater reductions in the size of a Petri net's reachability set than is possible in the example of fusion of equivalent places. Let $N = (P, T, B, F, M_0)$ be a Petri net with $P = \{p_0, p_1\}$, $T = \{t_0, t_1\}$,

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

$$F = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

and ${M_0}^T = [0, k]$. If $T_F = \{t_0\}$, $T_H = \{t_1\}$ and $p = p_0$ then T_F and T_H are post-fusible and the resultant Petri net is $N' = (P', T', B', F', M_0')$ after fusion where $P' = \{p_0, p_1\}$, $T' = \{t_0, t_1\}$,

$$B' = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix},$$

and

$$F' = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

and $M_0'^T = [0, k]$. Actually, p_0 and t_1 can be removed as they are isolated. The reachability set size of N is

$$S_N(k) = |R_N(M_0)| = k + 1$$

and that for N' is

$$S_{N'}(k) = |R_{N'}(M_0')| = 1.$$

Hence, no matter what the size of the reachability set, the fusion of post-fusible transitions reduces this example to a trivial reachability set.

6.1.5 Pre-Fusion of Transitions

Algorithm 6.5 finds all candidates $h \in T$, $T_F \subseteq T$ and $p \in P$ that satisfy Conditions (5.12), (5.13), (5.14), (5.15) and (5.16) (Definition 5.8 on page 49) such that T_F is pre-fusible with h with respect to p:

Algorithm 6.5

```
{ Algorithm to find all possible candidate subsets of transitions
  with corresponding place and transition such that the transition
  and set of places are pre-fusible with respect to the place. }

procedure pre_fusion_candidates(B, F : incidence_matrices;
                                M0   : markings);

   type candidates = record
                        T_F : set of transition_range;
                        h   : transition_range;
                     end;

   var p         : place_range;
       t         : transition_range;
       candidate : array [ place_range ] of candidates;
       skipping  : boolean;

   begin { pre_fusion_candidates }

      for p := first_place to last_place do begin
```

```
         skipping := false;
         candidate[p].T_F := [];
         if M0[p] = 0 then
            for t := first_transition to last_transition do
               if not skipping then
                  if (B[p, t] = 1) and (F[p, t] = 0) then
                     candidate[p].T_F := candidate[p].T_F + [t]
                  else
                     if (B[p, t] = 0) and (F[p, t] = 1) then
                        candidate[p].h := t
                     else
                        if (B[p, t] <> 0) or (F[p, t] <> 0) then begin
                           candidate[p].T_F := [];
                           skipping := true
                           end
         end

   end; { pre_fusion_candidates }
```

After having collected all possible candidates for pre-fusion, Algorithm 6.6 verifies that pre-fusion can indeed take place:

Algorithm 6.6

```
{ Algorithm to test whether a given set of transitions are post-fusible
  with a given transition with respect to a given place. }

procedure pre_fusion_test(B, F : incidence_matrices;
                          T_F, : set of transition_range;
                          h    : transition_range;
                          p    : place_range);

   var q           : place_range;
       t           : transition_range;
       pre_fusible : boolean;

   begin { pre_fusion_test }

      if T_F <> [] then begin
         post_fusible := true;
         for q := first_place to last_place do
            if q <> p then begin
               if B[q, h] > 0 then
                  for t := first_transition to last_transition do
                     if t <> h then
                        if B[q, t] <> 0 then
                           pre_fusible := false;
               if F[q, h] <> 0 then
                  pre_fusible := false
               end
         end

   end; { pre_fusion_test }
```

In terms of a Petri net $N = (P, T, B, F, M_0)$, Algorithm 6.5 has a time complexity

$$T(N) = O(|P||T|)$$

and Algorithm 6.6 has a time complexity of

$$T(N) = O(|P||T|).$$

This gives the complexity of finding all such sets of pre-fusible transitions as

$$T(N) = O(|P|^2|T|^2). \tag{6.5}$$

Equation (6.5) gives the complexity in terms of the length of the input encoding $n = |\langle N \rangle|$ as

$$T(n) = O(n^2).$$

The example used in Section 6.1.4 can also be used to illustrate the size of reductions in the reachability set size that are achievable. The same resultant Petri net is possible by the pre-fusion of transitions if $p = p_0$, $T_F = \{t_0\}$ and $h = t_1$.

6.1.6 Lateral Fusion of Transitions

Algorithm 6.7 finds all candidate transitions t_1, t_2 and t_c and places p_1 and p_2 satisfying Conditions (5.17), (5.18), (5.19), (5.20), (5.21) and (5.23) (Definition 5.9 on page 51):

Algorithm 6.7

```
{ Algorithm to find all possible candidate transitions for
  lateral fusion and the transition and places with
  respect to which they are laterally fusible. }

procedure lateral_fusion_candidates(B, F : incidence_matrices;
                                    M0   : markings);

   type candidates = record
                        t1, t2, tc : transition_index;
                     end;

   var pi, pj     : place_range;
       t          : transition_index;
       candidate  : array [ place_range, place_range ] of candidates;
       skipping   : boolean;
```

```
   begin { lateral_fusion_candidates }

      for pi := first_place to last_place do
         for pj := first_place to pred(pj) do
            if M0[pi] = M0[pj] then begin
               candidate[pi, pj].tc := lambda;
               candidate[pi, pj].t1 := lambda;
               candidate[pi, pj].t2 := lambda;
               skipping := false;
               for t := first_transition to last_transition do
                  if not skipping then
                     if (B[pi, t] = 1) and (B[pj, t] = 0) and
                        (F[pi, t] = 0) and (F[pj, t] = 0) and
                        (candidate[pi, pj].t1 = lambda) then
                        candidate[pi, pj].t1 := t
                     else
                        if (B[pi, t] = 0) and (B[pj, t] = 1) and
                           (F[pi, t] = 0) and (F[pj, t] = 0) and
                           (candidate[pi, pj].t2 = lambda) then
                           candidate[pi, pj].t2 := t
                        else
                           if (B[pi, t] = 0) and (B[pj, t] = 0) and
                              (F[pi, t] = 1) and (F[pj, t] = 1) and
                              (candidate[pi, pj].tc = lambda) then
                              candidate[pi, pj].tc := t
                           else
                              if (B[pi, t] <> 0) or (B[pj, t] <> 0) or
                                 (F[pi, t] <> 0) or (F[pj, t] <> 0) then
                                 skipping := true
       end

   end; { lateral_fusion_candidates }
```

Once all possible candidates for lateral fusion have been found they can be verified using Algorithm 6.8 which checks that Conditions (5.22) and (5.24) are satisfied:

Algorithm 6.8

```
{ Algorithm to test whether a given pair of transitions
  are latterally fusible with respect to a given pair
  of places and a given transition. }

procedure lateral_fusion_test(B, F       : incidence_matrices;
                              p1, p2     : place_range;
                              t1, t2, tc : transition_index);

      var p                : place_range;
          t                : transition_range;
          laterally_fusible : boolean;
          u, v, w, x, y    : naturals;

   begin { lateral_fusion_test }
```

```
      if (t1 <> lambda) and (t2 <> lambda) and (tc <> lambda) then begin
         laterally_fusible := true;
         u := 0; v :=0; x := 0; y := 0;
         for p := first_place to last_place do
            if (p <> p1) and (p <> p2) then begin
               u := u + B[p, t1];
               v := v + B[p, t2];
               x := x + F[p, t1];
               y := y + F[p, t2];
               w := 0;
               for t := first_transition to last_transition do
                  if (t <> t1) and (t <> t2) and (t <> tc) then
                     w := w + B[pi, t];
               laterally_fusible := laterally_fusible and
                  (((B[p, t1] <> 0) and (B[p, t2] <> 0) and (w <>0)) or
                   (B[p, t1] = 0)) and
                  (((B[p, t2] <> 0) and (B[p, t1] <> 0) and (w <>0)) or
                   (B[p, t2] = 0))
               end;
         laterally_fusible := laterally_fusible and
            (((x <> 0) and (v = 0)) or (x = 0)) and
            (((y <> 0) and (u = 0)) or (y = 0))

   end; { lateral_fusion_test }
```

If N is a Petri net then the complexity of Algorithm 6.7 is

$$T(N) = O(|P|^2|T|)$$

and that of Algorithm 6.8 is

$$T(N) = O(|P||T|).$$

The complexity of finding all pairs of laterally fusible transitions, the pair of places and the transition with respect to which they are laterally fusible is, thus,

$$T(N) = O(|P|^3|T|^2). \tag{6.6}$$

If N is encoded such that $|\langle N \rangle| = n$, then the greatest value for Equation (6.6) will be realised when there are a constant number of transitions ($|T| = k$):

$$\begin{aligned} T(n) &= O((\frac{n}{2k+1})^3 k^2) \\ &= O(n^3). \end{aligned}$$

Algorithms 6.7 and 6.8 test for lateral fusion of transitions with respect to part one of Definition 5.9. Since part two of Definition 5.9 is merely a reversal of the roles of B and F, the complexity of lateral fusion of transitions with respect to part two is also $O(n^3)$.

To illustrate the possible reduction in the size of the reachability set, consider the Petri net $N = (P, T, B, F, M_0)$ where $P = \{p_1, p_2, p_3, p_4\}$, $T = \{t_1, t_2, t_c\}$,

$$B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix},$$

$$F = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

and $M_0{}^T = [0, 0, k, k]$. Transitions t_1 and t_2 are laterally fusible with respect to t_c, p_1 and p_2. N is not a minimal Petri net containing laterally fusible transitions (only 3 places are required in a minimal Petri net), however this 4 place Petri net simplifies the combinatorical argument for the size of the reachability set. Fusion of t_1 and t_2 results in the Petri net $N' = (P', T', B', F', M_0')$ where $P' = P$, $T' = \{t_1, t_c\}$,

$$B' = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix},$$

$$F' = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

and $M_0'{}^T = [0, k, k]$. The reachability set size of N is

$$S_N(k) = |R_N(M_0)| = (k+1)^2$$

and that for N' is

$$S_{N'}(k) = |R_{N'}(M_0')| = k + 1.$$

Hence the size of the reachability set is reduced by a factor $k + 1$ after lateral fusion of transitions t_1 and t_2.

6.2 Synthesis Transformations

When reducing a Petri net using reduction transformations the goal is always quite clear; to reduce the size of the Petri net and, possibly, the size of the reachability/coverability tree. This means that any transformation can be applied to any part of the Petri net as long as progress is made toward this goal. When synthesizing Petri nets using synthesis transformations the goal is not as easy to establish. The goal is not only to refine a given Petri net by applying a synthesis transformation to some arbitrary part of the Petri net, but that the resultant Petri net model a given system.

The view taken in this thesis is that of the engineer using Petri nets to model a system. Using synthesis transformations the modeller builds subcomponents and "attempts" to either refine or add additional subnets to them. Whichever the case it is the engineer who completely specifies what transformation is to be applied and what elements are involved. Thus, in this thesis it is only the complexity of deciding the applicability of particular synthesis transformations, with respect to given subcomponents of a Petri net, that are studied.

Synthesis transformations which guarantee that certain properties hold in the resultant system do not need any analysis to determine whether such properties hold. For this reason the effect on the size of the reachability/coverability tree is not considered for synthesis transformations.

6.2.1 Addition of a Derivable Subnet

Let $N = (P, T, B, F, M_0)$ be a Petri net to which a subnet $N_s = (P_s, T_s, B_s, F_s, M_{0s})$ is to be added as a derivable subnet (Definition 5.10 on page 54):

Theorem 6.7 *Deciding whether a given subnet of a given Petri net is derivable is Co-***EXPSPACE***-hard.*

The proof is by reduction from the coverability problem. Figure 6.4 illustrates the construction used in the proof:

Proof Given an instance of the coverability problem, $N = (P, T, B, F, M_0)$ and a marking M', construct a transition t_0 with inputs from all places $p \in P$ with arc-weights $M'(p)$. Construct places p_1, p_2 and p_3, and transitions t_1 and t_2 such that p_1 has as only input the transition t_0 with arc-weight 2 and as only output the transition t_1 with arc-weight one. Place p_2 has as only input transition t_1 and as only output transition t_2, both with

arc-weights of one. Place p_3 has as only input transition t_2 and as only output transition t_3, both with arc-weights of one. Initially the places p_1, p_2 and p_3 are not marked. Denote by $N_s = (P_s, T_s, B_s, F_s, M_{0s})$ the subnet where $P_s = \{p_1, p_2, p_3\}$, $T_s = \{t_1, t_2\}$,

$$B_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix},$$

$$F_s = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and ${M_{0s}}^T = [0, 0, 0]$ It is obvious N_s satisfies parts 1, 2, 3, 4, 6, 7 and 8 of Definition 5.10. Thus N_s is a derivable subnet if and only if part 5 of Definition 5.10 holds, that is, if and only if N_s is not re-entrant. Transition t_0 can only fire if the marking M' is coverable in the original Petri net, and if it does, it will deposit two tokens in N_s in which case N_s is re-entrant. If t_0 cannot fire then N_s will never have any tokens placed in it. The result follows since the coverability problem is **EXPSPACE**-hard (Theorem 4.19 on page 36).

□

The complexity of each of the parts of Definition 5.10 may be important if the hard questions have been answered by, for example, the modeller building a Petri net.

Algorithm 6.9 tests in linear time whether a given subnet is open, that is $T(N, N_s) = O(|P||T|)$:

Algorithm 6.9

```
{ Algorithm to test whether a given subnet is open. }

procedure test_open(B, F : incidence_matrices;
                    PS   : set of place_range;
                    TS   : set of transition_range;

   var p    : place_range;
       t    : transition_range;
       open : boolean;

   begin { test_open }

      open := true;
      for t := first_transition to last_transition do
         if t in TS then
            for p := first_place to last_place do
```

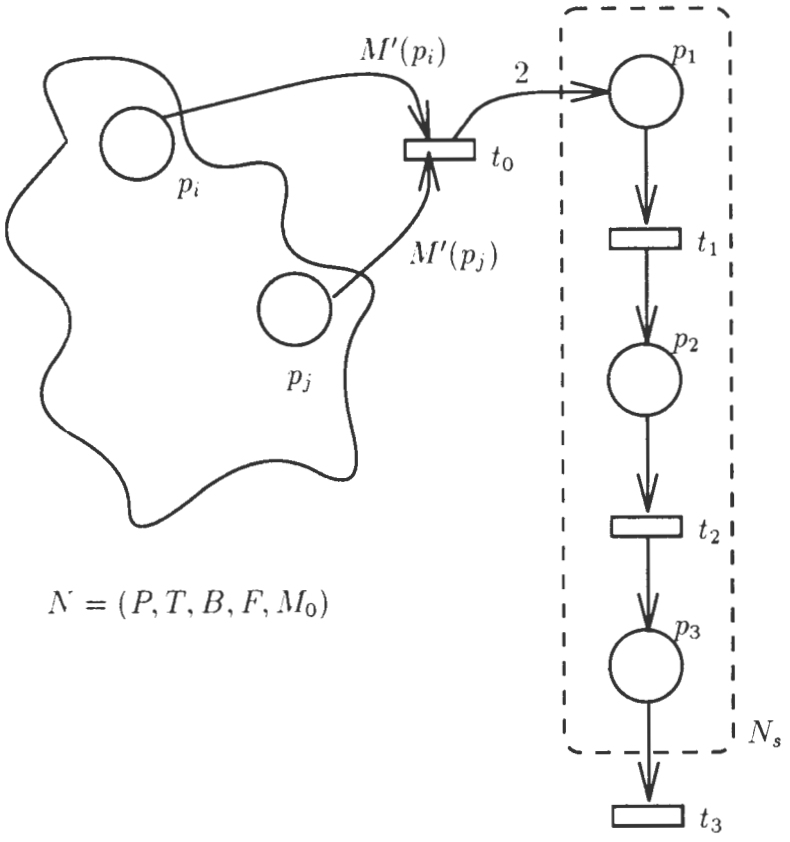


Figure 6.4: Reduction of coverability to the derivable subnet problem

```
            open := open and ((B[p, t] <> 0) or (F[p, t] <> 0))
                         and (p in PS);


   end; { test_open }
```

Algorithm 6.10 tests whether the given subnet is a marked graph with $T(N, N_s) = O(|P||T|)$:

Algorithm 6.10

```
{ Algortihm to test whether a given subnet is a marked graph. }

procedure test_marked_graph(B, F : incidence_matrices;
                            PS   : set of place_range;
                            TS   : set of transition_range);

   var p            : place_range;
       t            : transition_range;
       inputs       : naturals;
       outputs      : naturals;
       marked_graph : boolean;

   begin { test_marked_graph }

      marked_graph := true;
      for p := first_place to last_place do
         if p in PS then begin
            inputs := 0;
            outputs := 0;
            for t := first_transition to last_transition do
               if t in TS then begin
                  inputs := inputs + F[p, t];
                  outputs := outputs + B[p, t]
                  end;
            marked_graph := marked_graph and (inputs <= 1)
                                         and (outputs <= 1)

   end; { test_marked_graph }
```

Part 3 of Definition 5.10 can be decided in polynomial-time as follows: The depth first search algorithm (DFS) (described in, for example, [CLR90]) can be used to classify all edges of the subnet as either *tree* edges, *back* edges or *cross* edges. Since back edges appear if and only if there are cycles in the graph, the DFS algorithm can decide whether or not cycles exist in polynomial-time (the complexity of DFS for a graph $G = (V, E)$ is $O(|V| + |E|)$ assuming adjacency lists for representing the graph, and in terms of the data structures assumed in this thesis $O(|V|^2)$ — since $|E| \leq |V|(|V| - 1)$). Furthermore,

N_s is connected if and only if all the transitions and places of (or vertices of the graph $G = (V, E)$) are "discovered" by the DFS algorithm [CLR90].

Part 4 of Definition 5.10 can be verified using a topological sort of N_s, this can be accomplished in linear time [CLR90].

Part 6 of Definition 5.10, the partitioning of the places of P_s, can be accomplished and verified by Algorithm 6.11 with time complexity $T(N, N_s) = O(|P||T|)$:

Algorithm 6.11

```
{ Algorithm to test if places of subnet can be partitioned
  as per a deriveable subnet. }

procedure test_partition(B, F : incidence_matrices;
                         PS   : set of place_range;
                         TS   : set of transition_range);

   var p         : place_range;
       t         : transition_range;
       P_E       : set of place_range;
       P_S       : set of place_range;
       P_I       : set of place_range;
       inputs    : set of place_range;
       outputs   : set of place_range;
       partition : boolean;

   begin { test_partition }

      inputs := []; outputs := [];
      P_E := PS; P_S := PS;
      for p := first_place to last_place do
         if p in PS then
            for t := first_transition to last_transition do
               if t in TS then begin
                  if F[p, t] <> 0 then begin
                     P_E := P_E - [p];
                     outputs := outputs + [p]
                     end;
                  if B[p, t] <> 0 then begin
                     P_S := P_S - [p];
                     inputs := inputs + [p]
                     end
                  end;
      P_I := inputs * outputs;
      partition := (PS = P_E + P_S + P_I) and
                   (P_I * P_E = []) and
                   (P_I * P_S = []) and
                   (P_E * P_S)

   end; { test_partition }
```

Theorem 6.8 proves that testing part 7 is intractable:

Theorem 6.8 *Given a subnet $N_s = (P_s, T_s, B_s, F_s, M_{0s})$ of a Petri net $N = (P, T, B, F, M_0)$, the problem of whether N_s can be emptied by firing transitions in T_s and non-conflicting transitions of T is* **EXPSPACE**-*hard.*

This proof is similar to the proof of Theorem 6.7. Figure 6.5 illustrates the construction used in the proof:

Proof Consider a given instance of the coverability problem: $N = (P, T, B, F, M_0)$ and a marking M'. Construct the additional places and transitions as for the proof of Theorem 6.7, except that the arc-weight from transition t_0 to place p_1 is now only one. Construct a place p_4 initially marked with one token, with no inputs and only one output to transition t_1 with arc-weight one. N_s is thus not empty. Construct place p_5 which has as input only the transition t_0 with arc-weight one and as output only the transition t_3 also with arc-weight one. Construct a place p_6 initially marked with one token, with no inputs and only one output to transition t_0 with arc-weight one. $N_s = (P_s, T_s, B_s, F_s, M_{0s})$ where $P_s = \{p_1, p_2, p_3, p_4\}$, $T_s = \{t_1, t_2\}$,

$$B_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 1 & 0 \end{bmatrix},$$

$$F_s = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

and ${M_{0s}}^T = [0, 0, 0, 0]$ is a subnet of the Petri net augmented with the constructed places, transitions, arcs and initial markings. Since the token on place p_1 can be removed from the subnet N_s if and only if transition t_0 fires, it follows that part 7 holds if and only if there exists a marking $M \in R(M_0)$ such that $M \geq M'$ and the result follows. □

The construction of the subnet N_s in Theorem 6.8 is such that all the conditions except part 7 of Definition 5.10 hold. Thus the proof of Theorem 6.8 also proves the intractability of the entire problem, and proves the following theorem:

Theorem 6.9 *Deciding whether a given subnet of a given Petri net is derivable is* **EXPSPACE**-*hard.*

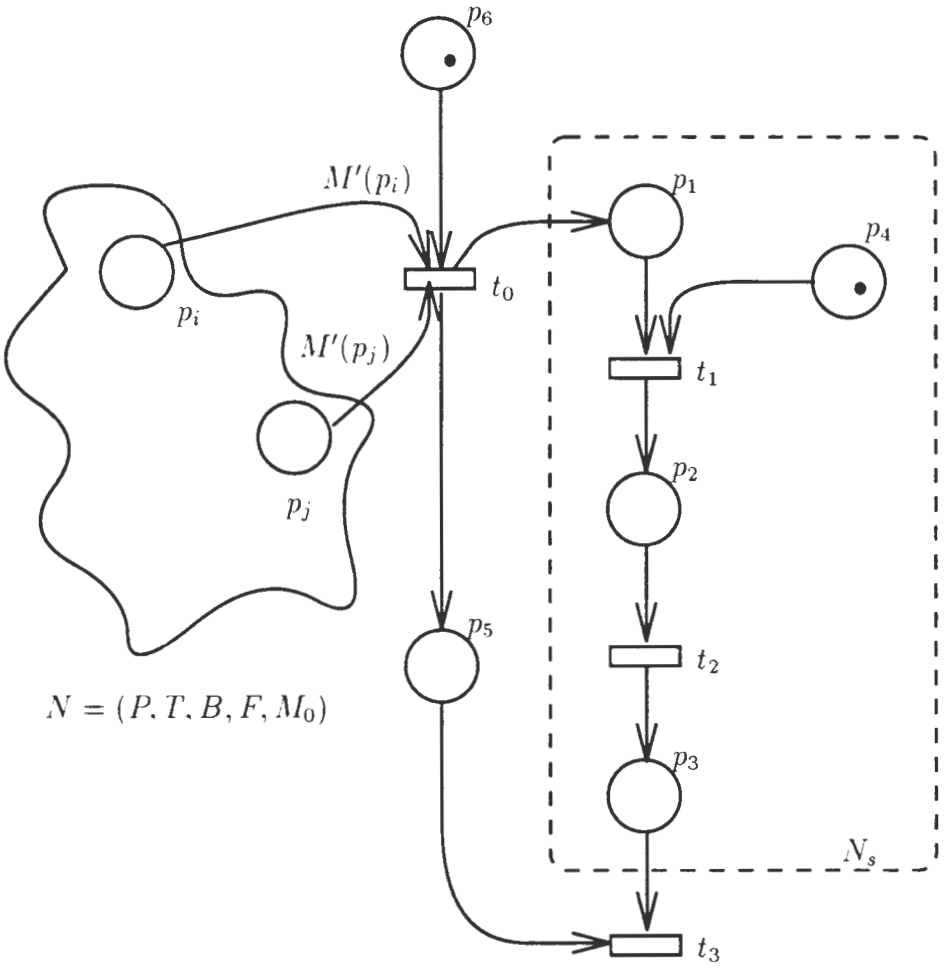


Figure 6.5: Reduction of the coverability problem to the problem of emptying a subnet

If it were known that a given subnet of a given Petri net satisfied either part 5, or alternatively part 7 of Definition 5.10, the problem of deciding whether the subnet is derivable would remain intractable. However, given that part 5 and part 7 of Definition 5.10 both hold, the problem is in $\mathcal{P}$ as the discussion of the algorithms above shows.

6.2.2 Regulation of Identical Transitions

Algorithm 6.12 decides whether a given set of transitions in a given Petri net are identical (in the sense of Definition 5.12 on page 57).

Algorithm 6.12

```
{ Algorithm to decide whether a given set of transitions are identical. }

procedure identical_test(B, F   : incidence_matrices;
                         S      : set of transition_range);

   var p         : place_range;
       t         : transition_range;
       identical : boolean;

   begin { identical_test }

      identical := true;
      for t := first_transition to last_transition do
         if t in S then
            for p := first_place + 1 to last_place do
               if B[first_place, t] <> B[p, t] then
                  identical := false
               else
                  if F[first_place, t] <> F[p, t] then
                     identical := false

   end; { identical_test }
```

Algorithm 6.12 has a worst case time complexity of

$$T(N, T_s) = O(|P||T|).$$

The candidate identical transitions do not have to take up any extra space in the input; some convention could be used, for example, specifying the candidate transitions as the first columns of B and F. Thus the input only has to contain the matrices B and F. If the encoded Petri net N is such that $|\langle N \rangle| = n$, then the complexity in terms of the input encoding length is

$$T(n) = O(n).$$

6.2.3 Stepwise Refinement of Transitions

Deciding whether a given transition in a given Petri net is k-enabled (Definition 5.13 on page 59) is an intractable problem:

Theorem 6.10 *Consider a given Petri net $N = (P, T, B, F, M_0)$, a transition $t_0 \in T$ and a nonnegative integer k. Deciding whether t_0 is k-enabled and not $(k+1)$-enabled is* **EXPSPACE**-*hard.*

The construction described in the following proof is illustrated in Figure 6.6.

Proof The proof is by a polynomial time reduction from the coverability problem. Let $N = (P, T, B, F, M_0)$, and a marking M' be an instance of the coverability problem. Construct a transition t' which has as inputs each of the places $p \in P$ with arc-weight $M'(p)$. Construct a place p_0, initially not marked, which has as input only the transition t' with arc-weight one. Transition t' has no other inputs or outputs. Construct another transition t_0 with an input from p_0 with arc-weight one and no outputs. Place p_0 has no other outputs. Finally, construct a place p_1, initially marked with one token, with no inputs and which is the only other input to transition t_0 with arc-weight one. Place p_1 has no other outputs. The transition t' will fire if and only if there exists a marking $M \in R(M_0)$ such that $M \geq M'$; this will ensure that at least one token could be placed on p_0. In this case t_0 could fire, removing the token from p_1. Thus there exists a reachable marking of the constructed Petri net such that t_0 is 1-enabled and not 2-enabled if and only if the given marking is coverable in the given Petri net. The hypothesis follows from Theorem 4.19 (page 36) (the coverability problem is **EXPSPACE**-hard). □

Leaving out the construction of the place p_1 and the arc to transition t_0 gives a reduction which shows that deciding whether a transition is k-enabled is **EXPSPACE**-hard.

If it is known that the transition t_0 is k-enabled but not (k+1)-enabled in some Petri net in which it is to be refined, then deciding whether a given Petri net is k-well-behaved (Definition 5.15 on page 59) is also intractable:

Theorem 6.11 *Consider a given Petri net $N = (P, T, B, F, M_0)$ a nonnegative integer k and transitions $t_{in}, t_{out} \in T$. Deciding whether N is k-well-behaved with respect to $t_{in}, t_{out} \in T$, $t_{in} \neq t_{out}$, is* **EXPSPACE**-*hard.*

The proof is by reduction from the single transition liveness problem. The construction used in the proof is illustrated in Figure 6.7:

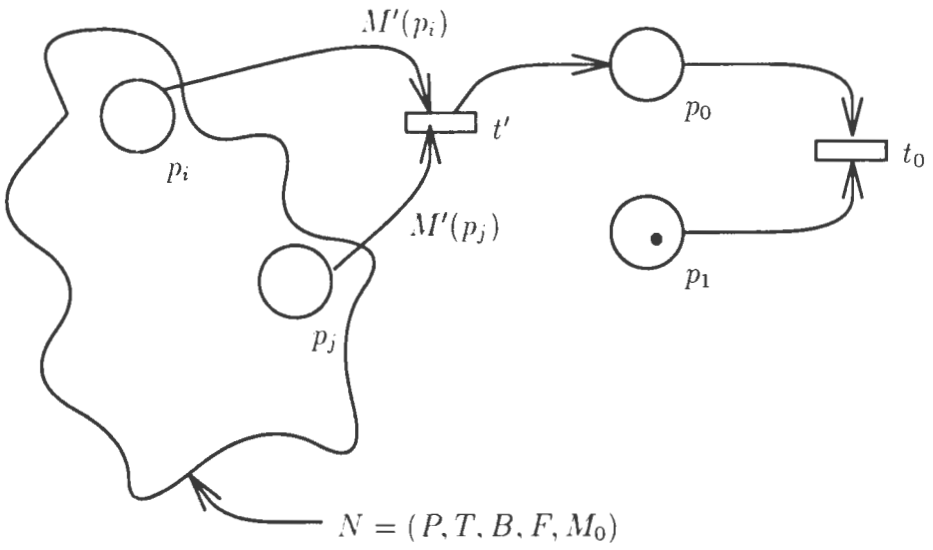


Figure 6.6: Reduction of the coverability problem to the k-enabled problem

Proof Consider a given instance of the single transition liveness problem; a Petri net $N = (P, T, B, F, M_0)$ and a transition $t_0 \in T$. Construct a place p_1 and a transition t_1 such that p_1 only has as input the transition t_0, and as output the transition t_1, both with arc-weights of one. Let this new Petri net be N'. N' is a block with respect to the transitions t_0 and t_1. Now consider the associated Petri net $N'' = B(N', t_0, t_1, 1)$. The loop $t_0 \rightarrow p_1 \rightarrow t_1 \rightarrow p_0 \rightarrow t_0$, marked with one token, does not affect the liveness of the transition t_0 in the given Petri net N. Furthermore, since the loop contains only one token, transitions t_0 and t_1 fire in strict alternation and so part 2 of Definition 5.15 holds. Since t_1 cannot fire until t_0 fires, part 3 of Definition 5.15 holds. Thus transition t_0 is live in N if and only if N' is 1-well-behaved with respect to t_0 and t_1. Hence according to Theorem 4.17 (page 35) and 4.18 the problem of deciding whether a given Petri net is k-well-behaved with respect to two given distinct transitions is **EXPSPACE**-hard. □

6.2.4 Refinement of Places

The S-transformation (Definition 5.16 on page 63) can always be applied to any place of a Petri net.

6.2.5 SL&SB Kit Refinement Rule $\mathcal{R}_3$

The definition of refinement rule $\mathcal{R}_3$ (Definition 5.18 on page 64) does not give application conditions, but rather, describes how the transformation is performed. Thus, this

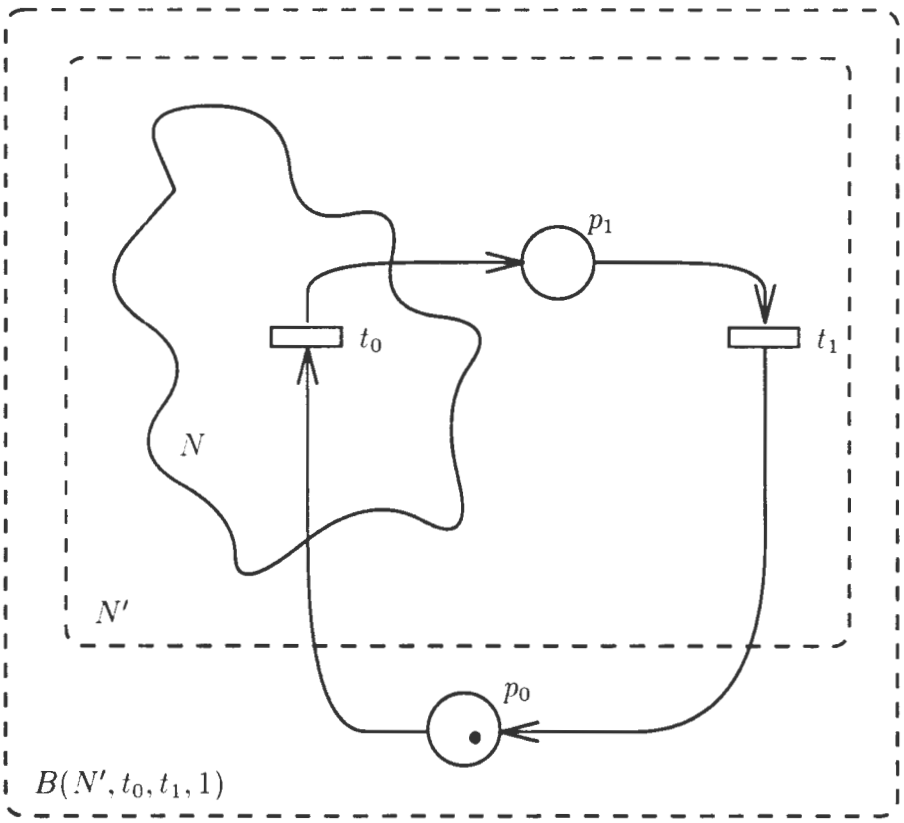


Figure 6.7: Reduction of the STL problem to the k-well-behaved problem

refinement rule can be performed on any Petri net and there is no problem deciding its applicability. If, however, a Petri net $N = (P, T, B, F, M_0)$ and a place $p_0 \notin P$ were given along with the arc-weights to and from the transitions in T, then there is the problem of deciding whether p_0 satisfies this rule. This problem is in $\mathcal{P}$ by Theorem 4.4 (page 30): Deciding whether there exists a solution $x \in \mathbb{Q}^n$ to a given system of linear equations $Ax = b$ where $A \in \mathbb{Z}^{m \times n}$ and $b \in \mathbb{Z}^m$ can be formulated as the following linear programming problem

$$\begin{bmatrix} A \\ -A \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \end{bmatrix}$$

and if in addition it is required that $x \in \mathbb{Q}^{+n}$ then

$$\begin{bmatrix} A \\ -A \\ -I \end{bmatrix} x \leq \begin{bmatrix} b \\ -b \\ O \end{bmatrix}. \tag{6.7}$$

6.2.6 SL&SB Kit Refinement Rule $\mathcal{R}_4$

Part 2b of the refinement rule $\mathcal{R}_4$ (Definition 5.18 on page 64) describes how the transformation is performed and parts 2a, part 2c and part 2d are the application conditions for the transformation.

Part 2a describes the condition that must be satisfied by the place p of that condition, which Algorithm 6.13 verifies, that is, that each of the input transitions of place p has other outputs and that each of place p output transitions have other inputs:

Algorithm 6.13

```
{ Algorithm to check that a place in a Petri net can be replaced
  by a state machine by refinement rule 4 of the SL&SB kit. }

procedure test_rule4(B, F : incidence_matrices; p0 : place_range);

   var p            : place_range;
       t            : transition_range;
       otherinputs  : boolean;
       otheroutputs : boolean;
       applicable   : boolean;

   begin { test_rule4 }

      otherinputs := false;
      otheroutputs := false;
```

```
      for t := first_transition to last_transition do
         for p := first_place to last_place do begin
            if (F[p0, t] > 0) and (p <> p0) and (B[p, t] <> 0) then
                  otheroutputs := true;
            if (B[p0, t] > 0) and (p <> p0) and (F[p, t] <> 0) then
                  otherinputs := true
            end;
      applicable := otherinputs and otheroutputs

   end; { test_rule4 }
```

The time complexity of Algorithm 6.13 is

$$T(N) = O(|P||T|).$$

If $|\langle N \rangle| = n$ then, as before, the complexity of the algorithm in terms of the input length is

$$T(n) = O(n).$$

Parts 2c and 2d of the refinement rule $\mathcal{R}_4$ can also be verified in polynomial-time: Consider $N'' = (P'', T'', B'', F'')$ and subsets $P''_I, P''_O \subseteq P''$ of way-in places and way-out places, respectively. The depth first search algorithm (DFS), [CLR90], can be used to verify Parts 2c and 2a: For each place $p'' \in P''$ search for a path from p'' to any place in P_I by using the reverse direction of the arcs of N''. For each of the searches the DFS can run with a time complexity of $T(N'') = O(|P''|^2)$ since N'' is a state machine, and it is the nodes of the state machine that are the nodes of the underlying graph. Also, the DFS can be used to search for a path from each of the places $p'' \in P''$ to each of the places in P''_O. The total time complexity of verifying whether N'' is a suitable candidate for the transformation is

$$\begin{aligned} T(N'') &= O(|P''||P''|^2 + |P''|^2|P''|^2) \\ &= O(|P''|^4). \end{aligned}$$

If $|\langle N'' \rangle| = n$ then

$$\begin{aligned} T(n) &= O((\frac{n}{2k})^4) \\ &= O(n^4). \end{aligned}$$

6.3 Event Graph Module Decomposition

Combining event graphs (Definition 5.20 on page 68) to form a composite model is a straightforward procedure.

For the minimal representation (Definition 5.22 on page 68) and its initial marking: Since the marked edges of the marked graph are always nonnegative integers, they can be regarded as nonnegative arc-weights in a weighted directed graph. The token distance between two transitions t_i and t_j in a marked graph is merely the path with minimum weight between the nodes t_i and t_j in the weighted directed graph. The token distance between t_i and t_j is thus the *shortest path weight* of the *single-pair shortest-path problem.* Since no algorithm for this problem has an asymptotic time complexity better than the *single-source shortest paths problem* [CLR90], and since the token distances are required from each transition $t_i \in T_{\text{in}}$ to each transition $t_j \in T_{\text{in}} \cup T_{\text{out}}$, Dijkstra's algorithm for the single-source shortest-paths problem may be more appropriate. Dijkstra's algorithm (which uses adjacency lists to represent the graph) for a graph $G = (V, E)$ has a time complexity of $T(G) = O(|V|^2)$. This gives the complexity of finding all the token distances in a marked graph $\mathcal{N} = (P, T, B, F, M_0)$ with $T_{\text{in}} \subseteq T$ as

$$\begin{aligned} T(\mathcal{N}, T_{\text{in}}) &= O(|T_{\text{in}}||T|^2) \\ &= O(|T|^3). \end{aligned}$$

If most of the transitions are in T_{in} then it may be more appropriate to use an *all-pairs shortest-paths* algorithm. In the worst cases (dense graphs), however, these have no better time complexities. The *Floyd-Warshall algorithm* for the all-pairs shortest-paths problem has a time complexity of $T(G) = O(|V|^3)$ and represents the graph as a weighted adjacency matrix W with the entries being the weights between the nodes [CLR90]. Considering a marked graph $\mathcal{N} = (P, T, B, F, M_0)$, the matrix W can be computed as follows: Construct the adjacency matrix A of the marked graph as the product $F^T B$. Construct a matrix E with entries $E(i, j)$ defined by

$$E(i,j) = \sum_{k=1}^{|P|} F(k,i)B(k,j)M_0(k),$$

then W is the matrix with entries $W(i, j)$ defined by

$$W(i,j) = \begin{cases} E(i,j) & \text{if } A(i,j) = 1 \text{ and} \\ \infty & \text{otherwise.} \end{cases}$$

The complexity of computing W is only $T_W(\mathcal{N}) = O(|P||T|^2 + |P||T|^2)$, by considering the multiplications required to multiply the matrices. The sets T_{in} and T_{out} can be specified by reorganizing the columns of B and F, and thus no extra space is required in the input encoding to specify them. With respect to the standard encoding of Petri nets, if $|\langle \mathcal{N} \rangle| = n$, then the complexity of finding the token distances is

$$\begin{aligned} T(n) &= O((\frac{n}{2k})^3 + k(\frac{n}{2k})^2 + k(\frac{n}{2k})^2) \\ &= O(n^3). \end{aligned}$$

6.4 Complete Reduction of Classes of Petri net

The discussion in Section 6.1 showed that, for liveness and/or boundedness analysis, tractable reducibility of the general Petri net, to a Petri net class in which liveness and boundedness analysis is tractable, is not possible. The classes of Petri net considered in this section are such that the complexity of determining the properties of liveness and boundedness are in $\mathcal{P}$. Thus although one might expect these classes to have tractable and effective reduction transformations, they are of less significance than, for example, reduction in the general Petri net when used to perform liveness and boundedness analysis. However, two important uses have emerged for such transformations:

- Only Petri nets within the larger class can be reduced to Petri nets of the smaller or simpler class, so the reduction transformations can be used to decide whether a given Petri net belongs to this class.
- Some of the results in this area have been useful in the development of efficient algorithms for the analysis of the properties that characterise the larger class [KB91].

Since reachability analysis is not necessary for these classes of Petri net in order to establish liveness and boundedness, the possible effects on the size of the state space is not considered.

6.4.1 Well-behaved Free Choice Systems

The applicability of the reduction transformations used by [Des90] to reduce well-behaved free choice systems (Definition 5.24 on page 72) can all be determined in polynomial-time.

The Place Reduction

Algorithm 6.14 finds all places in a given Petri net $N = (P, T, B, F, M_0)$ which can be removed using the P-reduction.

Algorithm 6.14

```
{ Algorithm to find all places in a Petri net which can be removed
  by the P-reduction. }

procedure place_reduction_find(B, F : incidence_matrices;
                               M0   : markings);
```

```
   var p1, p2     : place_range;
       t          : transition_range;
       removable : packed array [place_range] of boolean;

   begin { place_reduction_find }

      for p1 := first_place+1 to last_place do
         for p2 := first_place to p1-1 do begin
            removable[p1] := true;
            for t := first_transition to last_transition do
               if (B[p1, t] <> B[p2, t]) or (F[p1, t] <> F[p2, t]) or
                  (M0[p1] <> M0[p2]) then
                  removable[p1] := false
             end

   end; { place_reduction_find }
```

The time complexity of Algorithm 6.14 is $T(N) = O(|P|^2|T|)$ and so if $|\langle N \rangle| = n$, the complexity in terms of the length of the encoded input is

$$\begin{aligned} T(n) &= O((\frac{n}{2k+1})^2 k) \\ &= O(n^2). \end{aligned}$$

The Transition Reduction

Algorithm 6.15 finds all transitions that can be removed from a given Petri $N = (P, T, B, F, M_0)$ net using the T-reduction.

Algorithm 6.15

```
{ Algorithm to find all transitions in a Petri net which can be
  removed by the T-reduction. }

procedure transition_reduction_find(B, F : incidence_matrices;
                                    M0   : markings);

   var t1, t2     : transition_range;
       p          : place_range;
       removable : packed array [transition_range] of boolean;

   begin { transition_reduction_find }

      for t1 := first_transition+1 to last_transition do
         for t2 := first_transition to t1-1 do begin
            removable[t1] := true;
            for p := first_place to last_place do
               if (B[p, t1] <> B[p, t2]) or (F[p, t1] <> F[p, t2) then
                  removable[t1] := false
```

```
            end

   end; { transition_reduction_find }
```

The time complexity of Algorithm 6.15 is $T(N) = O(|P||T|^2)$. If $n = |\langle N \rangle|$ then the worst case complexity in terms of this input length is

$$\begin{aligned} T(n) &= O(k(\frac{n-k}{2k})^2) \\ &= O(n^2). \end{aligned}$$

The Flow Reduction

Section 6.1.4 has shown that the complexity of deciding the applicability of post-fusion of transitions is in $\mathcal{P}$. Furthermore, [JLL77] has shown that deciding k-boundedness of free choice nets is in $\mathcal{P}$ (a check for safeness can be used to check for possible contact situations), and hence deciding the applicability of the F-reduction is in $\mathcal{P}$.

The "A" Reduction

Algorithm 6.16 finds all pairs of transitions in a given Petri net $N = (P, T, B, F, M_0)$ for which the A-reduction is applicable.

Algorithm 6.16

```
{ Algorithm to find all pairs of transitions for which the A-reduction
  is applicable. }

procedure a_reduction_find(B, F : incidence_matrices);

   var t1, t1      : transition_range;
       p           : place_range;
       applicable  : packed array [transition_range, transition_range]
                     of boolean;
       outputs     : naturals;

   begin { a_reduction_find }

      for t1 := first_transition to last transition do
         for t2 := first_transition to last_transition do
            if t1 = t2 then
               applicable[t1, t2] := false
            else begin
               outputs := 0;
               applicable[t1, t2] := true;
               for p := first_place to last_place do begin
```

```
                outputs := outputs + B[p, t2];
                applicable[t1, t2] := applicable[t1, t2] and
                   (((B[p, t2] > 0) and (F[p, t1] > 0))) or
                    (F[p, t1] > 0))
                end;
             applicable[t1, t2] := applicable[t1, t2] and
                (outputs > 1)
             end

end; { a_reduction_find }
```

The time complexity of Algorithm 6.16 is $T(N) = O(|P||T|^2)$ and as shown above, if $|\langle N \rangle| = n$ then in terms of the length of the input instance the time complexity is

$$T(n) = O(n^2).$$

6.4.2 Reduction of Live and Bounded Free Choice Petri Nets

Esparza [Esp91], shows that each of his refinement rules (Section 5.3.2 on page 75) can be checked and performed in polynomial-time. This means that each of Esparza's kits of rules can completely reduce the class of live and bounded free-choice nets to an atomic net in polynomial-time [Esp91].

Chapter 7

Conclusion

This thesis has explored the viability of certain Petri net transformations. Those transformations, all proposed in the literature, that the author has selected for study, are ones which may be useful in liveness and boundedness analysis. Both synthesis and reduction transformations have been studied. Liveness and boundedness analysis is intractable for general Petri nets. Such analysis using reduction transformations was shown to be unviable in the general Petri net, as it contradicts the intractable nature of these problems. For synthesis transformations, the lack of suitable goals in building models necessitates the use of interactive tools so that a modeller can guide the construction of the Petri net.

For each of the reduction transformations the author has considered the complexity of determining the applicability of the transformations. The reduction in the size of the state space that can be achieved has also been illustrated. For synthesis transformations, the complexity of deciding the applicability of the transformations on specified components of a given Petri net has also been investigated. These complexity results are summarised in Table 7.1. The description of the problems in Table 7.1 assumes that $N = (P, T, B, F[, M_0])$ and that $|\langle N \rangle| = n$.

Additional results regarding special cases of well-known problems, which proved useful in establishing the transformation complexity results, were proved in Chapter 4.

Due to the inherent complexity of the liveness and boundedness problems and based on the results of this thesis, the author believes:

- Petri net transformations will prove themselves most suitable in their synthesis forms and when they are used in interactive modelling tools.
- Even though reduction transformations may not be useful in the analysis of Petri nets, they may provide useful insight into certain problems concerning Petri net

Problem Description	Complexity
Is a given place $p \in P$ redundant? Theorem 6.2 (page 85).	$\mathcal{P}$
Can Condition 5.4 of Definition 5.4 be satisfied such that for all $t_i \in T$, $w_i \leq b$, for some $b \in \mathbb{N}$? Theorem 4.14 (page 33).	$\mathcal{NPC}$
Are given places $p_1, p_2 \in P$ doubled? Theorem 6.6 (page 89)	$\mathcal{C}o$-**EXPSPACE**-hard
Can Condition 5.9 of Definition 5.5 be satisfied? Theorem 6.4 (page 87).	$\mathcal{C}o$-**EXPSPACE**-hard
Find all pairs of fusible equivalent places. Algorithm 6.2 (page 90).	$\mathcal{P}$, $O(n^3)$
Find all candidate for post-fusion of transitions. Algorithms 6.3 and 6.4 on pages 94 and 95.	$\mathcal{P}$, $O(n^2)$
Find all candidates for pre-fusion of transitions. Algorithms 6.5 and 6.6 on pages 97 and 98.	$\mathcal{P}$, $O(n^2)$
Find all candidates for lateral fusion of transitions. Algorithms 6.7 and 6.8 on pages 99 and 100.	$\mathcal{P}$, $O(n^3)$
Is a given subnet derivable? Theorem 6.7 (page 103) and Theorem 6.8 (page 108).	$\mathcal{C}o$-**EXPSPACE**-hard **EXPSPACE**-hard
Is a given subnet open? Algorithm 6.9 (page 104).	$\mathcal{P}$
Is a given subnet a marked graph? Algorithm 6.10 (page 106).	$\mathcal{P}$
Is a given subnet connected and void of any circuits (part 3 of Definition 5.10)? DFS algorithm (see page 106).	$\mathcal{P}$
Is there a path in a given subnet from any of its transitions to all the other transitions in the subnet (part 4 of Definition 5.10)? Topological sorting.	$\mathcal{P}$
Can a set of places in a subnet be partitioned according to part 6 of Definition 5.10? Algorithm 6.11 (page 107).	$\mathcal{P}$
Are a given set of transitions identical, and hence can be regulated? Algorithm 6.12 (page 110).	$\mathcal{P}$, $O(n)$
Is a given transition $t \in T$ k-enabled and not $(k+1)$-enabled, where $k \in \mathbb{N}$? Theorem 6.10 (page 111).	**EXPSPACE**-hard
Is a Petri net k-well-behaved with respect to two given distinct transitions? Theorem 6.11 (page 111).	**EXPSPACE**-hard
Can $p \in P$ be replaced by a state machine using $\mathcal{R}_4$ of the SL&SB kit? Algorithm 6.13 (page 114).	$\mathcal{P}$, $O(n)$
Does a given Petri net N'' satisfy the conditions for $\mathcal{R}_4$ of the SL&SB kit? DFS algorithm (see page 115).	$\mathcal{P}$, $O(n^4)$
Find the minimal representation of an augmented event graph. Shortest paths algorithms (see page 116).	$\mathcal{P}$, $O(n^3)$
Find all places that can be removed using the P-reduction. Algorithm 6.14 (page 117).	$\mathcal{P}$, $O(n^2)$
Find all transitions that can be removed using the T-reduction. Algorithm 6.15 (page 118).	$\mathcal{P}$, $O(n^2)$
Find all pairs if transitions for which the "A"-reduction is applicable. Algorithm 6.16 (page 119).	$\mathcal{P}$, $O(n^2)$

Table 7.1: Summary of transformation complexity results

analysis. This "side-effect" of Petri net transformation theory has already proved useful in the development of efficient algorithms for the analysis of Petri nets (see, for example, [KB91]).

The underlying complexity of the liveness and boundedness problems imposes restrictions on the ability of Petri net reduction transformations to reduce the general Petri net reachability set to a manageable size. The possibility exists that if other properties were considered for preservation as well as liveness and boundedness, useful classes of Petri nets could be reducible to more manageable problems. The complexity results of the transformations in [Des90] and [Esp91] show this to be true for the free-choice property.

7.1 Future Work

The aim of the analysis in this thesis was to establish whether the problems regarding the applicability of the Petri net transformations are tractable or not. For the hard problems, the thesis has not shown whether these problems are decidable. The decidability of a problem X can be established by showing that $X \preceq D$ for some decidable problem D. Alternatively, the undecidability could be established by showing that $U \preceq X$ for some undecidable problem U. The coverability problem, the reachability problem and the problems polynomially equivalent to the reachability problem (Theorem 4.17), are promising candidates for such decidable problems (Theorems 4.21 and 4.22) based on the constructions used in this thesis.

If reduction transformations are to be used in a tool to reduce the size of a Petri net reachability set, then for the given set of transformation an important property would be the Church-Rosser property, that is, whether the order of applying the transformations rules would be important in order to reach a *terminal* Petri net (one in which none of the application conditions hold) with a reachability set of a certain size. Alternatively, if the Church-Rosser property did not hold for a certain set of reduction transformations, then it would be useful to know whether an order of precedence could be established such that the least complex (to analyse) Petri net is produced.

Finally, it is the aim of the Data Network Architectures Laboratory research group to build tools for the analysis of protocol specifications, manufacturing systems and interactive Petri net modelling tools. If the underlying models of the manufacturing systems and protocol specifications can be represented by a class of Petri nets, then new transformations to assist either in the analysis, or the synthesis of such models would be useful. In particular, it would allow the Petri net modelling tools to test for certain properties of large protocol specifications and manufacturing systems.

Bibliography

[ABC86] M. Ajmone Marsan, Gianfranco Balbo, and Gianni Conte. *Performance Models of Multiprocessor Systems.* Computer Systems Series. The MIT Press, 1986.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[Baa88] Sara Baase. *Computer Algorithms: Introduction to Design and Analysis.* Addison-Wesley, second edition, 1988.

[BD90] Eike Best and Jörg Desel. Partial order behaviour and structure of Petri nets. *Formal Aspects of Computing*, 2(2):123–138, 1990.

[Ber86a] G. Berthelot. Checking properties of nets using transformations. In G. Rozenberg, editor, *Advances in Petri Nets 1985, Lecture Notes in Computer Science, volume 222*, pages 19–40. Springer-Verlag, 1986.

[Ber86b] G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Lecture Notes in Computer Science, volume 254*, pages 359–376. Springer-Verlag, 1986.

[BK91] Falko Bause and Peter Kemper. *QPN-Tool Version 1.0.* LS Informatik 4, Universität Dortmund, October 1991.

[BK93] Falko Bause and Pieter S. Kritzinger. *Introduction to Stochastic Petri Net Theory.* Submitted to Springer-Verlag, Berlin, 1993.

[Bro89] J. Glenn Brookshear. *Theory of Computation: Formal Languages, Automata, and Complexity.* Benjamin/Cummings, 1989.

[BRV80] G. Berthelot, G. Roucairol, and R. Valk. Reductions of nets and parallel programs. In Wilfried Brauer, editor, *Net Theory and Applications, Lecture Notes in Computer Science, volume 84*, pages 213–223. Springer-Verlag, 1980.

[BT87] Eike Best and Pazhamaneri S. Thiagarajan. Some classes of live and safe Petri nets. In *Advances in Petri Nets*, pages 71–94. Springer-Verlag, 1987.

[Chi85] Giovanni Chiola. A software package for the analysis of generalized stochastic Petri net models. In *International Workshop on Timed Petri nets, Torino, Italy*, pages 136–143. IEEE Computer Society, IEEE, July 1985.

[CLR90] Thomas H. Corman, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT/McGraw-Hill, 1990.

[Des90] Jörg Desel. Reduction and design of well-behaved free choice systems. Sonderforschungsbereich 342, Institut für Informatik, Technische Universität München, Postfach 20 24 20, D-8000 München 2, W-Germany, 1990.

[DLR79] David Dobkin, Richard J. Lipton, and Steven Reiss. Linear programming is log-space hard for *P*. *Information Processing Letters*, 8(2):96–97, February 1979.

[ES90] Javier Esparza and Manuel Silva. Top-down synthesis of live and bounded free choice nets. In *Application and Theory of Petri Nets: 11th International Conference*, pages 63–83, June 1990.

[Esp90] Javier Esparza. Synthesis rules for Petri nets, and how they lead to new results. In J. C. M. Baeton and J. W. Klop, editors, *CONCUR '90 Theories of Concurrency: Unification and Extension, Lecture Notes in Computer Science, volume 458*, pages 182–198. Springer-Verlag, 1990.

[Esp91] Javier Esparza. Reduction and synthesis of live and bounded free choice Petri nets. Informatik berichte, Institut für Informatik, Universität Hildesheim, Samelsonplatz 1, D-3200 Hildesheim, Germany, June 1991.

[Eve79] Shimon Even. *Graph Algorithms*. Computer Science Press, 1979. Also published by Pitman.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GSW80] H. J. Genrich and E. Stankiewicz-Wiechno. A dictionary of some basic notations of net theory. In Wilfried Brauer, editor, *Net Theory and Applications, Lecture Notes in Computer Science, volume 84*, pages 519–535. Springer-Verlag, 1980.

[HU69a] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata*. Addison-Wesley, 1969.

[Hu69b] T. C. Hu. *Integer Programming and Network Flows.* Addison-Wesley, 1969.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[JLL77] Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. Complexity of some Petri net problems. *Theoretical Computer Science*, 4:277–299, 1977.

[Joh90] David S. Johnson. A catalog of complexity classes. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Sceince: Volume A - Algorithms and Complexity*, chapter 2, pages 67–161. Elsevier/The MIT Press, 1990.

[JW83] Kathleen Jensen and Niklaus Wirth. *Pascal User Manual and Report.* Springer-Verlag, third edition, 1983.

[KB91] Peter Kemper and Falko Bause. An efficient polynomial-time algorithm to decide liveness and boundedness of free-choice nets. Technical report, Informatik IV, Universität Dortmund, Postfach 50 05 00, 4600 Dortmund 50, Germany, 1991.

[Knu69] Donald E Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, 1969.

[Kri93] Pieter S. Kritzinger. Private communication on future plans and ideas. Department of Computer Science, University of Cape Town, Private Bag RONDEBOSCH, 7700, South Africa. E-mail: postmaster@cs.uct.ac.za, August 1993.

[May84] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, August 1984.

[Mol89] Michael K. Molloy. Petri net modeling: The past, the present and the future. In *Third International Workshop on Petri Nets and Performance Models*, pages 2–9. IEEE, IEEE, 1989.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[NW88] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorical Optimization.* Wiley, 1988.

[Pet81] James L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, 198..

[Rei85] Wolfgang Reisig. *Petri nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1985.

[Sah74] Sartaj Sahni. Computationally related problems. *SIAM Journal on Computting*, 3(4):262–279, December 1974.

[SM83] Ichiro Suzuki and Tadao Murata. A method for stepwise refinement and abstraction of Petri nets. *Journal of Computer and System Sciences*, 27(1):51–76, 1983.

[SX92] Vanio M. Savi and Xiaolan Xie. Liveness and boundedness analysis for Petri nets with event graph modules. In K. Jensen, editor, *Application and Theory of Petri Nets 1992, Lecture Notes in Computer Science, volume 616*, pages 328–347. Springer-Verlag, 1992.

[Val79] R. Valette. Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, 18(1):35–46, 1979.

[vEB79] Peter van Emde Boas. Complexity of linear problems. In *Conference on Fundamentals of Computation Theory*, pages 117–120, 1979.

[vEB90] Peter van Emde Boas. Machine models and simulations. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Sceince: Volume A - Algorithms and Complexity*, chapter 1, pages 1–66. Elsevier/The MIT Press, 1990.

[vzGS78] Joachim von zur Gathen and Malte Sieveking. A bound on solutions of linear integer equalities and inequalities. *Proceedings of the American Mathematical Society*, 72(1):155–158, October 1978.

[WGR89] Fei-Yue Wang, Kevin Gildea, and Alan Rubenstein. A colored Petri net model for connection management services in MMS. *Computer Communication Review*, 19(3):76–98, July 1989.

[WW86] K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its Applications. D. Reidel Publishing Company, 1986.

[YC88] Stephen S. Yau and Ching-Roung Chou. Control flow analysis of distributed computing system software using structured petri net design. In *Proceedings of the Workshop on the Future Trends of Distributed Computing Systems in 1990s, Hong-Kong 1988*, pages 174–183, Washington, 1988.