Methods and Application of Petri Net Unfoldings for Discrete Event System Verification

January 1997

Toshiyuki Miyamoto

Methods and Application of Petri Net Unfoldings for Discrete Event System Verification

(ペトリネット展開を用いた離散事象システム検証に関する研究)

by

Toshiyuki Miyamoto

A dissertation submitted in partial fulfillment
of the requirements for the degree of
DOCTOR of ENGINEERING
at
the Graduate School of Engineering of Osaka University
Osaka, Japan
January 1997

Abstract

A Petri net is a formal model of concurrent systems which can be seen as an extension of automata theory to include concurrency with the aid of algebra and graph theory. Besides the advantages of the formal description of concurrent behavior that provides verification ability based on linear algebra, Petri nets are also attractive since they have a graphical representation. The formalism of Petri nets allows one to develop a detailed investigation method of complex behavior of a discrete event system.

Thus, the major strength of Petri nets is their ability of analyzing many properties and problems associated with the correctness in the design of discrete event systems. Reachability, liveness and safeness are fundamental problems in both applications and theoretical development of using Petri nets. The reachability problem in Petri nets can be described as; for given an initial state and a target state, "Is the target state reachable from the initial state?" It has been shown that the reachability problem is decidable, but it needs at least exponential time and space with respect to the size of a problem. When we restrict the structure of Petri nets, we can solve these problems in polynomial time. The methods, however, strictly depend on the structure, and it is impossible to extend the methods for Petri nets without restriction. Recent researches are made concerning on the computational aspect of these problems.

This thesis discusses a novel verification methodology of discrete event systems. The proposed method is based on the partial order defined in an occurrence net of events, called an unfolding. In this thesis, an efficient verification method is developed motivated with the application to the synthesis of asynchronous circuits.

We first study the construction of unfoldings. Unfoldings provide three types of

relation between instantiations of events. These relations play an important role in the verification of Petri nets. The efficiency of the construction of unfoldings affects much on that of the verification of properties of Petri nets. The proposed method efficiently constructs unfoldings. Experimental results shows efficiency the method. Next, we propose an efficient method of verification based on unfoldings of Petri nets. We introduce a concept of concurrent-relation graphs. The graph represents concurrent relation between instances in an unfolding. With the concurrent-relation graph, we can verify the reachability problem and the upper bound problem in an efficient way. Though the computational complexity of these two problems is exponential in theory, experimental results shows that the method can solve these problems in practical time in most case. Finally, we apply the unfolding method to the synthesis of asynchronous circuits which is the original motivation of the research of the thesis. Asynchronous circuits are important candidates of the next generation architecture such as fast and low power consumption devices. In the synthesis of asynchronous circuits modeled by Petri nets, however there exists a substantial problem of so called state explosion. To avoid this problem, state space reduction and parallel algorithm based on an efficient unfolding generation can be derived. A novel logic function deriving algorithm is proposed which is an extension of reachability problem considered in the previous chapter. The experimental results demonstrate the efficiency of the method compared with the existing methods.

Acknowledgments

I wish to express my appreciation to Professor Sadatoshi Kumagai of Osaka University for supervising this thesis, encouraging me during the course of this work and providing me with the ideal environment for my research activity.

I especially wish to express my sincere gratitude to Professor Dong-Ik Lee of Kwangju Institute Science and Technology for his continuous guidance and numerous valuable discussions throughout this work.

I am thankful to Professor Sinzo Kodama of Kinki University, Professor Hajime Maeda of Osaka University, Professor Toshimitsu Ushio and Dr. Shigemasa Takai for his encouragement throughout this work.

My sincere thanks are due to Professors Kenji Matsuura, Kiichiro Tsuji, Ryozo Aoki, Takatomo Sasaki, Jyunji Shirafuji, Tatsuhiko Yamanaka and Masahiro Nakatsuka for serving as members on my dissertation committee.

I am grateful to Professor Alex Kondratyev of the University of Aizu for his helpful comments on this work.

I would like to appreciate Professor Takashi Nanya of University of Tokyo for his helpful comments.

I would like to thank the members of Kumagai Laboratory at Osaka University for their kindly cooperation. In particular, Mr. Saito and Miss Hachikawa are to be thanked for helping me. Also I would like to thank Miss Momosaki for her encouragement.

Finally, I warmly thank my parents, who always support and encourage me.

Contents

Chapter 1

Introduction

1.1 Discrete Event Systems

A discrete event system is a dynamic system that evolves according to the spontaneous and asynchronous occurrence of events. Examples of discrete event systems include flexible manufacturing systems, communication networks, computer operating systems, traffic systems, database management systems and many other asynchronous distributed systems. Several models for discrete event systems have been proposed. The simplest models, called untimed models, ignore the timing of occurrence of events, and deal with only the order of event sequences. Untimed models are classified into logical models such as automata and Petri nets[48, 40] and algebraic models such as communicating sequential processes (CSP)[14] and a calculus for communicating systems (CCS)[32].

A Petri net is a formal model based on concepts from automata theory, linear algebra and graph theory. Besides the advantages of the formal description of concurrent behavior that provides verification ability based on linear algebra, Petri nets are also attractive since they have a graphical representation. In an early stage of design process, this graphical representation offers a visual aid to understand inner structure of a concurrent system; it gives a clear image of concurrency, sequentiality, and choice, which are the fundamental features of discrete event systems both in a intuitive visual level and a concrete graph-theoretical level. Moreover, the formal-

ism of Petri nets allows one to develop a detailed investigation method of complex behavior of a discrete event system.

To verify the correctness of discrete event systems is not a simple matter. Major strength of Petri nets is their ability of analyzing many properties and problems associated with discrete event systems. Two types of properties can be studied with a Petri net model; those which depend on the initial marking, and those which are independent of the initial marking. The former type of properties is referred to as marking dependent or behavioral properties, whereas the latter type of properties is called structural properties. Most of control problems of discrete event system such as delivering only correct results, termination, and absence of failure, etc., can be formulated by reachability problem in net theory. For given an initial state and a target state, the reachability problem in Petri nets can be described as; "Is there any firing sequence that transfers an initial state to a target state?" It has been shown that the reachability problem is decidable, but it needs at least exponential time and space with respect to the size of a problem. Deadlock-freeness and absence of overflow can be formulated by liveness and safeness in net theory, respectively. Reachability, liveness and safeness are fundamental problems in both applications and theoretical development of using Petri nets. As other fundamental properties of Petri nets, we can mention boundedness, reversibility, persistence. coverability, synchronic distance and fairness. These properties are structural properties, and similar to the reachability problem, we need at least exponential time and space to analyze these properties.

In analyzing these properties, several methods have been proposed, for example a reachability(coverability) graph method[25, 28, 40], a matrix equation approach[40], reduction or decomposition techniques[40, 26, 44], a symbolic model method[47, 12] and an unfolding method[29, 19, 37]. The reachability graph method is an exhaustive method. The method generates all the state space, therefore the method is a strong analysis method with the cost of constructing huge state space. Therefore the method is applicable only for small nets because of the state space explosion. The matrix equation approach or reduction and decomposition techniques are efficient methods comparing with reachability graph approach. These methods, however,

have limitation: the matrix equation approach can solve only a few problems, and decomposition techniques have strong restriction on the structure of Petri nets. The symbolic model method[47, 49, 12, 22] can deal with huge nets by using Binary Decision Diagrams (BDD's)[2]. Symbolic BDD traversal of a reachability graph allows us to obtain its implicit representation that is more compact than explicit enumeration of states. The BDD traversal, however, can be executed efficiently only if the property to be verified can be formulated by characteristic predicate. Therefore this method is effectively applied to systems modeled by k-bounded nets and finite capacity nets. The unfolding method was introduced to avoid generating the reachability graph for a bounded Petri nets[29, 19, 36]. It provides three types of relation between instantiations of events in the Petri net. Though the computational complexity of the symbolic model method and the unfolding method is still exponential, these methods perform efficiently in fairly general cases. The aim of these methods is to solve problems in practical time and space.

This thesis discusses a verification methodology of discrete event systems modeled by bounded Petri nets. The methodology is based on the partial order provided by the unfolding. The objective of the thesis is to provide practical verification method in the various aspect of the design of discrete event systems. This thesis is organized as stated in the next section.

1.2 Contributions and Organization of the Thesis

Chapter 2 gives basic notations on describing Petri nets and verifying nets.

In Chapter 3, we study a constructing method of unfoldings. Unfoldings were proposed by McMillan in 1993[29]. Unfoldings provide three types of relation between instantiations of events. First, we point out a problem in the McMillans's unfoldings. That is McMillan's unfoldings happens to be redundantly large. On the other hand, to use unfoldings for verification, care must be paid on the reduction size of unfoldings. An efficient construction method is proposed to achieve proper reduction of the size of unfoldings, and as a result the reduction of the time to construct an unfolding can be obtained.

Chapter 4 considers verification problems on Petri nets with unfoldings. First, we introduce a concept of concurrent-relation graphs. The graph represents concurrent relation between instances in an unfolding. We introduce two graph division methods, and it is shown that maximal complete subgraphs is preserved by the division methods. Next, it is shown that the reachability problem results to find a maximal complete subgraphs in a concurrent-relation graph, and then an algorithm to find the maximal complete subgraph is shown. Finally, we introduce a new problem, called the upper bound problem, and show that the upper bound problem also reduces to find the maximum complete subgraph in a concurrent-relation graph. An algorithm to find the maximum complete subgraph is shown for this problem.

In Chapter 5, we apply the unfolding method to the synthesis of asynchronous circuits. Asynchronous circuits are important candidates of the next generation architecture such as fast and low power consumption devices. In the synthesis of asynchronous circuits modeled by Petri nets, however there exists a substantial problem of so called state explosion. That is we have to generate the whole state space. To avoid this problem, state space reduction and parallel algorithm based on an efficient unfolding generation is discussed. We introduce concepts of P-Net's, T-Net's, tree, congruent relation, optimal tree selection and nesting. The P(T)-Net's are sub-nets of an unfolding, and they can be obtained by using concurrent relation between instances in the unfolding. We have succeeded to propose an efficient algorithm to derive next state logic functions by using the P(T)-Net's and other concepts proposed. A novel logic function deriving algorithm is proposed as an extension of reachability problem and the simulation results demonstrate the efficiency of the method compared with the existing methods.

Chapter 6 concludes the thesis.

Chapter 2

Preliminaries

This chapter gives basic notations on describing Petri nets and verifying nets.

2.1 Petri Net Model for Discrete Event Systems

This section introduces Petri nets to describe discrete event systems.

A Petri net(PN) is described by a five-tuple[40, 48]

$$\mathrm{PN} = (P, T, F, W; M_0), \tag{2.1}$$

where P is the set of places, T is the set of transitions, $F \subseteq (T \times P) \cup (P \times T)$ is flow relations and $W : F \to \mathcal{N}$ is a weight function, where $\mathcal{N}$ is the set of nonnegative integers. It is assumed that $P \cap T = \emptyset$. An arrangement of tokens on a Petri net is usually called a marking. In this thesis, we define a marking M as a multiset on P. M_0 is an initial marking. The number of tokens placed on p under a marking M is denoted by

$$\sharp(p, M). \tag{2.2}$$

A Petri net structure without any specific initial marking is denoted by $N = (P, T, F, W)$. A Petri net with the given initial marking M_0 is denoted by (N, M_0). In Figure 2.1, an example of Petri nets is shown, where places are drawn as circles, transitions are drawn as boxes, black dots in a place are tokens and arcs are labeled with their weights. Labels for unity weight are omitted.

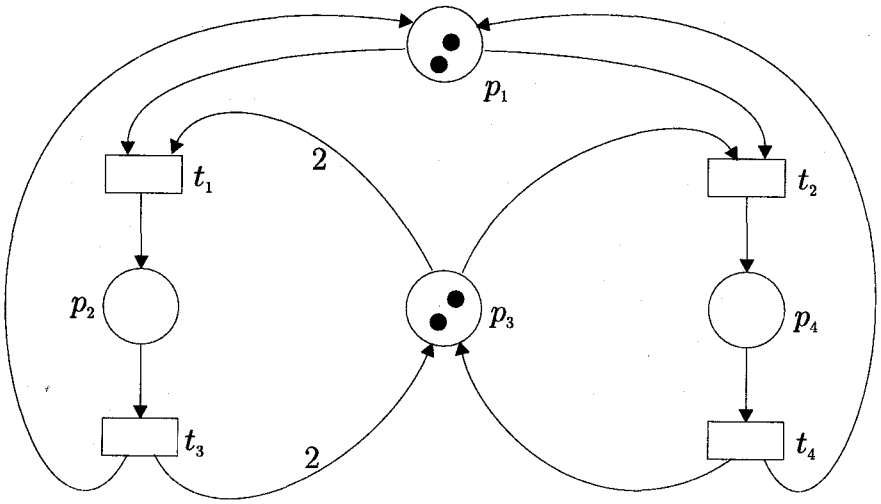

Figure 2.1: An example of Petri net.

A net N is called *ordinary*, if a weight for each arc in F is one. For a node[1] $x \in P \cup T$, $\bullet x$ (resp. $x\bullet$) denotes the multiset of input (resp. output) nodes of x. The number of appearing nodes in $\bullet x$ or $x\bullet$ is given by the weight function, namely, if for nodes x_1 and x_2, $(x_1, x_2) \in F$ and $W(x_1, x_2) = 2$, then there are two x_1 in $\bullet x_2$. A node x is called a *source* (resp. *sink*) node if $\bullet x = \emptyset$ (resp. $x\bullet = \emptyset$).

We review execution rules for a Petri net N briefly[40, 48]. We assume that no two transitions can fire simultaneously in N. A transition $t \in T$ is said to be *enabled* at a marking M if and only if each input place $p \in P$ of t has more than or equal to $W(p, t)$ tokens, denoted by the following notation,

$$M[t> . \tag{2.3}$$

By firing of a transition t, a marking M transforms to M', denoted by the following notation,

$$M[t> M'. \tag{2.4}$$

[1] A node is a place or a transition.

M' is obtained from M by deleting $W(p,t)$ tokens from each input place of t and adding $W(t,p)$ tokens to each output place of t. In case of Figure 2.1, transitions t_1 and t_2 are enabled at marking $M_0 = \{p_1, p_1, p_3, p_3\}$. After a fire of transition t_1, M_0 changes to new marking $M_1 = \{p_1, p_2\}$.

2.2 Behavioral Properties of Petri Nets

A major strength of Petri nets is their support of analysis of many properties and problems associated with discrete event systems. Two types of properties can be studied with a Petri net model: those which depend on the initial marking and those which are independent of the initial marking. The former type of properties is referred to behavioral properties, whereas the latter type of properties is called structural properties. This section gives basic behavioral properties and their analysis problems, which are related to this thesis.

2.2.1 Reachability

Reachability is a fundamental basis for studying the dynamic properties of discrete event systems. A sequence of firings produces a sequence of markings. A marking M_n is said to be reachable from a marking M_0, if there exists a sequence of firings that transforms M_0 to M_n. The set of all possible markings reachable from M_0 is denoted by,

$$R(N, M_0) \qquad \text{or simply} \qquad R(M_0). \tag{2.5}$$

The set of all possible markings reachable from M_0 without firing of transition $t \in T$ is denoted by

$$R(N, M_0)\backslash t, \tag{2.6}$$

and the set of all possible markings reachable from M_0 without firing of transitions in the set $Q \subseteq T$ is denoted by

$$R(N, M_0)\backslash Q. \tag{2.7}$$

The reachability problem for Petri nets is the problem of finding if $M_n \in R(N, M_0)$ for a given marking M_n in a net (N, M_0). The reachability problem is decidable[25, 28] although it takes exponential space and time to verify in the general case.

2.2.2 Boundedness

A net (N, M_0) is called *bounded* or *safe*, if for any marking M reachable from an initial marking M_0, $\sharp(p, M)$ is bounded or at most one for each $p \in P$, respectively. Places in a Petri net are often used to represent buffers and registers for storing intermediate data. By verifying that the net is bounded or safe, it is guaranteed that there will be no overflows in the buffers or registers, no matter what firing sequence is taken.

Upper Bound. An upper bound $\sharp(p, \overline{M})$ of a place $p \in P$ is the maximum number of tokens that are placed at the place p in a marking reachable from M_0, that is

$$\sharp(p, \overline{M}) = \max_{M \in R(N, M_0)} \sharp(p, M). \tag{2.8}$$

The upper bound problem for Petri nets is the problem of finding the upper bound for each place $p \in P$. By finding the upper bound, we can estimate the number of required buffers or registers. The upper bound $\overline{M}$ of the net in Figure 2.1 is as follows:

$$\overline{M} = \{p_1, p_1, p_2, p_3, p_3, p_4, p_4\}. \tag{2.9}$$

2.2.3 Liveness

A net (N, M_0) is said to be *live* if, no matter what marking has been reached from M_0, it is possible to fire ultimately any transition of the net by progressing through some further firing sequence. This means that a live Petri net guarantees deadlock-free operation, no matter what firing sequence is chosen. The net in Figure2.1 is a live net.

2.2.4 Reversibility

A net (N, M_0) is said to be *reversible* if, for each marking $M \in R(N, M_0)$, M_0 is reachable from M. The net in Figure2.1 is a reversible net.

2.2.5 Persistence

A net (N, M_0) is said to be *persistent* if, for any two enabled transitions, the firing of one transition will not disable the other. Persistence is closely related to the speed-independent implementation of asynchronous circuits[4, 19, 21, 22, 24, 30]. The net in Figure2.1 is not a persistent net.

2.3 Analysis Methods

This section reviews analysis methods of Petri nets. Analysis methods have been classified into the following three groups so far:

1. the reachability(coverability) graph method,
2. the matrix equation approach, and
3. reduction or decomposition techniques.

Recently, other methods, called a symbolic model method and partial order approach or unfolding method, draw much attention.

2.3.1 The Reachability Graph Method

A reachability graph RG of a Petri net PN = (N, M_0) is a digraph and is described by a two-tuple

$$\text{RG} = (V_R, E_R), \tag{2.10}$$

where V_R is the set of markings reachable from M_0 in the PN, i.e.,

$$V_R = R(M_0), \text{ and} \tag{2.11}$$

$$E_R = \{(M_i, M_j) | M_i, M_j \in V_R, \exists t \in T \text{ such that } M_i[t> M_j\} \tag{2.12}$$

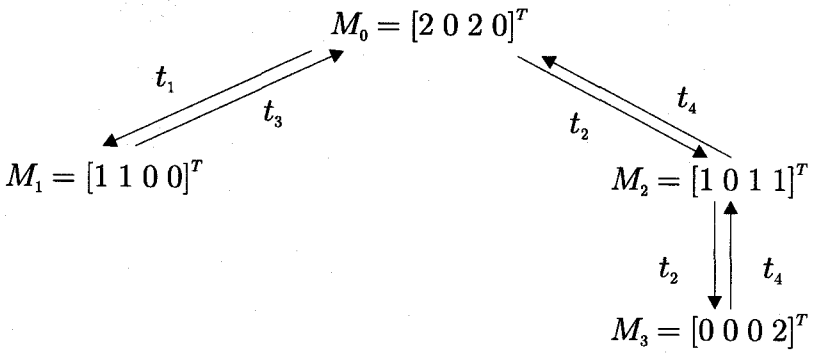

Figure 2.2: The reachability graph of the Petri net in Figure2.1.

is the set of directed edges. A directed edge (M_i, M_j) is drawn from M_i to M_j. In Figure 2.2, the reachability graph of the Petri net in Figure 2.1 is shown. In the figure, a marking M is written as an $m \times 1$ column vector, where m is the number of places. The j-th entry of M denotes the number of tokens in place p_j.

The reachability graph method is applicable only to bounded net, since, if a given net is not bounded, the set of reachable markings increases infinitely. In case of unbounded net, we use the coverability graph instead of the reachability graph[40]. This thesis studies only bounded nets, therefore discussion about the coverability graphs is omitted.

The reachability graph method is the most powerful method in analyzing Petri net properties. By using this method, all the previous properties can be verified. It, however, requires exponential time and space in general case, therefore it is impossible to apply this method to large Petri nets.

2.3.2 Incidence Matrix and State Equation Methods

Incidence Matrix. For a Petri net N with n transitions and m places, the incidence matrix $A = [a_{ij}]$ is an $n \times m$ matrix of integers and its typical entry is given by[40]

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-} \tag{2.13}$$

where $a_{ij}^{+} = W(i,j)$ is the weight of the arc from transition i to its output place j and $a_{ij}^{-} = W(j,i)$ is the weight of the arc to transition i from its input place j. Let us consider the Petri net shown in Figure2.1. Its incidence matrix A is given by

$$A = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \begin{array}{c} \begin{array}{cccc} p_1 & p_2 & p_3 & p_4 \end{array} \\ \begin{bmatrix} -1 & 1 & -2 & 0 \\ -1 & 0 & -1 & 1 \\ 1 & -1 & 2 & 0 \\ 1 & 0 & 1 & -1 \end{bmatrix} \end{array} \tag{2.14}$$

State Equation. In writing matrix equations, we write a marking M_k as an $m \times 1$ column vector. The j-th entry of M_k denotes the number of tokens in place j immediately after the k-th firing in some firing sequence. The k-th firing vector u_k is an $n \times 1$ column vector of $n-1$ 0's and one non-zero entry, a 1 in the i-th position indicating that transition i fires at the k-th firing. Since the i-th row of the incidence matrix A denotes the change of the marking as the result of firing transition i, we can write the following state equation for a Petri net[39, 40]:

$$M_k = M_{k-1} + A^T u_k \ , \ k = 1, 2, \cdots \tag{2.15}$$

The state equation provides a necessary condition for the reachability problem[40]. Suppose that a destination marking M_d is reachable from M_0 through a firing sequence $u_1, u_2, \cdots, u_d$. Writing the state equation (2.15) for $i = 1, 2, \cdots, d$ and summing them, we obtain

$$M_d = M_0 + A^T \sum_{k=1}^{d} u_k \tag{2.16}$$

which can be rewritten as

$$A^T x = \Delta M \tag{2.17}$$

where $\Delta M = M_d - M_0$ and $x = \sum_{k=1}^{d} u_k$. Here x is an $n \times 1$ column vector of nonnegative integers. The existence of such a vector x is the necessary condition for that M_d is reachable from M_0.

Structural Boundedness. A Petri net N is said to be *structurally bounded* if it is bounded for any finite initial marking M_0. For structural boundedness, there is a well-known proposition[40].

Proposition 2.1 A Petri net N is structurally bounded if and only if there exists an m-vector y of positive integers such that

$$Ay \leq 0. \tag{2.18}$$

Invariants. An integer solution y of the homogeneous equation,

$$Ay = 0 \tag{2.19}$$

is called an S-invariant, and an integer solution x of the transposed homogeneous equation $A^T x = 0$ is called a T-invariant.

The set of places (transitions) corresponding to nonzero entries in an S-invariant $y \geq 0$ (T-invariant $x \geq 0$) is called the support of an invariant. A support is said to be minimal if no proper nonempty subset of the support is also a support. An invariant y is said to be minimal if there is no other invariant y_1 such that $y_1(p) \leq y(p)$ for all p. Given a minimal support of an invariant, there is a unique minimal invariant corresponding to the minimal support. We call such an invariant a minimal-support invariant. The set of all possible minimal-support invariants can serve as a generator of invariants. That is, any invariant can be written as a linear combination of minimal-support invariants[31]. In [27], a fast algorithm to obtain all minimal-support invariants is discussed.

The upper bound discussed in subsection 2.2.2 can be calculated by using minimal-support S-invariants for structurally bounded Petri nets. The upper bound is given by the following equation.

$$\sharp(p, \overline{M}) \leq \min[M_0^T y_i / y_i(p)] \tag{2.20}$$

where the minimum is taken over all nonnegative minimal-support S-invariant y_i such that $y_i(p) \neq 0$. It is shown in [51] that this upper bound can not be improved by using any other invariants. This method can be used only a case where, for each

place $p \in P$, there exists at least one S-invariant y such that $y(p) > 0$. Let us consider the Petri net shown in Figure2.1. Its incidence matrix A is given by equation (2.14). $y_1 = [1\ 1\ 0\ 1]^T$ and $y_2 = [0\ 2\ 1\ 1]^T$ are minimal-support S-invariants. Consider equation (2.20) for place p_1 and $M_0 = [2\ 0\ 2\ 0]^T$.

$$\begin{aligned} \sharp(p_1, \overline{M}) &\leq \min[M_0^T y_1 / y_1(p_1), M_0^T y_2 / y_2(p_1)] \\ &= \min[2/1, 2/1] \\ &= 2 \end{aligned} \tag{2.21}$$

We can obtain the upper bound $\overline{M}$ by applying equation (2.20) for each place. The upper bound is

$$\overline{M} = \begin{bmatrix} 2 & 1 & 2 & 2 \end{bmatrix}^T. \tag{2.22}$$

The incidence matrix and state equation methods require polynomial time and space. Therefore, this method is efficient with respect to computational complexity as compared with the reachability graph method. This method, however, can be applied only to some subclass of Petri nets, and can not be applied to some other problems, for example reachability, liveness, reversibility and persistence.

2.3.3 Reduction and Decomposition Methods

Reduction Method. To facilitate the analysis of a large system, we often reduce the system model to a simpler one, while preserving the system properties to be analyzed. In [40, 52], simple transformations, which can be used for analyzing liveness, safeness and boundedness, are shown. In [42], an abstraction method for Petri nets based on an equivalence of firing sequences of a specified subnet or a specified subset of transitions was discussed.

As pointed out previously, a major weakness of Petri nets is the complexity problem. Thus the reduction method is very important. This method, however, can not be applied to some problems, for example reachability, reversibility and persistence.

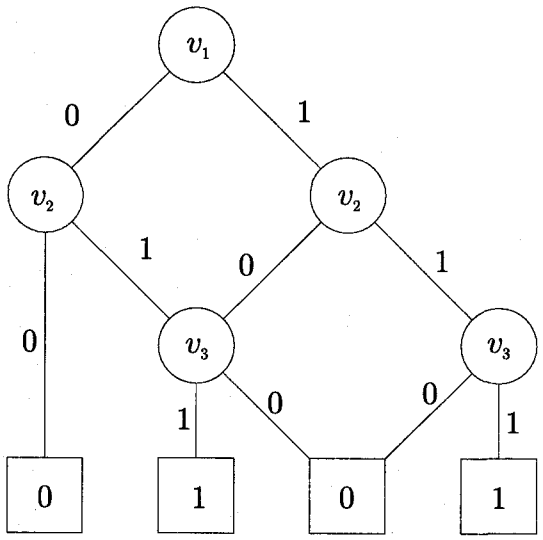

Figure 2.3: A BDD representing a Boolean function $(v_1 \vee v_2) \wedge v_3$.

Decomposition Method. As another method to facilitate the analysis of a large system, there is the decomposition method[3, 6, 7, 11, 26, 56, 57]. In [26], analysis methods for liveness, safeness and reachability of free choice nets[40] are discussed. In [56, 57], the decomposition method is applied to equal conflict systems. Computational complexity of these methods is polynomial, thus the decomposition method provides an efficient verification method. It, however, can be applied only to subclass of Petri nets. It will be impossible that we apply the decomposition technique to general Petri nets, and the restriction is too strong to apply general systems.

2.3.4 The Symbolic Model Method

This subsection reviews the symbolic model method[1, 2, 12, 13, 47, 49, 50] .

Binary Decision Diagrams. Binary decision diagrams (BDD's) have been proposed for representing and manipulating Boolean functions[1, 2]. A BDD is a rooted, directed acyclic graph. The graph in Figure 2.3 is representing a Boolean function

$$f(v_1, v_2, v_3) = (v_1 \vee v_2) \wedge v_3. \tag{2.23}$$

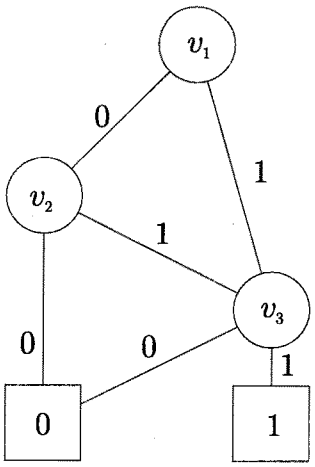

Figure 2.4: A reduced ordered BDD for $(v_1 \vee v_2) \wedge v_3$.

We enter at the root node and then proceed downward through the graph, at each node checking the value of its variable and taking the indicated branch. When a 0 or a 1 value is reached, this gives the value of f.

A given function can be represented by many different BDD's. Bryant proposed restrictions on BDD's so that any function has exactly one canonical BDD[2]. A total ordering is given for the variables, and a reduction of BDD's is introduced. A BDD can be reduced in the size without changing the denoted function by eliminating redundant vertices and duplicate subgraphs. Figure 2.4 shows an example of reduction of BDD's. For any Boolean function f, there is a unique (up to isomorphism) reduced ordered BDD (ROBDD) denoting f. This means that ROBDD's give a canonical form for each Boolean function. Most commonly encountered functions have a reasonable representation. For example, all symmetric functions are represented by ROBDD's where the number of vertices grows at most the square of the number of arguments. In addition, the time complexity of any single operation ($\vee$, $\wedge$, etc.) is bounded by the product of the graph size.

The ordering of variables affects the size of an ROBDD. The problem of computing an ordering that minimizes the size of a given ROBDD is a co NP-complete problem. However, by using a small set of heuristics, one can select an adequate

ordering most the time.

Hereafter, we assume that BDD's are reduced and ordered.

Modeling $R(N, M_0)$ with BDD's. BDD's can represent large Boolean functions compactly. If we can express a marking by a Boolean function, this will be a good expression of the set of reachable markings. This technique was introduced in [47].

Let us explain the technique briefly. Assume that the upper bound of a given Petri net (N, M_0) is $\overline{M}$. For each place $p_i \in P$, we need q_i variables, which is given by

$$q_i = \lceil \log_2(\sharp(p_i, \overline{M}) + 1) \rceil, \tag{2.24}$$

where $\lceil x \rceil$ is the smallest integer that is more than or equal to x. Therefore each marking that is reachable from M_0 is expressed by a Boolean function with v variables, where

$$|v| = \sum_{i=1}^{m} q_i. \tag{2.25}$$

By summing Boolean functions, we can represent the set of reachable markings with a BDD.

Example 2.1 As calculated before, the upper bound of the Petri net in Figure 2.1 is given by (2.22). Required variables are the following

$$q_1 = q_3 = q_4 = 2, \tag{2.26}$$

$$q_2 = 1, \text{and} \tag{2.27}$$

$$|v| = 7. \tag{2.28}$$

Assume that v_1 and v_2 are used for place p_1, v_3 is used for p_2, v_4 and v_5 are used for p_3 and v_6 and v_7 are used for p_4. Each reachable marking shown in Figure 2.2 is expressed by the following functions.

$$f_0 = v_1 \overline{v_2}\,\overline{v_3}\, v_4\, \overline{v_5}\,\overline{v_6}\,\overline{v_7} \tag{2.29}$$

$$f_1 = \overline{v_1}\, v_2\, v_3\, \overline{v_4}\,\overline{v_5}\,\overline{v_6}\,\overline{v_7} \tag{2.30}$$

$$f_2 = \overline{v_1}\, v_2\, \overline{v_3}\,\overline{v_4}\, v_5\, v_6\, \overline{v_7} \tag{2.31}$$

$$f_3 = \overline{v_1}\,\overline{v_2}\,\overline{v_3}\,\overline{v_4}\,\overline{v_5}\, v_6\, \overline{v_7} \tag{2.32}$$

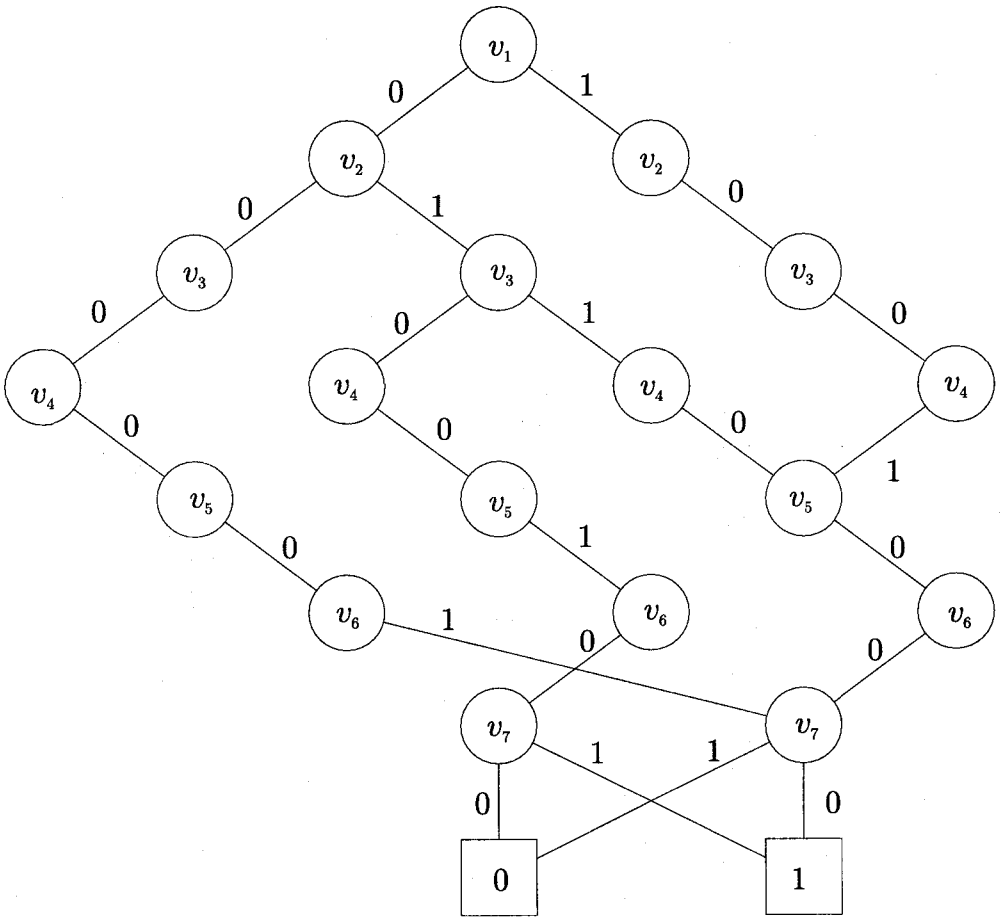

Figure 2.5: A BDD representation of $R(N, M_0)$.

The BDD of $R(N, M_0)$ is drawn in Figure 2.5. In the figure some edges are omitted to simplify. All the omitted edges are drawn to the node 0.

BDD Construction Algorithm. In Figure 2.6, an algorithm to construct a BDD from a PN is shown.

BDD's make it possible to operate on the set of markings, therefore computational complexity of the method depends on depth of the reachability graph. Consequently, great improvement on both time and space can be expected.

There exist several results on this area. In [47], verification of safeness, liveness, persistence is discussed. In [12], verification of concurrent systems described by finite

Algorithm 2.1 *Construction of a BDD from a PN*
input: a Petri net PN.
output: a BDD.
begin
$From = M_0$;
$Reach = M_0$;
while $From \neq \emptyset$ **do**
foreach transition $t \in T$
New = the set of markings generated by firing t from $From$;
$From = From \cup New$;
endforeach
$From = From \cap \overline{Reach}$;
$Reach = Reach \cup From$;
endwhile
return $Reach$;
end

Figure 2.6: An Algorithm to Construct BDD from a PN.

capacity Petri nets is discussed. In [49], verification of deadlock freeness and home state property is discussed. In [13], a method for computing reduced representation of vector state spaces consisting of infinitely many states is presented.

2.3.5 The Partial Order Method

Occurrence Nets. Occurrence nets (OCN's) were introduced to represent both causality and concurrency between events[29]. Intuitively speaking, an occurrence net represents possible system traverse by its net structure.

Definition 2.1 (Occurrence net) An OCN of a PN $= (P, T, F, W; M_0)$ is a five-tuple

$$\text{OCN} = (P', T', F'; M_0'; \mathcal{L}), \tag{2.33}$$

where $\mathcal{L} : P' \cup T' \rightarrow P \cup T$ is a function which maps P' onto the set P and T' onto the set T. The net must have the following properties:

1. $\forall p \in P' : |\bullet p| \leq 1$,
2. $\forall t_1, t_2, t_3 \in T' : (t_1, t_3) \in F'^*, (t_2, t_3) \in F'^*$ and $\bullet t_1 \cap \bullet t_2 \neq \emptyset \Rightarrow t_1 = t_2$,
3. $\forall t_1, t_2 \in T' : \mathcal{L}(t_1) = \mathcal{L}(t_2), \bullet t_1 = \bullet t_2 \Rightarrow t_1 = t_2$,
4. $\forall t \in T' : \mathcal{L}(\bullet t) = \bullet \mathcal{L}(t)$ and $\mathcal{L}(t\bullet) = \mathcal{L}(t)\bullet$, and
5. $M_0' = *P'$ and $\mathcal{L}(M_0') = M_0$.

For a set of nodes X' of an OCN, $\mathcal{L}(X') = \{\mathcal{L}(x') | x' \in X'\}$. $*P$ is the set of source places on P. F^* is the transitive closure of F.

An OCN is an ordinary net, therefore the weight function is omitted. A node in an OCN is called a (i-th) instance of a node in the PN, for example, p_1^1 is called the first instance of p_1. An OCN is an acyclic directed graph, and therefore the transitive closure of the flow relation defines the partial order between nodes. This partial order will be called the precedence relation and denoted by $\Rightarrow$. When

$$x_1 \Rightarrow x_2, \tag{2.34}$$

node x_1 is called a predecessor of node x_2 and node x_2 is called a successor of node x_1. For two nodes that are not in precedence relation in an OCN, let us define two relations[29, 19].

Definition 2.2 (Conflict) In an OCN $= (P', T', F'; M_0'; \mathcal{L})$, nodes $x_1, x_2 \in P' \cup T'$ are in *conflict* relation, denoted by

$$x_1 \sharp x_2, \tag{2.35}$$

if there exist distinct transitions $t_1, t_2 \in T'$ such that $\bullet t_1 \cap \bullet t_2 \neq \emptyset$, and $(t_1, x_1), (t_2, x_2) \in F'^*$.

Definition 2.3 (Concurrent) Nodes x_1 and x_2 in an OCN are in *concurrent* relation, denoted by

$$x_1 // x_2, \tag{2.36}$$

if they are neither in conflict relation nor in precedence relation.

In Figure 2.7, an algorithm to construct an OCN from a given PN is shown[29].

In Figure 2.8, a PN and its OCN are shown. An OCN has following properties. The image of the set of markings reachable from M_0' by the map $\mathcal{L}$ is equal to the set of markings reachable from M_0 of the corresponding PN. Therefore

$$R(N, M_0) = \mathcal{L}(R(N', M_0')). \tag{2.37}$$

An OCN is a safe Petri net, because each place has at most one input transition and M_0' is a safe marking. In an OCN an unsafe marking of a PN is represented by plural copies of the unsafe place. See Figure 2.8, an unsafe marking $M = \{p_1, p_1\}$ of the Petri net is represented by $M' = \{p_1^1, p_1^2\}$ that is a safe marking.

The most interesting feature of OCN's is their structural properties. In a PN, even if events can occur concurrently, one of them can be a predecessor of the other when the given PN is a cyclic PN. In a OCN, however, if instances are in concurrent relation they remain to be in concurrent relation in any firing sequences. In an OCN, transitions t_1 and t_2 are in concurrent relation if and only if there exists a reachable marking M, in which both t_1 and t_2 are enabled and the firing of any of them does not disable the other. Places p_1 and p_2 are in concurrent relation if and only if there exists a reachable marking, in which both p_1 and p_2 are marked. Moreover by seeking relationship between nodes in OCN's, we can obtain detail information about reachable states. For example, in Figure 2.8, for three instances p_3^1, p_3^2 and p_4^2, we can find three relations: $p_3^1 // p_3^2, p_3^1 // p_4^2$ and $p_3^2 \sharp p_4^2$. These relations show that in the PN markings $\{p_3, p_3\}$ and $\{p_3, p_4\}$ are reachable, but $\{p_3, p_3, p_4\}$ is not reachable. Note that, for a given flow relation, nodes x_1 and x_2 are at least in one of the relations $x_1 \Rightarrow x_2$,$x_2 \Rightarrow x_1$,$x_1 \sharp x_2$, or $x_1 // x_2$.

Algorithm 2.2 *Construction of an OCN from a PN*

input: a Petri net PN.

output: an occurrence net OCN.

begin

Copy all places $p \in M_0$ into the OCN.;

/* If M_0 is not safe marking, make as many copies of a place as the number of appearances in M_0 */

while as many as possible **do**

Choose a transition $t \in T$ from the PN.;

For each place in $\bullet t$, find a copy in the OCN. Let denote the set of copies by $P'_{\bullet t} \subseteq P'$. Do not choose the same set twice for the t.;

if You could not find the set $P'_{\bullet t}$, **then**

Go back to the top of this while loop.;

else

if $\exists p_1, p_2 \in P'_{\bullet t} : p_1 \sharp p_2$, $p_1 \Rightarrow p_2$ or $p_2 \Rightarrow p_1$ **then**

Go back to the top of this while loop.;

else

Make a copy of t in the OCN. Call it t'.;

Draw an arc from each places in $P'_{\bullet t}$ to t'.;

For each place in $t\bullet$, make a copy in the OCN and draw an arc from t' to it.;

endif

endif

endwhile

end

Figure 2.7: An Algorithm to construct an OCN.

Unfoldings For live Petri nets, OCN's can be extended infinitely, so it is impossible to apply OCN's for the verification of Petri nets. Although an OCN for a cyclic PN can be extended infinitely, it is always possible for a bounded PN to truncate the

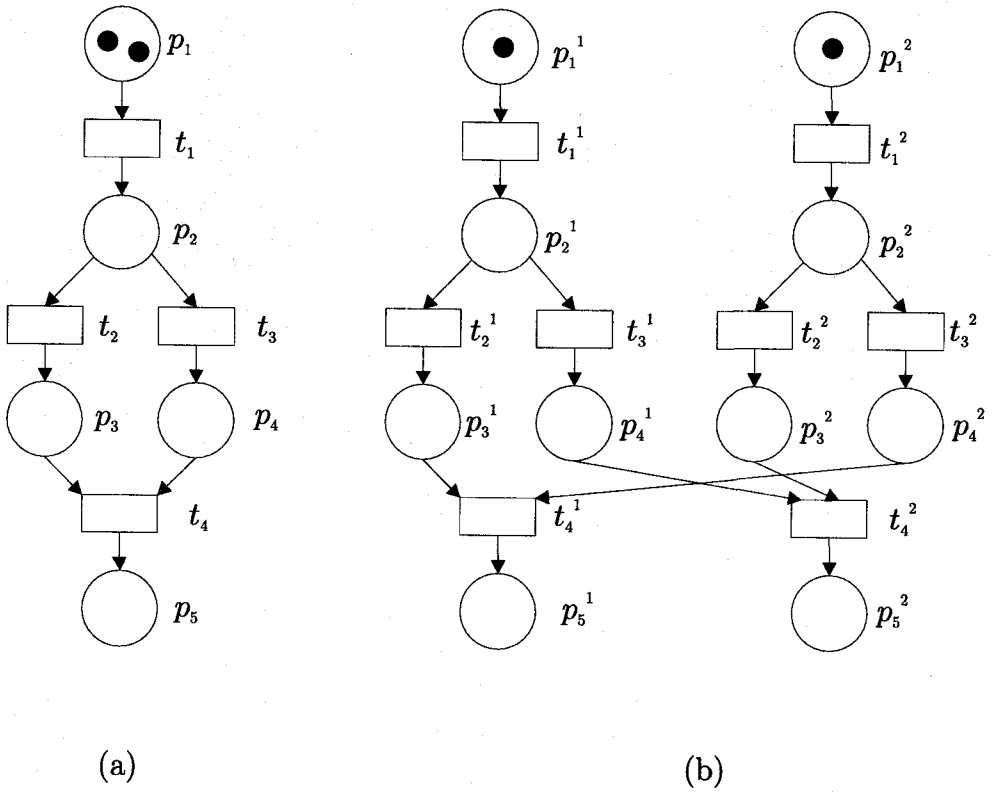

Figure 2.8: A PN (a) and its OCN (b).

OCN up to a finite prefix that preserve all information about reachable markings of the original PN. This prefix is called an *unfolding*.

A notion of cutoff points was introduced in [29]. This is necessary for truncating an OCN.

Definition 2.4 (Cutoff point) In an OCN $= (P', T', F'; M_0'; \mathcal{L})$, a transition $t \in T'$ is a *cutoff point* if and only if the following equation holds.

$$\mathcal{L}(R(N', M_0')) = \mathcal{L}(R(N', M_0')\backslash t) \tag{2.38}$$

Several conditions to find cutoff points have been proposed[29, 19, 24, 8, 36]. In Chapter 3, we discuss the condition a transition to be a cutoff point. Let Q be a maximal set of cutoff points such that $\mathcal{L}(R(N', M_0')) = \mathcal{L}(R(N', M_0')\backslash Q)$. The unfolding is defined as follows.

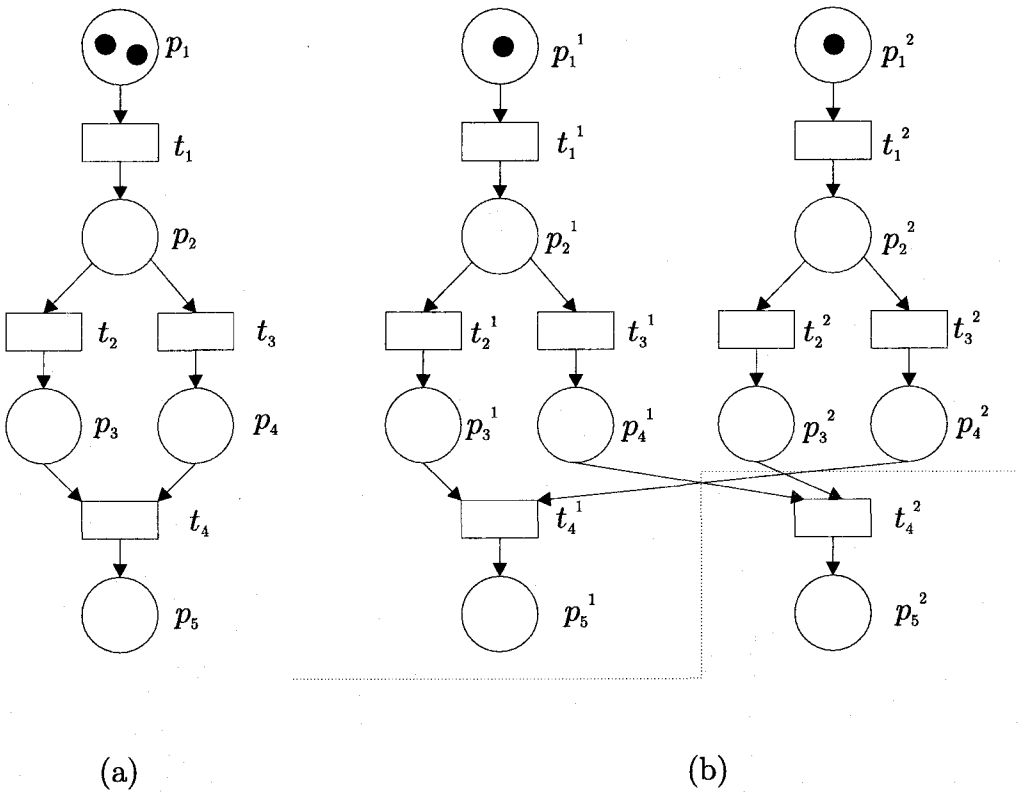

Figure 2.9: A PN (a) and its unfolding (b).

Definition 2.5 (Unfolding) An unfolding is a finite prefix of an OCN, and it is obtained by removing cutoff points in Q and all the places and transitions that succeed a cutoff point in Q.

It has already been shown that unfoldings are finite for any bounded Petri nets in [29].

In previous works[29, 19, 36], cutoff points are included in the unfolding. Strictly speaking, unfoldings should not contain cutoff points. See Figure 2.9, in this case an instance t_4^2 can be a cutoff point. If t_4^2 is a part of the unfolding and p_5^2 is not, an empty marking is reachable after firing t_1^1, t_1^2, t_2^2, t_3^1 and t_4^2, however in the original marking any empty marking is not reachable. Therefore by containing cutoff points redundant markings arises. In [8], succeeding places are contained, namely, in the case Figure 2.9 an instance p_5^2 is also a part of the unfolding. It is obvious from

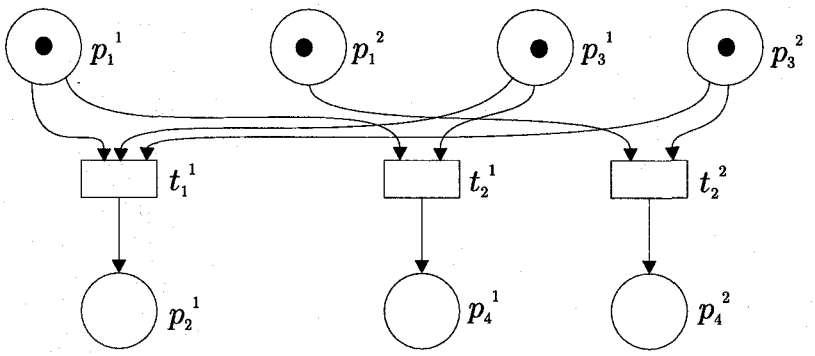

Figure 2.10: An unfolding of the PN in Figure 2.1.

Definition 2.4 that all reachable markings are preserved even if unfoldings do not contain cutoff points. In this thesis we assume that unfoldings do not contain any cutoff points. In Figure 2.9, upper side of dotted line is the unfolding in this case.

In Figure 2.10, the unfolding of the PN in Figure 2.1 is shown. The set of reachable markings of the unfolding is as follows.

$$R(N', M_0') = \{\{p_1^1, p_1^2, p_3^1, p_3^2\}, \{p_1^2, p_2^1\} \tag{2.39}$$

$$\{p_1^2, p_3^2, p_4^1\}, \{p_1^1, p_3^1, p_4^2\}, \{p_4^1, p_4^2\}\} \tag{2.40}$$

The PN is a live net, therefore its OCN extends infinitely. The unfolding, however, is finite, and all reachable markings of the PN are preserved.

In [29], a method for analyzing deadlock by using branch and bound technique is shown. In [19, 24], a method for checking safeness, boundedness and persistence by using unfoldings has been shown. In [55], a method for preventing the deadlock by using unfoldings has been shown.

Chapter 3

Constructing Petri Net Unfolding

This chapter considers to construct Petri net unfoldings that have the advantage of being smaller than unfoldings in the previous works.

3.1 Introduction

For a live Petri net, an OCN can be extended infinitely, so it is impossible to apply OCN's for the verification of Petri nets in general. Although an OCN for a live PN can be extended infinitely, it is always possible for a bounded PN to truncate the OCN up to a finite prefix that preserves all information about reachable markings of the original PN. This prefix is called the *unfolding*. McMillan first introduced a concept of an unfolding in [29]. We call the unfolding M-unfolding in this thesis. Kondratyev et al. introduced a concept of reduced unfolding in[19]. We call the unfolding K-unfolding in this thesis. The difference between M-unfolding and K-unfolding is the condition a transition to be a cutoff point.

Let us consider characteristics of the two types of unfoldings. As for M-unfolding, the unfolding happens to be redundantly large. See Figure 3.1. In the figure, a sample Petri net(a) that has a series of choices and its unfolding(b) are shown. Here, the upper side of the dashed line labeled M-unfolding is the M-unfolding. The number of markings of the Petri net in Figure 3.1(a) is n, and the number of places of the M-unfolding in Figure 3.1(b) is over 2^n.

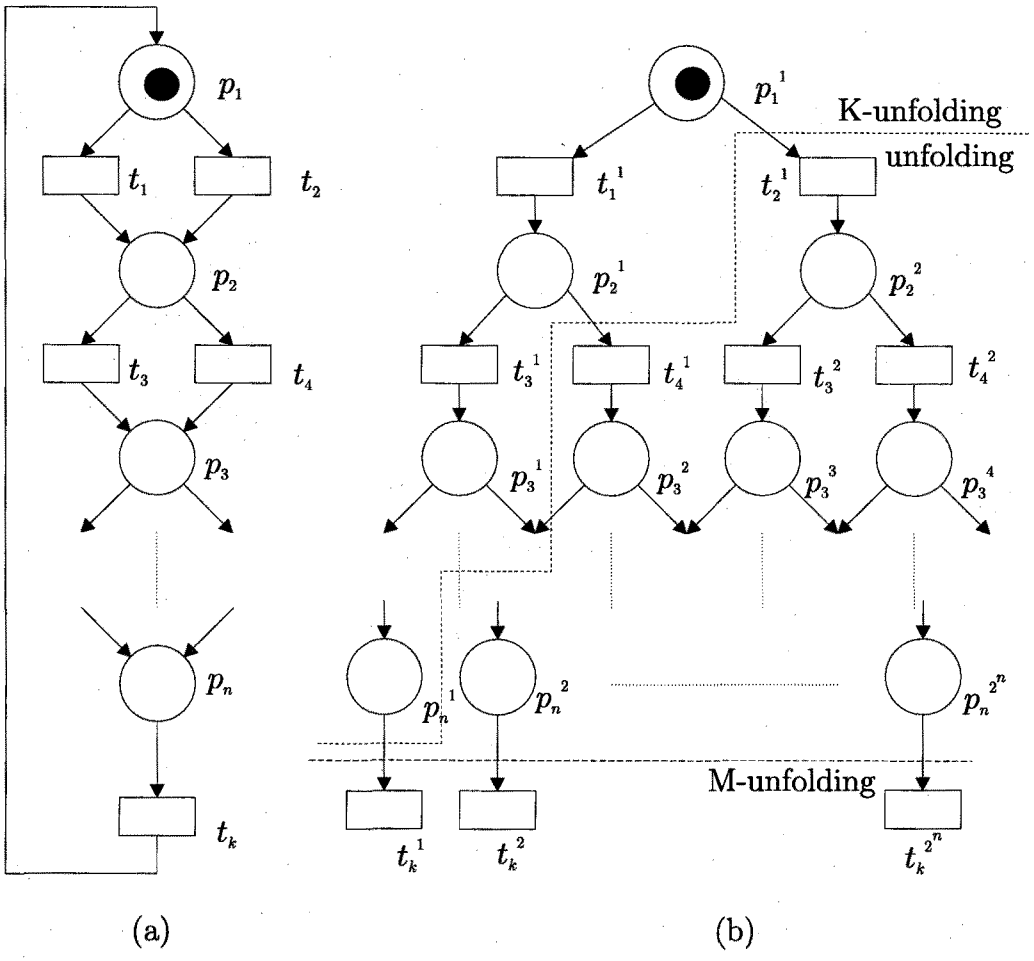

Figure 3.1: A Petri net with a series of choices (a) and its unfolding (b).

Finding a cutoff point of K-unfolding is simpler than that of M-unfolding. In K-unfolding, however, all markings are preserved only for a subclass of Petri nets. In Figure 3.2(b), upper side of dashed line labeled by K-unfolding is a K-unfolding of the Petri net of Figure 3.2(a). In this case, t_1^2 is a cutoff point. Though a marking $\{p_5\}$ is reachable, any corresponding marking does not appear in the K-unfolding.

Thus the problem is to reduce the size of M-unfolding as small as possible for general Petri nets. Let us consider the problem of reducing the size of unfoldings.

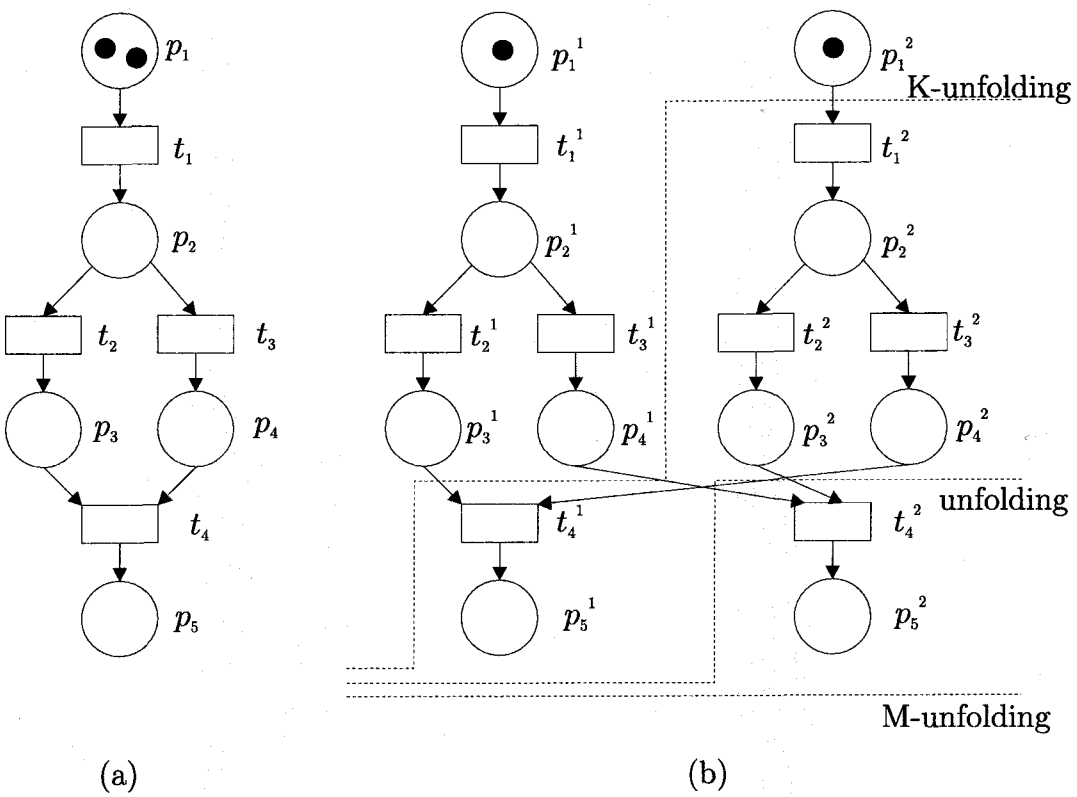

Figure 3.2: A Petri net that is not supported by K-unfolding (a) and its unfolding (b).

3.2 Preliminaries

This section explains the M-unfolding and the K-unfolding.

Notations. Let us introduce some notions on OCN's.

Definition 3.1 (Configuration) In an OCN, a subset T_c defined by transitions is a *configuration* if :

1. $\forall t \in T_c$: T_c contains all predecessors of t, and
2. $\forall t_1, t_2 \in T_c : \neg(t_1 \sharp t_2)$.

Definition 3.2 (Post-set, Final State) Let T_c be a configuration of OCN N'. The *post-set* of T_c is the set of places $p \in P'$ such that :

1. $\forall t \in T_c : p \notin \bullet t$, and
2. $\forall t \in T' - T_c : p \notin t\bullet$.

The post-set of T_c is denoted by

$$T_c \bullet . \tag{3.1}$$

A *final state* of T_c, denoted by $\mathcal{F}(T_c)$, is defined by

$$\mathcal{F}(T_c) = \mathcal{L}(T_c\bullet). \tag{3.2}$$

In an OCN, it is ensured that a configuration T_c exists for any reachable marking M in the original PN such that $\mathcal{F}(T_c) = M$, and T_c represents the set of transitions which will fire in order to reach the marking M from M_0 in the original PN.

Definition 3.3 (Local Configuration) In an OCN, a local configuration of transition $t \in T'$ is the smallest configuration that contains t and all the transitions preceding t by $\Rightarrow$. A local configuration of transition t is denoted by

$$(\Rightarrow t). \tag{3.3}$$

A local configuration $(\Rightarrow t)$ is the smallest set of transitions including t that must be fired for t to fire.

Definition 3.4 (Basic Marking) In an OCN, a *basic marking* of a transition $t \in T'$ is defined as follows :

$$BM(t) = \mathcal{F}((\Rightarrow t)). \tag{3.4}$$

As a special case of the basic marking, we define a basic marking of *null* transition as follows :

$$BM() = M_0. \tag{3.5}$$

The null transition is used when we want to represent the initial marking M_0 by the basic marking. Without loss of generality, we can assume that the null transition precedes all transitions in an OCN.

M-unfolding. In the M-unfolding, for a condition that a transition is to be a cutoff point, the following condition is used.

Condition 3.1 For a transition $t \in T'$ in an OCN, a condition is defined as follows:

$$\exists t' \in T' : t' \Rightarrow t \quad \text{and} \quad BM(t') = BM(t). \tag{3.6}$$

In the case of Figure 3.1, the basic marking of the null transition and transitions t_k^i become

$$BM() = \{p_1\}, \quad \text{and} \tag{3.7}$$
$$BM(t_k^i) = \{p_1\} \qquad i = 1 \cdots 2^n, \tag{3.8}$$

and the null transition precedes all transitions in the OCN, therefore transitions t_k^i are cutoff points. In Figure 3.1(b), upper side of dashed line labeled by M-unfolding is the M-unfolding of the Petri net of Figure 3.1(a).

In the case of Figure 3.2, there exists no transition that satisfies the Condition 3.1. In Figure 3.2(b), upper side of dashed line labeled by M-unfolding is the M-unfolding of the Petri net of Figure 3.2(a).

K-unfolding. In the K-unfolding, for a condition that a transition is to be a cutoff point, the following condition is used.

Condition 3.2 For a transition $t \in T'$ in an OCN, a condition is defined as follows:

$$\exists t' \in T' : BM(t') = BM(t). \tag{3.9}$$

In the case of Figure 3.1, the basic marking of the null transition and transitions t_k^1 become

$$BM() = \{p_1\}, \quad \text{and} \tag{3.10}$$
$$BM(t_k^1) = \{p_1\}, \tag{3.11}$$

therefore transition t_k^1 is a cutoff point, and basic markings of transitions t_{2i-1}^1 and t_{2i}^1 for $i = 1 \cdots n-1$ become

$$BM(t_{2i-1}^1) = \{p_{i+1}\}, \quad \text{and} \tag{3.12}$$

$$BM(t_{2i}^1) = \{p_{i+1}\}, \tag{3.13}$$

therefore transitions t_{2i}^1 are cutoff points. In Figure 3.1(b), upper side of dashed line labeled by K-unfolding is the K-unfolding of the Petri net of Figure 3.1(a).

In the case of Figure 3.2, basic markings of transitions t_1^1 and t_1^2 become

$$BM(t_1^1) = \{p_2\}, \quad \text{and} \tag{3.14}$$

$$BM(t_1^2) = \{p_2\}, \tag{3.15}$$

therefore transition t_1^2 is a cutoff point. In Figure 3.2(b), upper side of dashed line labeled by K-unfolding is the K-unfolding of the Petri net of Figure 3.2(a).

3.3 Improved Algorithm to Construct Unfolding

This section considers the problem of reducing the size of unfoldings.

3.3.1 New Condition for Cutoff Points

We can use Condition 3.1 as a condition that a transition is a cutoff point, for general Petri nets. Thus we can obtain the following lemma.

Lemma 3.1 In an OCN $= (P', T', F'; M_0'; \mathcal{L})$, a transition $t \in T'$ is a cutoff point when it satisfies Condition 3.1.

Proof: It has already proved in [29]. ∎

We introduce another condition as follows.

Condition 3.3 For a transition $t \in T'$ in an OCN, a condition is defined as follows:

$\exists t' \in T'$ such that $t' \sharp t$,

$$BM(t') = BM(t), \text{ and}$$

There exists no cutoff points which is concurrent with t'. (3.16)

As for Condition 3.3, we can obtain the following lemma.

Lemma 3.2 In an OCN $= (P', T', F'; M'_0; \mathcal{L})$, a transition $t \in T'$ is a cutoff point when it satisfies Condition 3.3.

Proof: In an OCN, we can obtain the following equation without loss of generality.

$$R(N', M'_0) = R(N', M'_0)\backslash t \cup R(N', (\Rightarrow t)\bullet) \tag{3.17}$$

Since $t' \sharp t$, disabled transition t does not affect the firability of successors of t'. Therefore we can obtain the following equation.

$$R(N', (\Rightarrow t')\bullet) \subset R(N', M'_0)\backslash t \tag{3.18}$$

Since $BM(t') = BM(t)$ and there exist no cutoff points which are concurrent with t', thus

$$\begin{aligned} \mathcal{L}(R(N', (\Rightarrow t)\bullet)) &\subseteq \mathcal{L}(R(N', (\Rightarrow t')\bullet)) \\ &\subset \mathcal{L}(R(N', M'_0)\backslash t). \qquad \text{(by (3.18))} \end{aligned} \tag{3.19}$$

Consequently,

$$\begin{aligned} \mathcal{L}(R(N', M'_0)) &= \mathcal{L}(R(N', M'_0)\backslash t \cup R(N', (\Rightarrow t)\bullet)) \qquad \text{(by (3.17))} \\ &= \mathcal{L}(R(N', M'_0)\backslash t) \cup \mathcal{L}(R(N', (\Rightarrow t)\bullet)) \\ &= \mathcal{L}(R(N', M'_0)\backslash t), \qquad \text{(by (3.19))} \end{aligned} \tag{3.20}$$

therefore the transition t is a cutoff point. ∎

In this thesis, we adopt Condition 3.1 and Condition 3.3 as conditions for a transition to be a cutoff point. Since Condition 3.3 is independent of the class of Petri nets, we can use the condition for general Petri nets. By adding another condition(Condition 3.3), the size of unfolding becomes smaller than or equal to that of M-unfolding. Let us confirm with examples in Figure 3.1 and Figure 3.2.

In the case of Figure 3.1, the basic marking of the null transition and transitions t_k^1 become

$$BM() = \{p_1\}, \quad \text{and} \tag{3.21}$$
$$BM(t_k^1) = \{p_1\}, \tag{3.22}$$

and the null transition precedes all the transition in the OCN, therefore transition t_k^1 is a cutoff point. Basic markings of transitions t_{2i-1}^1 and t_{2i}^1 for $i = 1 \cdots n-1$ become

$$BM(t_{2i-1}^1) = \{p_{i+1}\}, \quad \text{and} \tag{3.23}$$
$$BM(t_{2i}^1) = \{p_{i+1}\}, \tag{3.24}$$

transitions t_{2i-1}^1 and t_{2i}^1 for $i = 1 \cdots n-1$ are in conflict relation, and no cutoff point exists, which is in concurrent with t_{2i}^1, therefore transitions t_{2i}^1 are cutoff points. In Figure 3.1(b), upper side of dashed line labeled by unfolding is the unfolding of the Petri net of Figure 3.1(a).

In the case of Figure 3.2, basic markings of transitions t_4^1 and t_4^2 become

$$BM(t_4^1) = \{p_5\}, \quad \text{and} \tag{3.25}$$
$$BM(t_4^2) = \{p_5\}, \tag{3.26}$$

transitions t_4^1 and t_4^2 are in conflict relation, and no cutoff point exists, which is in concurrent with t_4^2, therefore transition t_4^2 is a cutoff point. In this case, transition t_1^2 cannot be a cutoff point, because transitions t_1^1 and t_1^2 are in concurrent relation. In Figure 3.2(b), upper side of dashed line labeled by unfolding is the unfolding of the Petri net of Figure 3.2(a).

As for the conditions that a transition is to be a cutoff point, a similar result is reported by Kondratyev et al. in [24]. Esparza et al. define an adequate total order on the configuration in [8], and introduce quite a difference condition.

3.3.2 Improved Algorithm

This subsection discusses an algorithm to construct unfoldings.

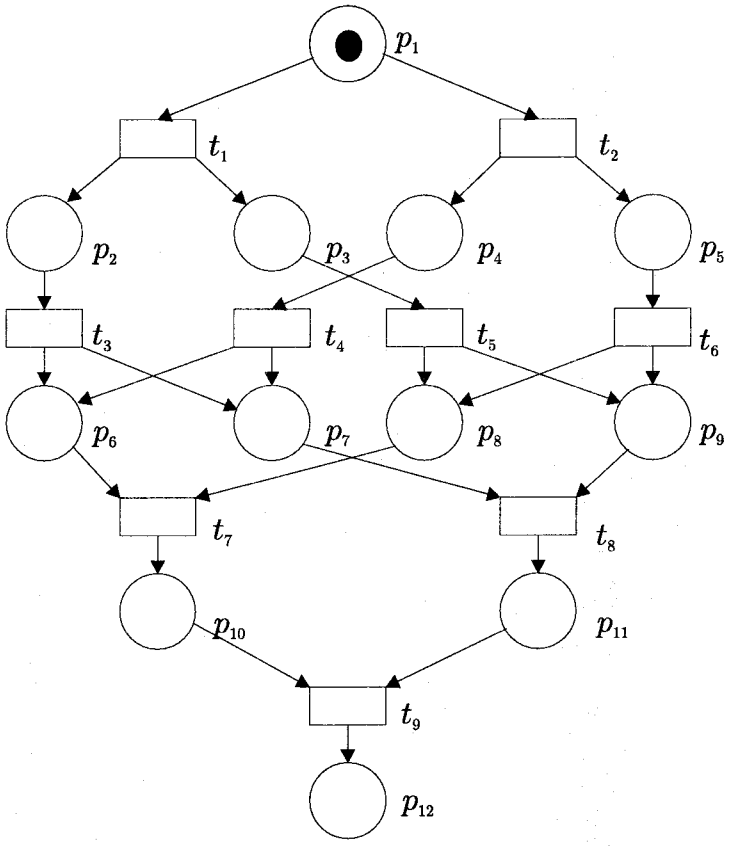

Figure 3.3: A safe Petri net.

Explaining the algorithm to construct unfoldings simply, it is similar to that to construct OCN's, and only the difference is that cutoff points and their succeeding places are not copied into the OCN. Then the output OCN is the unfolding. Therefore, when a transition is copied, it is checked whether a transition is a cutoff point or not.

Here, another problem, the order of coping transitions, comes into question. See Figure 3.3 and Figure 3.4. The OCN of the Petri net in Figure 3.3 is shown in Figure 3.4. Let us construct the unfolding from the Petri net. Assume that transitions t_1^1, t_2^1, t_3^1, t_4^1, t_5^1, t_6^1, t_7^1, t_7^2, t_8^1, t_8^2, t_9^1 and t_9^2 are copied in this order. In this case, transitions t_7^2 and t_8^2 are cutoff points, thus the upper side of the dashed line labeled case1 is the unfolding. Since transitions t_7^2 and t_8^2 are not copied, transition

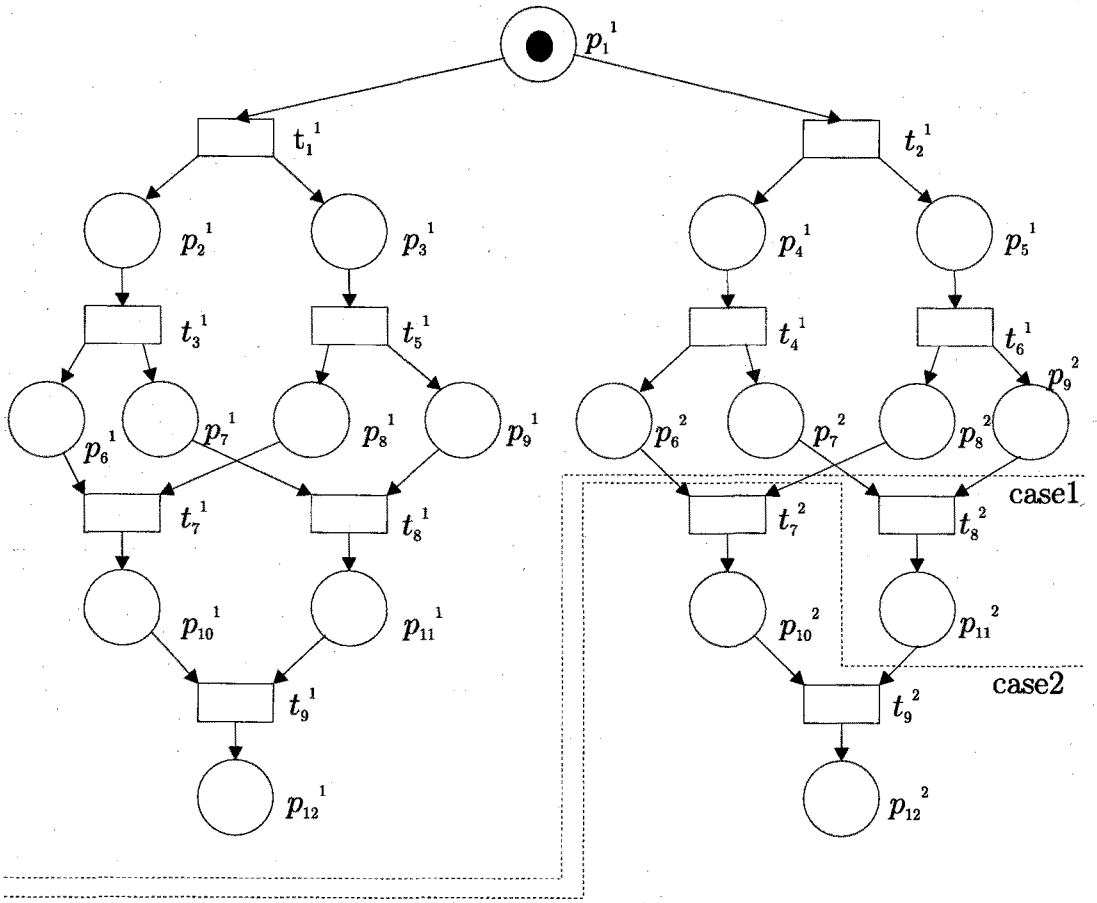

Figure 3.4: An OCN of the net in Figure3.3.

t_9^2 will not appear in the unfolding. If the order of transitions t_8^2 and t_8^1 are swapped, only the transition t_7^2 is the cutoff point. Transition t_8^1 can not be a cutoff point, since, for the corresponding transition t_8^2, there exists a concurrent transition t_7^2 that is a cutoff point. In this case, the upper side of the dashed line labeled case2 is the unfolding. This indicates that the size of unfoldings depends on the order of coping transitions.

In Figure 3.5, an algorithm to construct an unfolding from a given PN is shown. In Algorithm 3.1, union of transition t in the Petri net and the set $P'_{\bullet t}$ of places in the unfolding are stored in the *stack_of_union*, where $P'_{\bullet t}$ is the set of input places of the copy t' of t.

Algorithm 3.1 *Construction of an unfolding from a PN*

input: a Petri net PN.

output: an unfolding.

begin

Copy all places $p \in M_0$ into the unfolding.;

/* If M_0 is not safe, make as many copies as the number of appearances in M_0.*/

foreach $t_1 \in T$ and $P'_{\bullet t_1} \subseteq P'$, with $\mathcal{L}(P'_{\bullet t_1}) = \bullet t_1$ and $\forall p_1, p_2 \in P'_{\bullet t_1} : p_1 // p_2$

Push union of t_1 and $P'_{\bullet t_1}$ to *stack_of_union*.;

endforeach

while *stack_of_union* is not empty **do**

Pop the first union of t_1 and $P'_{\bullet t_1}$ from *stack_of_union*.;

if t_1 is not a cutoff point **then**

Make a copy of t_1 in the unfolding. Call it t'_1.;

Draw an arc from each place in $P'_{\bullet t_1}$ to t'_1.;

For each place in $t_1\bullet$, make a copy in the unfolding and draw an arc from t'_1 to it. Denote the set of copies by $P'_{t_1\bullet}$.;

foreach t_2 and $P'_{\bullet t_2}$, such that t_2 is enabled on $P'_{\bullet t_2}$ and $P'_{t_1\bullet} \cap P'_{\bullet t_2} \neq \emptyset$

if the same union of t_2 and $P'_{\bullet t_2}$ is not in *stack_of_union* **then**

Push the union to *stack_of_union*.;

endif

endforeach

endif

endwhile

end

Figure 3.5: An Algorithm to construct an unfolding.

From Figure 3.4, we can find two groups of transitions,

$$\text{GROUP}_1 = \{t_1^1, t_3^1, t_5^1, t_7^1, t_8^1, t_9^1\}, \text{and} \tag{3.27}$$

$$\text{GROUP}_2 = \{t_2^1, t_4^1, t_6^1, t_7^2, t_8^2, t_9^2\}. \tag{3.28}$$

For any pair of $t_1 \in \text{GROUP}_1$ and $t_2 \in \text{GROUP}_2$, t_1 and t_2 are in conflict relation.

Condition 3.3 causes the order sensitive feature of constructing unfoldings. To avoid redundant unfolding construction, transitions in GROUP_2 should be copied after transitions in GROUP_1 is copied. Algorithm 3.1 solves the problem by using a stack *stack_of_union*. In the case of Petri net in Figure 3.3, the *stack_of_union* shifts as follows.

transitions in *stack_of_union* :	$[t_1, t_2]$	$\Rightarrow$	copy t_1 as t_1^1	$\Rightarrow$
	$[t_3, t_5, t_2]$	$\Rightarrow$	copy t_3 as t_3^1	$\Rightarrow$
	$[t_5, t_2]$	$\Rightarrow$	copy t_5 as t_5^1	$\Rightarrow$
	$[t_7, t_8, t_2]$	$\Rightarrow$	copy t_7 as t_7^1	$\Rightarrow$
	$[t_8, t_2]$	$\Rightarrow$	copy t_8 as t_8^1	$\Rightarrow$
	$[t_9, t_2]$	$\Rightarrow$	copy t_9 as t_9^1	$\Rightarrow$
	$[t_2]$	$\Rightarrow$	copy t_2 as t_2^1	$\Rightarrow$
	$[t_4, t_6]$	$\Rightarrow$	copy t_4 as t_4^1	$\Rightarrow$
	$[t_6]$	$\Rightarrow$	copy t_6 as t_6^1	$\Rightarrow$
	$[t_7, t_8]$	$\Rightarrow$	t_7 is not copied	$\Rightarrow$
	$[t_8]$	$\Rightarrow$	t_8 is not copied	$\Rightarrow$
	$[]$			

3.4 Experimental Results

This section gives experimental results. McMillan's unfolding and the new proposed unfolding are compared on size and CPU time.

In Table 3.1, experimental results are shown. In the row "Petri net", data of Petri net are shown: "net" is the name of Petri net, "$|P|$" is the number of places and "$|T|$" is the number of transitions. In the row "McMillan's Unfolding", data of M-Unfolding are shown. In the row "New Unfolding", data of unfolding are shown: "$|P'|$" is the number of places, "$|T'|$" is the number of transitions and "time" is the CPU time to construct the unfolding. All the times have been measured on a SUN SPARCstation 4 with 32MB main memory. These examples are taken from

Table 3.1: Experimental Results

Petri net			McMillan's Unfolding			New Unfolding		
net	$\|P\|$	$\|T\|$	$\|P'\|$	$\|T'\|$	time	$\|P'\|$	$\|T'\|$	time
pla	10	8	14	9	0.0	14	9	0.0
rw	4	4	10	6	0.0	7	3	0.0
rw10	4	4	120	100	0.6	30	10	0.0
vme	17	17	39	35	0.0	22	19	0.0
vme2	38	36	252	220	1.1	78	68	0.1
vme3	57	53	678	586	9.5	117	101	0.3

[40, 46, 54].

Results show that

- by adding Condition 3.3, reduction of unfoldings is achieved, and
- as a result, time to construct unfoldings is shortened.

3.5 Concluding Remarks

Construction of unfoldings have been studied. Previous proposed unfoldings called M-unfolding and K-unfolding were compared first, some problems on these unfoldings were pointed out. The problem is how to reduce the size of M-unfolding without loss of generality.

Next, a new condition for a transition was proposed, and it was proved that the condition is a condition for a transition to be a cutoff point. The condition, however, depends on the order of coping transitions, to avoid the ordering problem an improved algorithm to construct unfoldings was proposed. The algorithm solves the ordering problem by using a stack. By using a condition that was proposed by McMillan and the new condition together, the reduction of unfoldings was successful in term of constructing time as well as required space. Experimental results show a significant effectiveness of the proposed method.

Chapter 4

Verifying Petri Nets with Unfoldings

4.1 Introduction

Petri nets are widely recognized as a powerful model for discrete event systems characterized by asynchronous and concurrent behavior. Petri nets have graphical and mathematical features. Graphical feature provides with an environment to design and to comprehend discrete event systems. Mathematical feature provides an analysis power for several properties of such systems.

In verifying Petri nets, several methods have been proposed, for example, a reachability(coverability) graph method[25, 28, 40], a matrix equation approach[40], reduction or decomposition techniques[40, 26, 44], a symbolic model method[47, 12] and an unfolding method[29, 19, 8, 36]. Though the reachability(coverability) graph method is a strong verification method, it is applicable only for small nets because of state space explosion. The matrix equation approach and reduction or decomposition techniques have limitation on applicable classes. The symbolic model method can deal with huge nets by using Binary Decision Diagrams (BDD's)[2]. The BDD traversal can be executed efficiently only if the property to be verified can be formulated by characteristic predicate. Therefore this method is effectively applied to systems modeled by k-bounded nets and finite capacity nets. On the other hand,

the unfolding method was introduced to avoid generating the reachability graph for a general Petri nets[29, 19, 36].

This chapter considers a computational problem of reachability of finite state systems modeled by bounded nets, and discusses the upper bound problem, which can be defined as a problem of finding upper bound on tokens on a place in reachable markings. This chapter clarifies the relation between these problems and an unfolding, and provides a method to verify these problems.

4.2 Concurrent-Relation Graph

The problems are closely related to the concurrent relation in unfoldings. Let us introduce a notion of concurrent-relation graph.

Definition 4.1 (Concurrent-Relation Graph) For a PN $= (P, T, F, W; M_0)$ and its unfolding $= (P', T', F'; M_0'; \mathcal{L})$, a concurrent-relation graph CG is an undirected graph and denoted by a two-tuple

$$\mathrm{CG} = (V, E), \tag{4.1}$$

where

$$V \subseteq P' \cup T' \tag{4.2}$$

is the set of vertices and

$$E = \{(v_1, v_2) | v_1, v_2 \in V, v_1 // v_2\} \tag{4.3}$$

is the set of edges.

When the set of vertices are given by the following equation

$$V_x = \{x' | x \in P \cup T, \mathcal{L}(x') = x\}, \tag{4.4}$$

a subgraph defined by the concurrent relation with V_x is denoted by

$$\mathrm{CG}\backslash x. \tag{4.5}$$

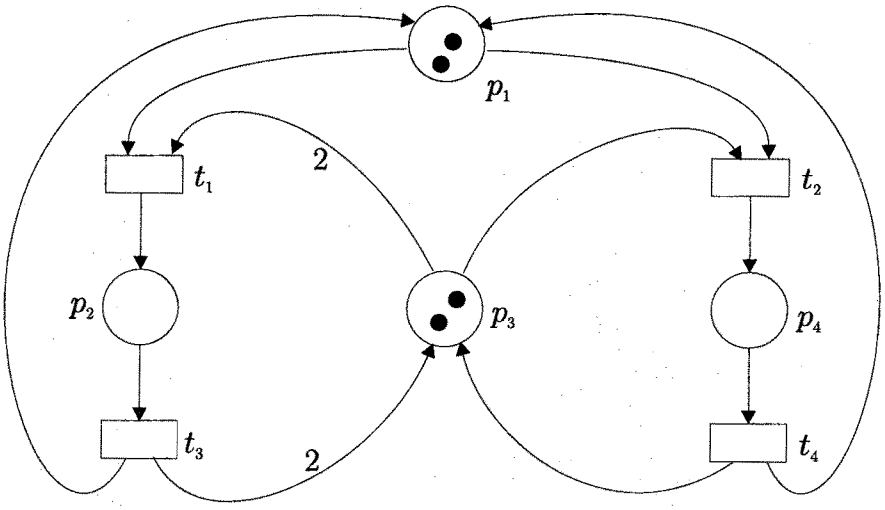

Figure 4.1: A readers-writers system.

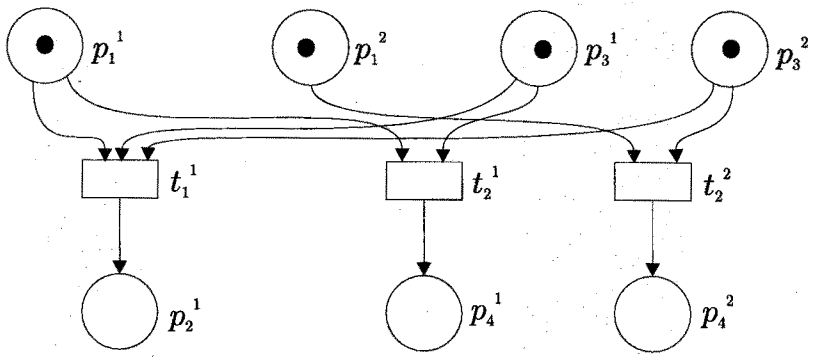

Figure 4.2: The unfolding of the Petri net in Figure 4.1.

This graph is called a concurrent-relation graph induced by a node x. Similarly when the set of vertices are given by

$$V_X = \{x' | X \subseteq P \cup T, \mathcal{L}(x') \in X\}, \tag{4.6}$$

a subgraph induced by V_X is denoted by

$$\mathrm{CG}\backslash X, \tag{4.7}$$

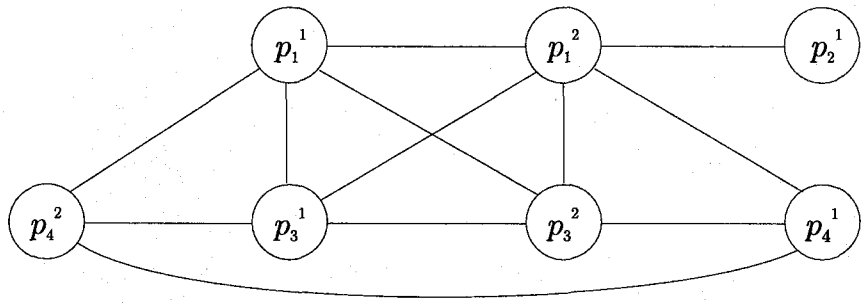

Figure 4.3: CG\P of the unfolding in Figure 4.2.

and is called a concurrent-relation graph induced by a set X. As an example, a concurrent-relation graph induced by a set P for the unfolding of Figure 4.2 is shown in Figure 4.3.

A CG is a graph representation of concurrent relations between instances in an unfolding. A complete subgraph of the CG represents the set of instances in which any pair of instances is in concurrent relation. Especially maximal subgraphs of complete subgraphs are important. Let us introduce notions of maximal complete subgraph and maximum complete subgraph.

Definition 4.2 (Maximal Complete Subgraph) For a graph $\mathrm{G} = (V, E)$, a subgraph is called a maximal complete subgraph (or maximal clique) and denoted by

$$\mathrm{G}^K = (V^K, E^K), \tag{4.8}$$

when

- G^K is a complete graph, and
- there is no complete graph $\mathrm{G}^{K\prime} = (V^{K\prime}, E^{K\prime})$ such that $V^K \subset V^{K\prime}$.

Definition 4.3 (Maximum Complete Subgraph) For a graph $\mathrm{G} = (V, E)$, a subgraph is called a maximum complete subgraph (or maximum clique) and denoted by

$$\mathrm{G}^{K_{\max}} = (V^{K_{\max}}, E^{K_{\max}}), \tag{4.9}$$

when

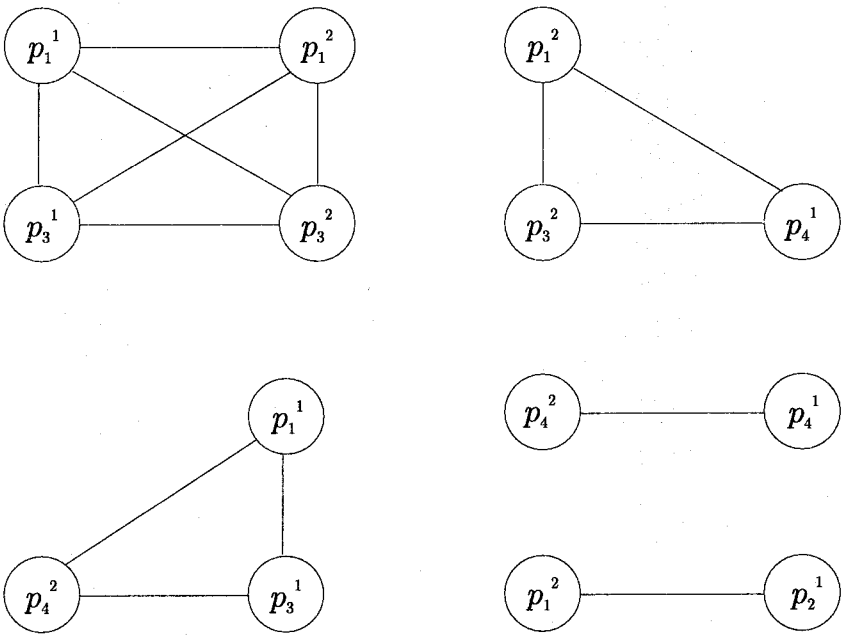

Figure 4.4: Maximal complete subgraphs of the CG in Figure 4.3.

- $G^{K_{max}}$ is a maximal complete graph, and
- the number of vertices is maximum.

In Figure 4.4, maximal complete subgraphs of the CG of Figure 4.3 are shown. There are five maximal complete subgraphs in the CG. The complete graph at the upper left in Figure 4.4 is the maximum complete subgraph.

In general, we must consider all combinations of vertices to find all maximal complete subgraphs. Finding maximal complete subgraphs of an arbitrary graph is a well-known NP-complete problem. Following lemmas show that maximal complete subgraphs are preserved after dividing the graph at any articulation point.

Lemma 4.1 For a graph G, the set of maximal complete subgraphs is invariant when we divide the graph G at an articulation point into non-separable subgraphs.

Proof: Let a graph G be divided into two non-separable subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ at a point $v \in V$. For each vertex $v_1(\neq v) \in V_1$ and $v_2(\neq v) \in V_2$,

they are not adjacent. Therefore there is no complete subgraph induced by V where $V \subseteq V_1 \cup V_2$, $V \not\subseteq V_1$ and $V \not\subseteq V_2$. Consequently, the set of maximal complete subgraphs is invariant. ∎

Let us introduce another division method.

Definition 4.4 For a graph $G = (V, E)$, a subgraph G_x with respect to a vertex $x \in V$ and its complementary subgraph $G_{\overline{x}}$ are defined as follows.

$$G_x = (V_x, E_x), \tag{4.10}$$

where

$$V_x = \{x, \text{adjacent vertices of } x\}, \text{and} \tag{4.11}$$
$$E_x = \{(x, y) | x, y \in V_x, (x, y) \in E\}. \tag{4.12}$$

$$G_{\overline{x}} = (V_{\overline{x}}, E_{\overline{x}}), \tag{4.13}$$

where

$$\begin{aligned} V_{\overline{x}} =& V - \{x, \text{ adjacent vertices of } x \\ & \text{whose all incident edges are in } E_x\}, \text{ and} \end{aligned} \tag{4.14}$$

$$\begin{aligned} E_{\overline{x}} =& \{(v_1, v_2) | v_1, v_2 \in V_{\overline{x}}, (v_1, v_2) \in E\} \\ & - \{(v', v'') | v', v'' \in V_x, \nexists y \in V_{\overline{x}} \text{ such that } (v', y), (v'', y) \in E\}. \end{aligned} \tag{4.15}$$

Figure 4.5 shows subgraphs (a)$G_{p_3^1}$ and (b)$G_{\overline{p_3^1}}$ of the graph in Figure 4.3. Here, a vertex p_1^1 is removed from $V_{\overline{p_3^1}}$, because all incident edges are in $E_{p_3^1}$. Figure 4.6 shows subgraphs $G_{p_3^2}$ and $G_{\overline{p_3^2}}$ of the graph in Figure 4.5(b). Here, an edge (p_1^2, p_4^1) is removed from $E_{\overline{p_3^2}}$, because there exists no vertex $y \in V_{\overline{p_3^2}}$ such that $(p_1^2, y), (p_4^1, y) \in E$.

Lemma 4.2 For a graph $G = (V, E)$, the set of maximal complete subgraphs is invariant when we divide the graph G into subgraphs G_x and $G_{\overline{x}}$ with respect to a vertex $x \in V$.

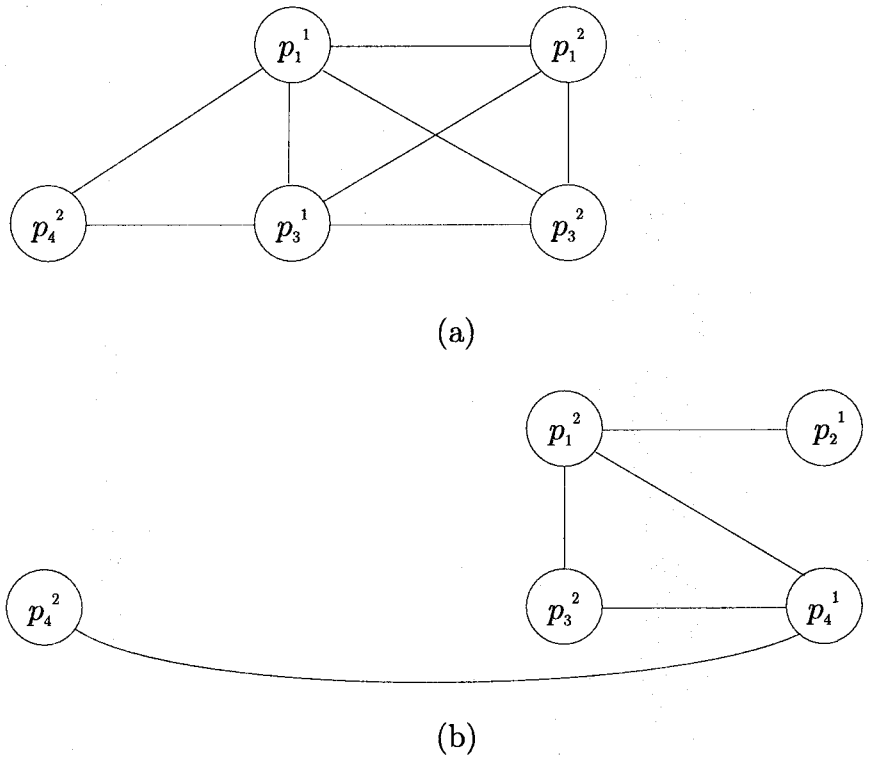

Figure 4.5: Components with respect to p_3^1 of the CG in Figure 4.3. (a)$G_{p_3^1}$ and (b)$G_{\overline{p_3^1}}$.

Proof: We claim that all maximal complete subgraphs are preserved.
(1) It is trivial that each maximal complete subgraph in G whose vertices are in V_x is contained in G_x.
(2) Let v' and v'' be vertices which was removed from $E_{\overline{x}}$ in equation(4.15). With respect to v' and v'', we can obtain following two facts.

- A subgraph induced by $\{v', v'', x\}$ is a complete graph.
- Even if the edge (v', v'') is in $G_{\overline{x}}$, it is a maximal complete subgraph itself, because the vertices do not have any common adjacent vertex.

Therefore a maximal complete graph containing a edge (v', v'') is contained in G_x, and no maximal complete subgraph disappears from $G_{\overline{x}}$ by removing the edge

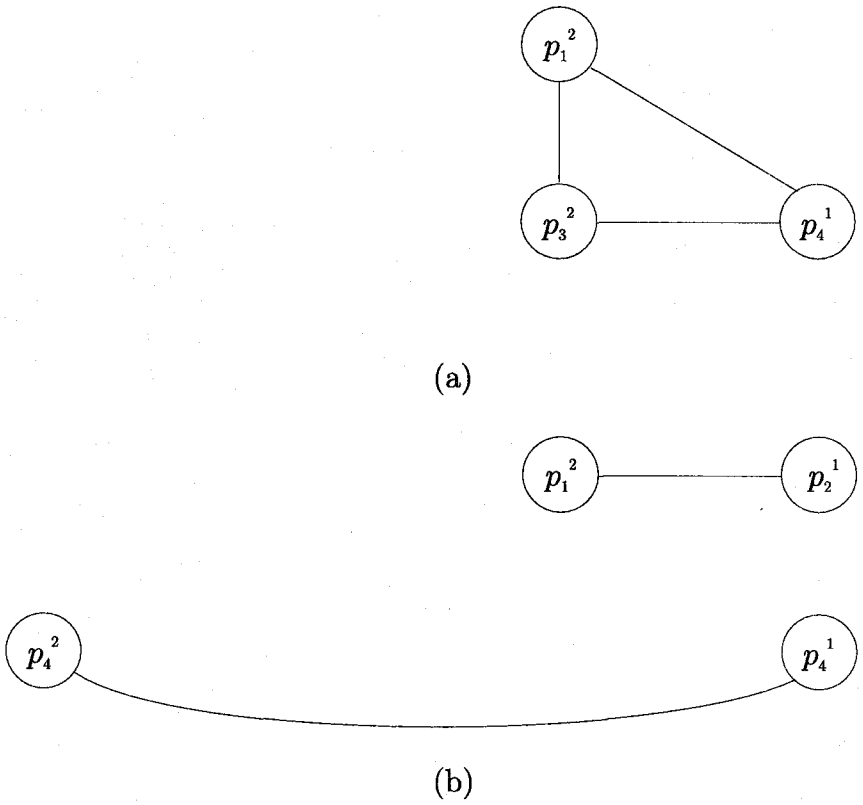

Figure 4.6: Components with respect to p_3^2 of the CG in Figure 4.5(b). (a)$G_{p_3^2}$ and (b)$G_{\overline{p_3^2}}$.

(v', v''). Therefore each maximal complete subgraph in G whose vertices are in $V_{\overline{x}}$ is contained in $G_{\overline{x}}$.
(3) There exist no complete subgraph G' = (V', E'), where $V' \subseteq V$, $V' \not\subseteq V_x$ and $V' \not\subseteq V_{\overline{x}}$, because nodes $v_1 \in V' \cap \overline{V_x}$ and $v_2 \in V' \cap \overline{V_{\overline{x}}}$ are not adjacent.
Therefore all maximal complete subgraphs are preserved.

We claim that no additional maximal complete subgraph arises. (1) From the definition of G_x, each maximal complete subgraph in G_x contains x. In G_x, all adjacent vertices with x and all incident edges between these vertices in G are contained. Each maximal complete subgraph in G_x is thus a maximal complete subgraph in G.
(2) From the definition of $G_{\overline{x}}$, each maximal complete subgraph in $G_{\overline{x}}$ contains a

Algorithm 4.1 *Divide a Graph into Maximal Complete Subgraphs*

input: A graph G.
output: Maximal Complete Subgraphs.
begin
Divide G into non-separable subgraphs.;
Append non-separable subgraphs to *list_of_graphs*.;
while *list_of_graphs* $\neq \emptyset$ **do**
Take the first graph G' from *list_of_graphs*.;
if Is G' a complete graph? **then**
Append G' to *list_of_max_complete_subgraphs*.;
break
endif
Divide G' into subgraphs G'_x and $G'_{\overline{x}}$.;
/* Don't select a vertex as x, which is adjacent with all other vertices.*/
Divide G'_x and $G'_{\overline{x}}$ into non-separable subgraphs.;
Append non-separable subgraphs to *list_of_graphs*.;
endwhile
end

Figure 4.7: An algorithm to divide a graph into maximal complete subgraphs.

vertex that is not in V_x. In $G_{\overline{x}}$, all adjacent vertices with the vertex and all incident edges to the vertex are contained, therefore each maximal complete subgraph in $G_{\overline{x}}$ is a maximal complete subgraph also in G.
Therefore no additional maximal complete subgraph arises. ■

In Figure 4.7, an algorithm to divide a graph into maximal complete subgraphs is shown.

4.3 Reachability Verification with Unfoldings

This section discusses the reachability problem and reachability verification.

4.3.1 Reachability Problem

Reachability is a fundamental basis for studying the dynamic properties of discrete event systems.

The reachability problem for Petri nets is formulated as follows.

Problem 4.1 (Reachability Problem) For a given PN and a marking M, is M reachable from M_0?

The reachability problem is decidable[25, 28] although it takes exponential space and time to verify in the general case. Therefore efficient algorithm for the reachability problem is required.

This thesis deals with only bounded Petri nets.

4.3.2 Unfoldings and the Reachability Problem

For the reachability problem, we obtained the following theorem.

Theorem 4.1 For a PN $= (P, T, F, W; M_0)$ and its unfolding $= (P', T', F'; M'_0; \mathcal{L})$, a marking M is reachable from M_0 if and only if in CG$\backslash P$ there exists a maximal complete subgraph (V^K, E^K) such that $M = \mathcal{L}(V^K)$.

Proof: ($\Rightarrow$) Since M is reachable from M_0, there exists a configuration T_c such that

$$\mathcal{F}(T_c) = M. \tag{4.16}$$

Let us assume that

$$p_1, p_2 \in T_c\bullet, \tag{4.17}$$

$$t_1, t_2 \in T_c, \text{ and} \tag{4.18}$$

$$(t_1, p_1), (t_2, p_2) \in F'. \tag{4.19}$$

Since $t_1, t_2 \in T_c$,

$$t_1 // t_2, \text{ or} \tag{4.20}$$

$$t_1 \Rightarrow t_2 (\text{or } t_2 \Rightarrow t_1). \tag{4.21}$$

If $t_1 // t_2$, then $p_1 // p_2$ since in an unfolding each place has at most one input transition. Else, without loss of generality we can assume $t_1 \Rightarrow t_2$, from the definition of the post-set and the configuration,

$$\neg(p_1 \Rightarrow t_2). \tag{4.22}$$

Since in an unfolding each place has at most one input transition,

$$p_1 // p_2. \tag{4.23}$$

Consequently, there exists a complete subgraph induced by $T_c\bullet$ in CG$\backslash P$.

Let us show that the subgraph is maximal. Assume that

$$\exists p \in P', p \notin T_c\bullet : \text{ a subgraph induced by } T_c \bullet \cup p \text{ is a complete subgraph,} \tag{4.24}$$

namely p is concurrent with any place in $T_c\bullet$.

(1) When the input transition of p is in T_c. From $p \notin T_c\bullet$ and Definition 3.2, there exists a transition t such that

$$t \in p\bullet \qquad \text{and} \qquad t \in T_c. \tag{4.25}$$

That implies

$$\exists p' \in T_c\bullet : \text{ such that } p \Rightarrow p'. \tag{4.26}$$

(2) On the other hand, when the input transition of p is not in T_c. (a) If all the preceding transitions of p are not in T_c, from Definition 3.2, all source places which precede p must be in $T_c\bullet$, i.e.,

$$\forall p_0 \in M_0' : \text{if } p_0 \Rightarrow p \text{ then } p_0 \in T_c \bullet . \tag{4.27}$$

(b) When some preceding transitions of p are in T_c, there exist transitions t', t'' and a place p'' such that

$$t' \Rightarrow p'' \Rightarrow t'' \Rightarrow p, t' \in T_c \text{ and } t'' \notin T_c. \tag{4.28}$$

If $p'' \in T_c\bullet$, then

$$\exists p'' \in T_c\bullet : p'' \Rightarrow p. \tag{4.29}$$

If $p'' \notin T_c\bullet$, then

$$\exists p''' \in T_c\bullet : p'' \Rightarrow p''' \text{ and } p'''\sharp p''. \tag{4.30}$$

All these ((4.26), (4.27), (4.29) and (4.30)) contradict to p is concurrent with any place in $T_c\bullet$(assumption (4.24)). Therefore a complete graph induced by $T_c\bullet$ is maximal.

($\Leftarrow$) For a maximal complete subgraph (V^K, E^K) such that $M = \mathcal{L}(V^K)$, there exists a unique configuration T_c such that $T_c\bullet = V^K$. A existence of a firing sequence in PN corresponding to T_c and that M is reachable from M_0 after the firing of the sequence are satisfied by the definition of unfolding. ∎

For the PN in Figure 4.1, we can obtain the set of reachable markings as follows from Theorem 4.1: $\{p_1, p_1, p_3, p_3\}$, $\{p_1, p_3, p_4\}$, $\{p_1, p_2\}$ and $\{p_4, p_4\}$. See Figure 4.4.

By modifying Algorithm 4.1, we can obtain Algorithm 4.2 for Problem 4.1. See Figure 4.8. Algorithm 4.1 is exhaustive, thus the algorithm is not efficient. In Algorithm 4.2, by removing subgraphs that do not contain all vertices corresponding target marking, we can reduce computation time.

Example 4.1 Let us practice Algorithm 4.2 with the net in Figure 4.9(a). Assume that the destination marking is $M = \{p_3, p_6\}$. Unfolding of the net is drawn in Figure 4.9(b), and Figure 4.10(a) is the concurrent-relation graph induced by P. The graph CG$\backslash P$ is a non-separable net, thus the graph is appended to *list_of_graphs*. Since CG$\backslash P$ is not a complete graph, it is divided into graphs $G_{p_6^1}$ and $G_{\overline{p_6^1}}$. These graphs are drawn in Figure 4.11(a) and Figure 4.10(b). Here, $G_{\overline{p_6^1}}$ does not contain any vertex corresponding to p_6, therefore this graph is removed for *list_of_graphs*.

Algorithm 4.2 *Is Reachable?*

input: A concurrent-relation graph $CG\backslash P$, and M.

output: TRUE or FALSE.

begin

Divide CG$\backslash P$ into non-separable subgraphs.;

Append non-separable subgraphs to *list_of_graphs*.;

/* Append only subgraphs containing all vertices corresponding to M. */

while *list_of_graphs* $\neq \emptyset$ **do**

Take the first graph G'= (V', E') from *list_of_graphs*.;

if G' is a complete graph, **then**

if $\mathcal{L}(V') == M$ **then**

return TRUE ;

endif

else

Divide G' into subgraphs G'_x and $G'_{\overline{x}}$.;

/* Do not select a vertex as x, which is adjacent with all other vertices.*/

Divide G'_x and $G'_{\overline{x}}$ into non-separable subgraphs.;

Append non-separable subgraphs to *list_of_graphs*.;

/* Append only subgraphs containing all vertices corresponding to M.*/

endif

endwhile

return FALSE ;

end

Figure 4.8: An algorithm for the reachability problem.

$G_{p_6^1}$ is divided into non-separable graphs, non-separable graphs are drawn in Figures 4.11(b) and (c). The graph in Figure 4.11(b) is a complete graph and satisfies the condition

$$\mathcal{L}(V) == M, \tag{4.31}$$

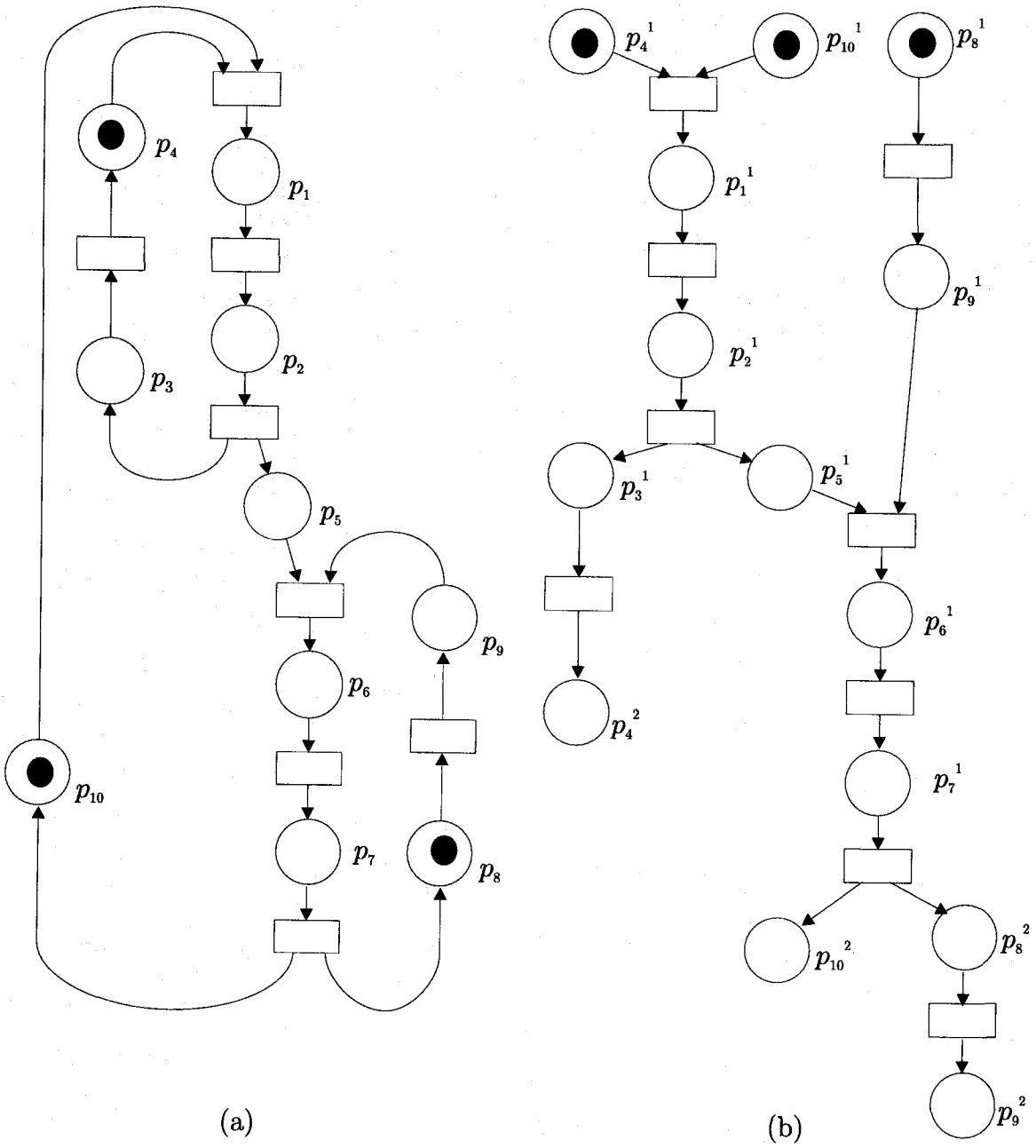

Figure 4.9: Petri net "pla" (a) and its unfolding (b).

where V is the set of vertices of the graph. Consequently, we can say that the destination marking $M = \{p_3, p_6\}$ is reachable.

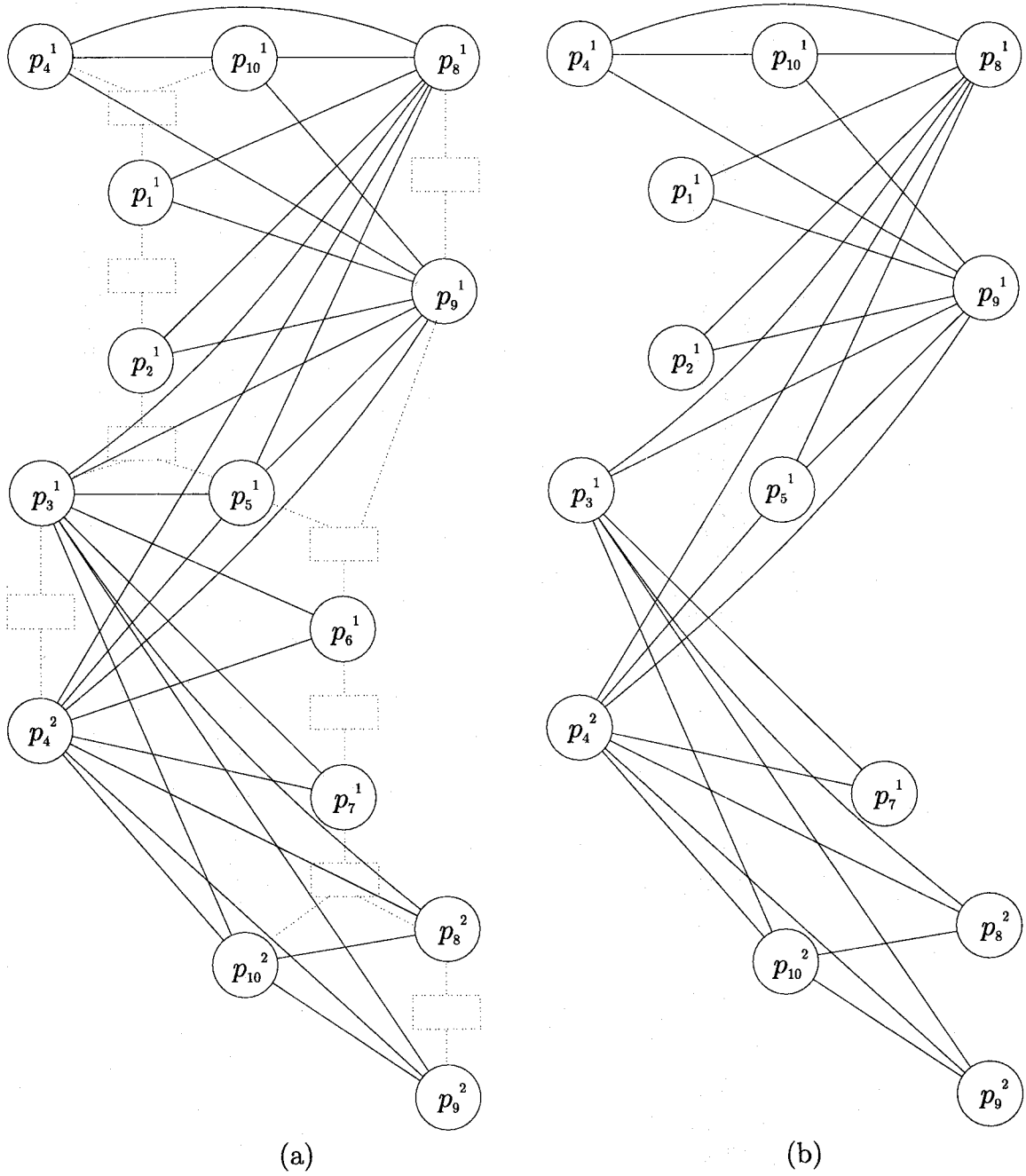

Figure 4.10: Reachability Verification(1). (a)CG\P. (b) $G_{\overline{p_6^1}}$.

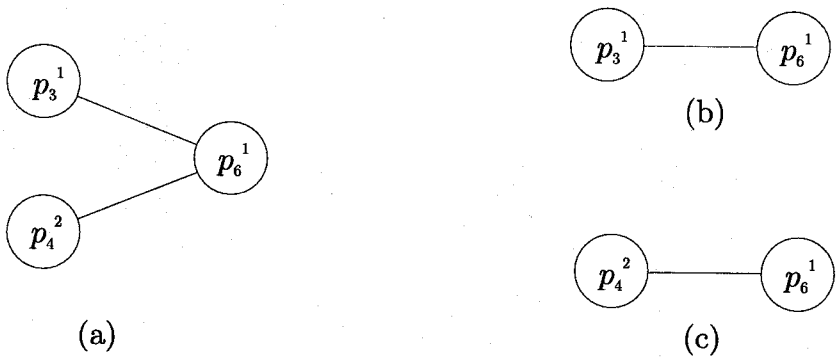

Figure 4.11: Reachability Verification(2). (a) $G_{p_6^1}$. (b),(c) non-separable graphs of (a).

4.3.3 Experimental Results and Comparison

This subsection compares the proposed method with other methods.

For verification of the reachability problem, we can enumerate the following methods:

- the reachability graph method,
- the symbolic model method, and
- the unfolding method.

Table 4.1 shows the experimental results. "PN" is the name of Petri net. In the row "RG", data of the reachability graph method are shown: "$|R(M_0)|$" is the number of reachable markings, and "time" is the CPU time to verify reachability by using the reachability graph method. In the row "BDD", data of the symbolic model method are shown: "# node" is the number of BDD nodes, and "time" is the CPU time to verify reachability by using the symbolic model method. In the row "unfolding", data of the unfolding method are shown: "# place" is the number of places in the unfolding, "# trans." is the number of transitions in the unfolding, and "time" is the CPU time to verify reachability by using the unfolding method. All the times have been measured on a SUN SPARCstation 4 with 32MB main memory.

Table 4.1: Experimental Results

PN	RG		BDD		unfolding		
	$\|R(M_0)\|$	time	# node	time	#place	#trans.	time
pla3.3	9856	145.2	231	5.1	90	62	0.7
pla3.4	14464	317.8	278	7.6	92	64	0.8
pla3.5	78720	–	314	19.2	136	97	2.1
vme2	586	0.8	128	2.7	64	55	0.3
vme3	10232	180.5	225	10.6	103	88	1.3
master-read	2108	7.1	3581	19.0	77	50	0.6
din10	6.0×10^7	–	796	21.9	80	50	1.1
din20	3.7×10^{15}	–	1419	80.0	160	100	9.8
dme20	2.2×10^7	–	239	5.1	81	61	0.7
dme40	4.5×10^{13}	–	476	19.5	161	120	5.4

Although for the reachability method there is no restriction about applicable class, the method can be applied only to small nets. Experiments on nets whose "time" value is "–" failed because of the lack of memory. Experimental results have proven that this method can not be applicable to large nets.

As for the symbolic model method, the calculation time includes the upper bound verification time, because we have to know the upper bound in order to represent a marking by a Boolean function. The details of the upper bound verification will be discussed in the next section. All example nets are structurally bounded, therefore checking the upper bound was done by the S-invariant method. As it is shown in the experimental results of the next section, checking the upper bound by the S-invariant method requires little time. Therefore we can consider the experimental results in Table 4.1 as the time to construct the BDD tree. Experimental results prove that the symbolic model method is efficient as for the consumption of time. As for the consumption of space, the symbolic model method seems to be a good method on the whole. For example, as for the net "dme40", though the net has 4.5×10^{13} states, the set of reachable markings is represented by a BDD tree with only 476 nodes. In

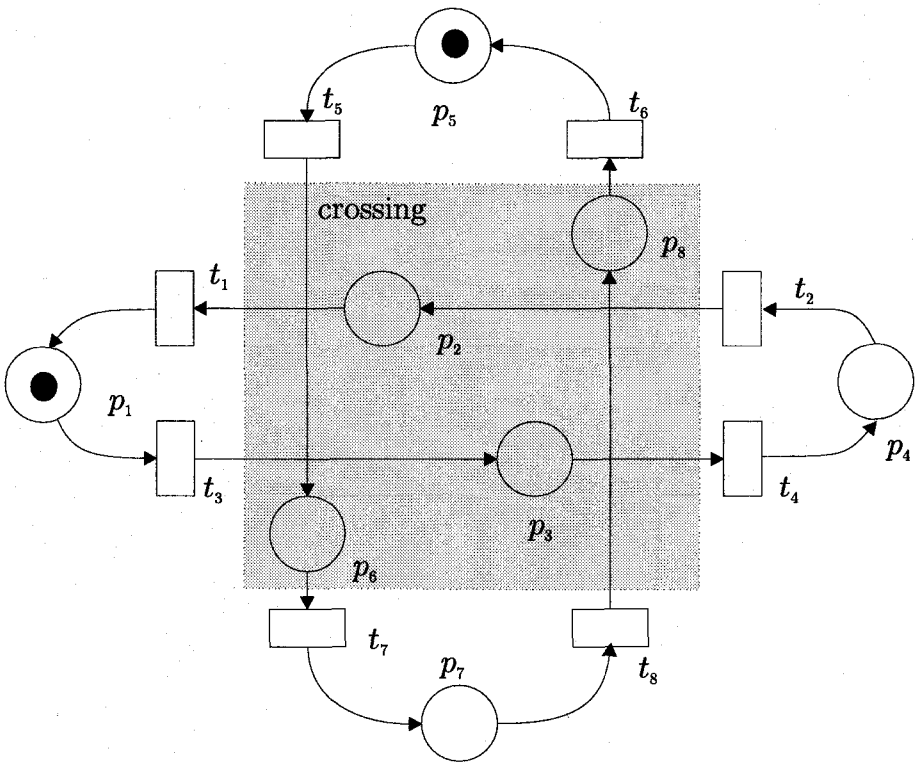

Figure 4.12: AGV system.

some cases, the method fails, for example, the net "master-read". In general, the size of the BDD tree depends on the order of variables. The problem of computing an ordering that minimizes the size of a given BDD is a co NP-complete problem. It seems that there exist some heuristics to obtain an adequate ordering[50, 12]. In this experiment, however, we ordered one by one. If we use heuristics, the size could be come smaller.

Similar to the symbolic model method, the unfolding method obtains better results than the reachability graph method. Experimental results prove that the unfolding method is efficient in terms of both the consumption of time and space.

Other advantages of the unfolding method are a by-product of a configuration and the structure of unfoldings. A configuration shows the set of transitions that must be fired, and the unfolding represents relations between transitions in the configuration. This will be useful when we intend to control a discrete event system

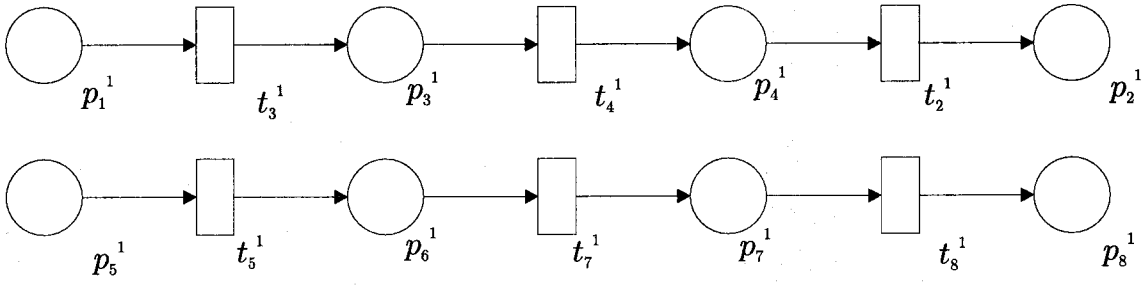

Figure 4.13: The unfolding of the AGV system.

Table 4.2: Reachability Verification

	Reachability Graph	BDD	unfolding
Efficiency Characteristic	non efficient	efficient	efficient structure of unfoldings

modeled by a Petri net[53]. See Figure 4.12. This figure shows a simple AGV system. Let us consider a control specification that the number of tokens in a zone "crossing" is at most one. Even if we adopt the symbolic model method, we can find that the system is reachable to forbidden markings $\{p_2, p_6\}$, $\{p_2, p_8\}$, $\{p_3, p_6\}$ and $\{p_3, p_8\}$. With the symbolic model method, however, it is difficult to find firings of what transitions cause the state transition to forbidden markings, because a BDD tree represents the set of reachable states. Cost to find firing sequences on the BDD tree is equal to the cost on the reachability tree. This is expensive. Figure 4.13 is the unfolding of the AGV system. Let us consider a case in which the forbidden marking is given by $\{p_2, p_6\}$. It is easily found that the system is reachable to the forbidden marking, and from the configuration for the marking we can find that firing of transitions t_2, t_3, t_4 and t_5 causes the state transition to the marking. Moreover ordering relations of these transitions are expressed by the structure of the unfolding.

As a summary, we can obtain Table 4.2.

4.4 Upper Bound Verification with Unfoldings

This section discusses the upper bound problem and upper bound verification.

4.4.1 Upper Bound Problem

An upper bound $\sharp(p, \overline{M})$ of a place $p \in P$ is the maximum number of tokens which are placed at the place p in a marking reachable from M_0, i.e.,

$$\sharp(p, \overline{M}) = \max_{M \in R(N, M_0)} \sharp(p, M). \tag{4.32}$$

The upper bound problem for Petri nets is formulated as follows.

Problem 4.2 (Upper Bound Problem) For a given PN, what is the upper bound of tokens in each place?

Although the upper bound problem is resolved for structurally bounded PN's by using S-invariants[40], for bounded but not structurally bounded PN's we must construct a reachability graph. It requires an exponential time. This section considers the relationship between these problems and an unfolding, and proposes a method to find an answer to these problems.

4.4.2 Unfoldings and Upper Bound Problem

For the upper bound problem, we can obtain the following theorem.

Theorem 4.2 For a PN $= (P, T, F, W; M_0)$, the upper bound for a place $p \in P$, denoted by $\sharp(p, \overline{M})$, is the number of vertices of the maximum complete subgraph in the concurrent-relation graph CG$\backslash p$, i.e.,

$$\sharp(p, \overline{M}) = |V^{K_{\max}}|, \qquad \mathrm{CG}^{K_{\max}} \subseteq \mathrm{CG}\backslash p. \tag{4.33}$$

Proof: By Theorem 4.1, it is guaranteed that any marking M reachable from M_0 is buried as a maximal complete subgraph in CG$\backslash P$. Thus any sub-marking with respect to a place p is buried as a maximal complete subgraph in CG$\backslash p$. Namely, the upper bound is given by the number of vertices of the maximum complete subgraph.

■

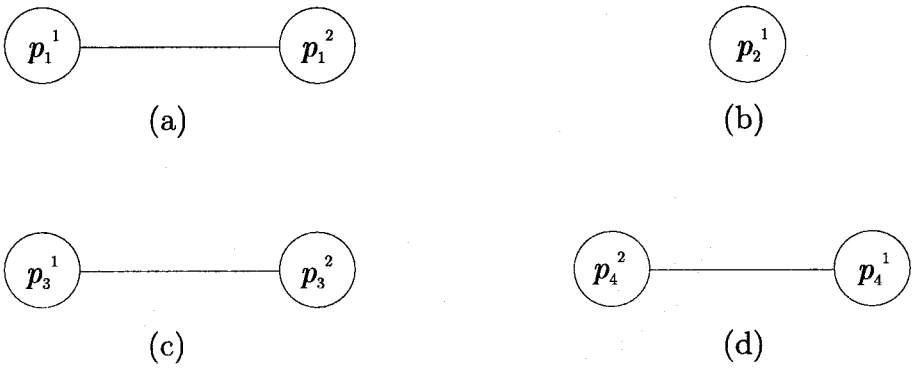

Figure 4.14: CG\$p$ of the unfolding in Figure4.2. (a)CG\$p_1$, (b)CG\$p_2$, (c)CG\$p_3$ and (d)CG\$p_4$.

Example 4.2 In Figure 4.14, concurrent-relation graphs induced by each place are shown, and each graph is the maximum complete graph. Therefore we can obtain the upper bound for each place as follows: $\sharp(p_1, \overline{M}) = 2$, $\sharp(p_2, \overline{M}) = 1$, $\sharp(p_3, \overline{M}) = 2$ and $\sharp(p_4, \overline{M}) = 2$.

By modifying Algorithm 4.1, we can obtain Algorithm 4.3 for Problem 4.2. See Figure 4.15. Algorithm 4.1 is exhaustive, thus the algorithm is not efficient. In Algorithm 4.3, by listing subgraphs in the order of the number of vertices, we can stop calculation when the graph being studied is a complete graph.

4.4.3 Experimental Results and Comparison

This subsection compares the proposed method with other methods.

For the verification of the upper bound problem, we can list the following methods:

- the reachability graph method,
- the S-invariant method, and
- the unfolding method.

Algorithm 4.3 *Upper Bound for a Place p*

input: A concurrent-relation graph CG\p.
output: Upper Bound for a Place p.
begin
Divide CG\p into non-separable subgraphs.;
Append non-separable subgraphs to *list_of_graphs*.;
/* Subgraphs are listed in the order of the number of vertices. */
while *list_of_graphs* $\neq \emptyset$ **do**
Take the first graph G'= (V', E') from *list_of_graphs*.;
if G' is a complete graph, **then**
/* The objective value is the number of vertices of graph G'.*/
return $|V'|$
else
Divide G' into subgraphs G'_x and $G'_{\overline{x}}$.;
/* Do not select a vertex as x, which is adjacent with all other vertices.*/
Divide G'_x and $G'_{\overline{x}}$ into non-separable subgraphs.;
Append non-separable subgraphs to *list_of_graphs*.;
/* Subgraphs are listed in order of the number of vertices. */
endif
endwhile
end

Figure 4.15: An algorithm for the upper bound problem.

Table 4.3 shows experimental results. "PN" is the name of a Petri net. In the row "RG", data of the reachability graph method are shown: "$|R(M_0)|$" is the number of reachable markings, and "time" is the CPU time to verify upper bound by using the reachability graph method. In the row "S-invariants", data of the S-invariant method are shown: "# inv." is the number of S-invariants, and "time" is the CPU time to verify upper bound by using the S-invariant method. In the row "unfolding", data of the unfolding method are shown: "# place" is the number of places in the

Table 4.3: Experimental Results

PN	RG		S-invariants		unfolding		
	$\|R(M_0)\|$	time	# inv.	time	#place	#trans.	time
n reader-writer							
$n = 10$	12	0.0	2	0.0	30	10	0.0
$n = 20$	22	0.0	2	0.0	60	20	0.2
$n = 30$	32	0.0	2	0.0	90	30	0.7
$n = 40$	42	0.0	2	0.0	120	40	1.8
$n = 50$	52	0.0	2	0.0	150	50	3.4
$n \times n$ ring							
$n = 10$	2	0.0	100	0.0	20	1	0.0
$n = 20$	2	0.0	400	0.2	40	1	0.0
$n = 30$	2	0.0	900	1.2	60	1	0.1
$n = 40$	2	0.0	1600	4.4	80	1	0.1
$n = 50$	2	0.0	2500	8.8	100	1	0.3
pla3.3	9856	145.6	13	0.0	90	62	0.1
pla3.4	14464	318.0	13	0.0	92	64	0.1
pla3.5	78720	–	16	0.0	136	97	0.2
vme2	586	0.8	8	0.0	64	55	0.1
vme3	10232	180.5	17	0.1	103	88	0.2
master-read	2108	7.1	18	0.0	77	50	0.1
din10	6.0×10^7	–	30	0.1	80	50	0.1
din20	3.7×10^{15}	–	60	0.7	160	100	0.6
dme20	2.2×10^7	–	21	0.1	81	61	0.2
dme40	4.5×10^{13}	–	41	0.9	161	120	1.0

unfolding, "# trans." is the number of transitions in the unfolding, and "time" is the CPU time to verify upper bound by using the unfolding method. All the times have been measured on a SUN SPARCstation 4 with 32MB main memory.

Although for the reachability method there is no restriction about applicable

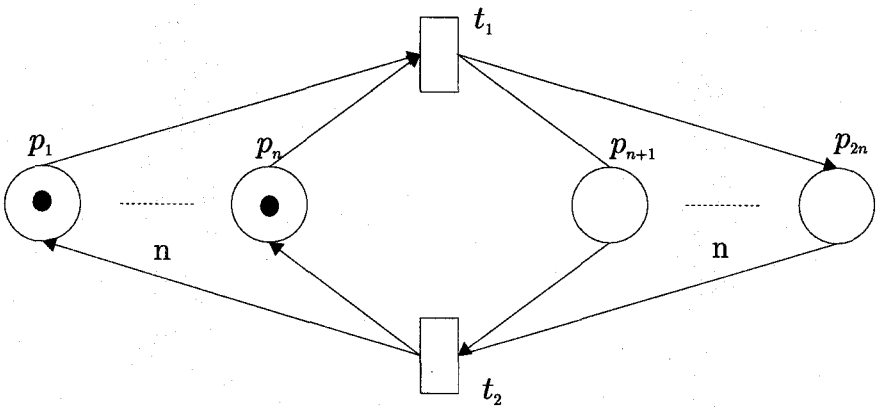

Figure 4.16: A $n \times n$ ring net.

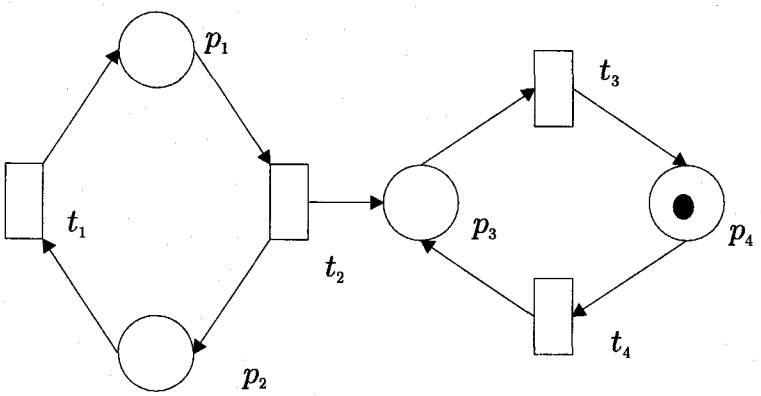

Figure 4.17: A non structurally bounded net.

class, the method can be applied only to small nets. Experiments on those nets whose "time" value is "–" failed because of the lack of memory.

As for the S-invariants method, computational complexity of the method is independent with the number of reachable markings, and it depends only on the number of S-invariants. This implies that the method can be applied to large nets if there are few numbers of S-invariants, and this is shown in the results of large nets like "pla3.3", "pla3.4", etc. Conversely, if there are too many S-invariants, the method requires much more time than other methods, even if there are few numbers of reachable markings. The net shown in Figure 4.16 consists of $2n$ places and 2 transitions.

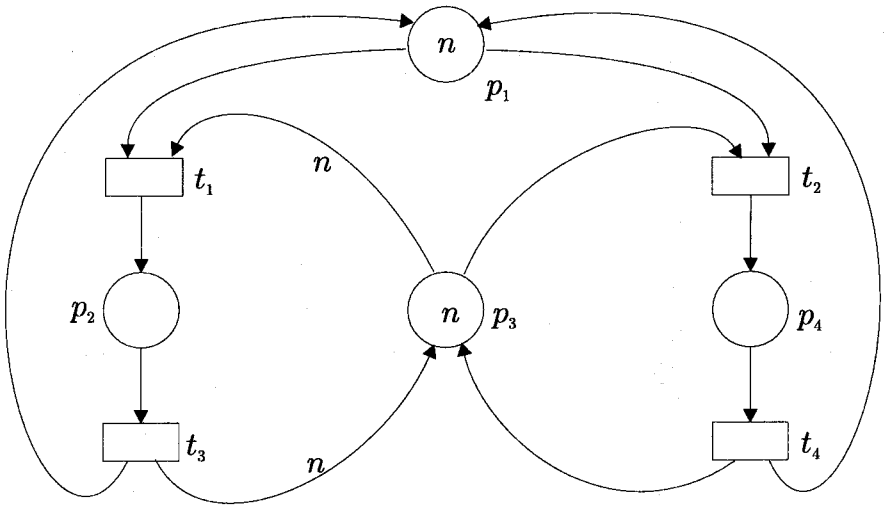

Figure 4.18: A readers-writers system with n process.

Arcs are drawn from transition t_1 to places $p_{n+1}, \ldots, p_{2n}$, from places $p_{n+1}, \ldots, p_{2n}$ to transition t_2, from transition t_2 to places $p_1, \ldots, p_n$ and from places $p_1, \ldots, p_n$ to transition t_1. Places $p_1, \ldots, p_n$ have a token at the initial marking. Although the number of reachable marking of the net is 2, the net has n^2 S-invariants. As a result, the method requires much more time than other methods. Moreover the method can be applied only for structurally bounded nets. A net in Figure 4.17 is not structurally bounded, since only one S-invariant of the net is as follows:

$$y = \begin{bmatrix} 0 & 0 & 1 & 1 \end{bmatrix}^T \tag{4.34}$$

Therefore the S-invariants method can not be applied to this net.

Similar to the S-invariants method, the unfolding method obtains better results than the reachability graph method. The method can be applied to large nets, and there is no restriction with respect to applicable class, namely this method can be applied to non structurally bounded nets, like the net in Figure 4.17. A shortcoming of the unfolding method is that as the number of tokens in a place becomes high, computational complexity increases. The readers-writers system with n processes is drawn in Figure 4.18. In the net, n tokens are placed in places p_1 and p_3 at the initial

Table 4.4: Upper Bound Verification

	Reachability Graph	S-invariant	unfolding
Class	no restriction	structurally bounded	no restriction
Efficiency	non efficient	efficient	efficient
Characteristic		marking independent	marking dependent

marking. Result of this example show that, more time is required as n increase. On the other hand, since the net structure of the system is changeless, the incidence matrix of the net is also changeless. Therefore S-invariants of the net is always two. Thus computational complexity for this net is invariant. The independence from the initial marking is a characteristic of the S-invariant method.

As a summary, we can obtain Table 4.4

4.5 Concluding Remarks

This chapter considered a computational problem of the reachability and the upper bound problems.

First, a concept of concurrent-relation graphs was introduced. The graph represents concurrent relation between instances in an unfolding. Two graph division methods were introduced, and it was shown that maximal complete subgraphs was preserved by the division methods.

Secondly, it was shown that the reachability problem results in finding a maximal complete subgraphs in a concurrent-relation graph induced by the set of places, and an algorithm to find the maximal complete subgraph was shown. Experimental results showed effectiveness of the proposed algorithm.

Finally, it was shown that the upper bound problem results in finding the maximum complete subgraph in a concurrent-relation graph induced by a place, and an algorithm to find the maximum complete subgraph was proposed. Experimental results showed effectiveness of the proposed algorithm.

Chapter 5

Synthesis of Asynchronous Circuits with Petri Net Unfoldings

This chapter discusses the problem of synthesizing asynchronous circuits from signal transition graphs (STG's) description with unfoldings.

5.1 Introduction

The progress of VLSI technology enables us to manufacture low cost and high performance software and hardware chips. We can divide the digital hardware systems broadly into two categories:

- Synchronous: use an external global **clock** for observing system states, and

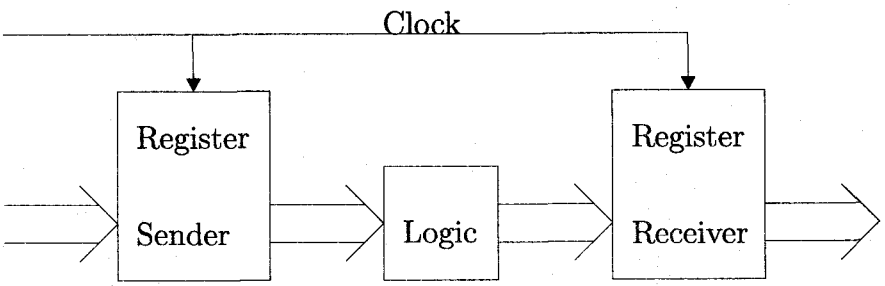

Figure 5.1: Synchronous Communication.

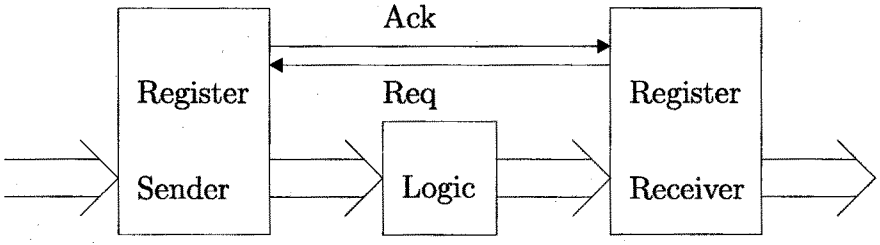

Figure 5.2: Asynchronous Communication.

- Asynchronous: use internal and external **events** for observing system states.

In Figures 5.1 and 5.2, synchronous communication and asynchronous communication are shown. In the synchronous system, a change of system state is triggered by an external global clock. On the other hand, in the asynchronous system, a change of system state is triggered by internal and external events. Although almost all the hardware is synchronous circuits, to materialize much lower cost and higher performance hardware, synchronous circuits have some problems, i.e., clock skew, power consumption, etc. Asynchronous circuits have many potential advantages against synchronous circuits, e.g.,

- high-speed operation: depending only on causal relation of signal transitions with *average* delay instead of *worst-case* delay,
- design cost reduction: due to separation between logical correctness and lower level circuit timing,
- timing fault tolerance: thanks to insensitivity to delay variance in layout, fabrication and operating environment, and
- low power consumption: due to signal transition made only when necessary.

Recently, there are urgent requirements for higher performance (high ability and low power consumption) hardware. Despite many advantages of asynchronous circuits over synchronous circuits, they have not been widely used beyond few applications, e.g., interface circuits. We can point out following disadvantages of asynchronous circuits.

- Area penalty in data path due to dual-rail encoding of data.
- Completion detection is a source of inefficiency, especially in memory devices.
- More sensitive to transient faults.
- Pin penalty due to 2-railed data.
- Interface with standard parts.
- More sophisticated and diversified design techniques.

To overcome these difficulties, an establishment of formal mathematical methodology and development of a synthesis tool are necessary.

For the systematic asynchronous circuits design methodology, assumption on wire and gate delay is important. We can enumerate four delay models.

- **Fundamental Mode** (Huffman model):
  - Variable delays with known bounds.
  - Input changes occur when circuit is stable.

  In this model, technology mapping is easy, but verification is difficult since delay must be considered.
- **Speed-Independent** (Muller model):
  - Finite, but unbounded, gate delays.
  - No wire delays.

  In this model, technology mapping is more difficult, but verification is easy since time is excluded.
- **Delay-Insensitive**:
  - Finite, but unbounded, gate delays and wire delays.
- **Quasi-Delay-Insensitive**:

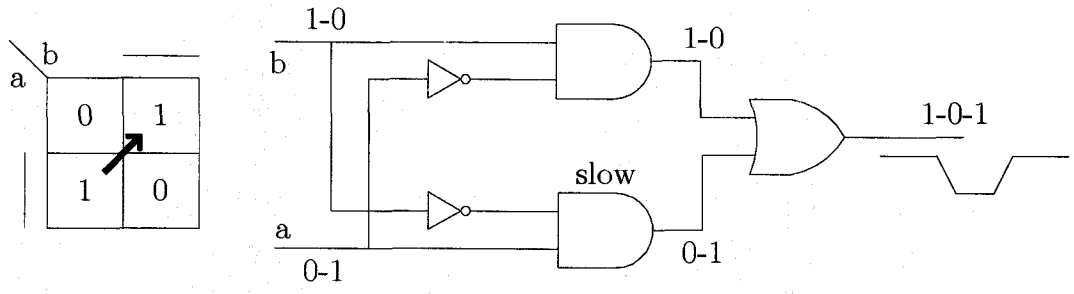

Figure 5.3: Hazards.

– Delay-Insensitive + Isochronic Forks

In practice this is the same as speed-independent.

A hazard is a signal transition not specified by the designer. In Figure 5.3, a typical example of hazards is shown. When input signals change from $\overline{a}b$ to $a\overline{b}$, and lower and-gate is slow, then the output signal of the or-gate will be $1 \rightarrow 0 \rightarrow 1$. Such an illegal signal change is called a hazard. For the completion of the asynchronous system design, the elimination of hazards is important, but hazards make the asynchronous system design difficult. In the speed-independent design, disabling cannot occur for unbounded gate delay model, therefore there exists no hazard, and a complex theory of hazards is not necessary for this delay model. This thesis assumes the speed-independent delay.

STG's were introduced in [4]. STG's are Petri nets, whose transitions are interpreted as a signal transition on the circuit inputs or gate outputs, and its marking represents a binary state of the circuit. Asynchronous behavior of circuits can be modeled by concurrent firings of transitions of STG's. Under an assumption that target circuits are speed independent, we can obtain a hazard free implementation from an STG. To derive a logic function of circuits, however, the knowledge of binary states, namely a set of all reachable markings is required. In a case an underling net is restricted to a live and safe free-choice net (LSFC net), methods to derive a logic function or to verify Complete State Coding (CSC) property in polynomial time have been presented without generating a whole state space[45, 46]. This method

uses the decomposition method discussed in section 2.3. As stated before, it is impossible to extend the result to a general case, since the method strongly depends on the structural characteristics of LSFC nets. It is shown in [54, 59] that LSFC net is too restrictive to represent asynchronous circuits. In a general case, a logic function can be derived by constructing a reachability graph, called a State Graph(SG). Unfortunately, as discussed in Chapter 4 the method based on SG's is not efficient since it requires exponential time complexity to construct an SG, and the method is not applicable to large STG's. The problem is thus to derive an efficient algorithm that reduces a computation time as much as possible.

In [29], the unfolding method to avoid the state space explosion problem in the verification of Petri nets is proposed. In [19], a property of STG's, called *correctness*, is defined, and a verification method for correctness by using reduced unfoldings is proposed. This chapter discusses a method on deriving logic functions of asynchronous circuits by using STG unfoldings, and compares with other methods. The method proposed in this chapter assumes the correctness of STG's.

5.2 Synthesis of Asynchronous Circuits from STG's

This section reviews the methods of synthesizing asynchronous circuits from STG's.

STG's are Petri nets, whose transitions are interpreted as a signal transition on the circuit inputs or gate outputs. A signal transition can be represented by $+s$ or $-s$ for a transition of signal s from 0 to 1, or from 1 to 0, respectively.

Definition 5.1 (STG) An STG is a triple $G = (\mathbf{N}, A, \lambda)$, where $\mathbf{N}$ is a PN or an OCN, A is the set of signals and $\lambda : T \rightarrow \{+, -\} \times A$ is the labeling function. Henceforth we assume $|A| = d$.

In Figure 5.4, an example of STG's and its unfolding are shown. For any firing sequence of $\mathbf{N}$, there corresponds a unique sequence of signal transitions in an STG.

In an STG, a marking of an STG or corresponding OCN represents a binary state of the circuit.

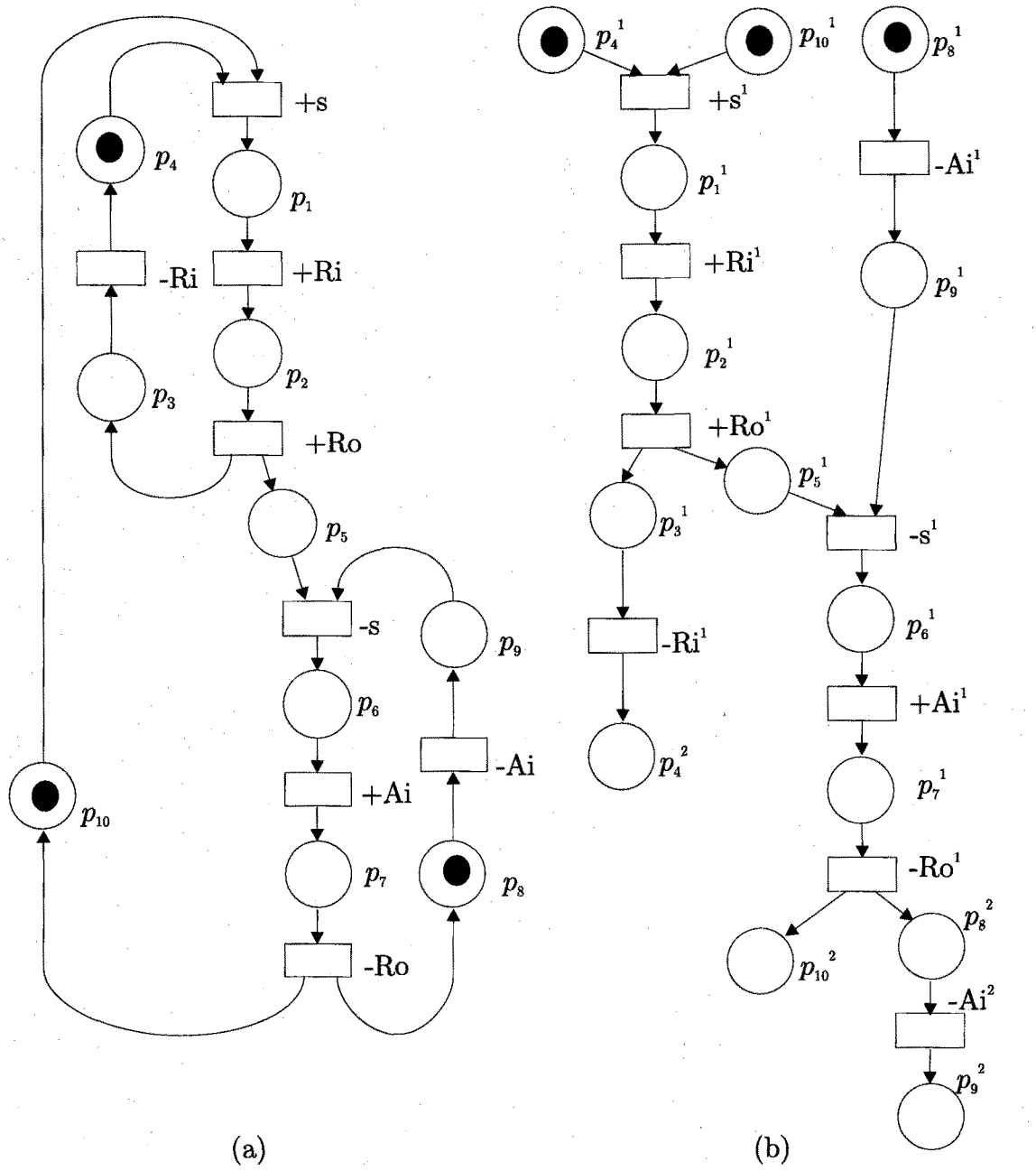

Figure 5.4: An STG(a) and its unfolding(b).

Definition 5.2 (State) In an STG, a *state* on a marking M, denoted by $S(M, V_0)$ or $S(M)$, is a binary vector $(s_1, \cdots, s_d)$ of the circuit, where V_0 is an initial state

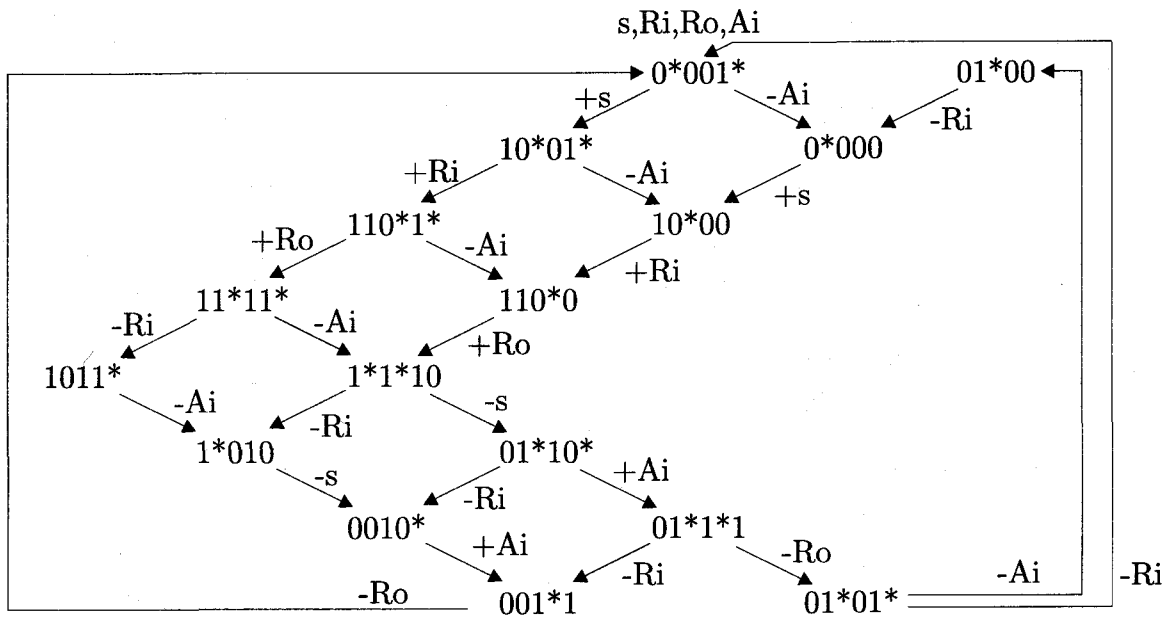

Figure 5.5: The SG of STG in Figure5.4(a).

corresponding to M_0. If $M_i[t>M_j$, then

$$S(M_j, V_0)_k = \begin{cases} 0 & \text{if} \quad \lambda(t) = -s_k \\ 1 & \text{else if} \quad \lambda(t) = +s_k \\ S(M_i, V_0)_k & \text{else} \end{cases} \tag{5.1}$$

For the set $\mathcal{M}$ of markings of an STG, $S(\mathcal{M}, V_0) = \{S(M, V_0) | M \in \mathcal{M}\}$.

Note that if an initial state V_0 is not given, we can derive it from an initial marking M_0 in polynomial time[19].

Let us consider deriving a logic function from an STG. A State Graph(SG) is the reachability graph of the STG in which a state is assigned to each marking reachable from an initial marking. In Figure 5.5, the SG of the STG in Figure 5.4(a) is shown. In the SG, the value of signals s, Ri, Ro and Ai are listed in this order, and changeable signals are labeled by $*$. For example, $0 * 001*$ indicates, current values of signal s, Ri and Ro are 0, that of signal Ai is 1, and signals s and Ai are changeable in this state.

Definition 5.3 Define a set of markings for signal s_i as follows:

$$\mathcal{M}_{s_i}^{+} = \{M \in R(M_0) | \, M[t>\}, \tag{5.2}$$

$$\mathcal{M}_{s_i}^{1} = \{M \in R(M_0) | \, S(M)_i = 1 \wedge (\neg M[t'>)\}, \tag{5.3}$$

$$\mathcal{M}_{s_i}^{-} = \{M \in R(M_0) | \, M[t'>\}, \text{ and} \tag{5.4}$$

$$\mathcal{M}_{s_i}^{0} = \{M \in R(M_0) | \, S(M)_i = 0 \wedge (\neg M[t>)\}, \tag{5.5}$$

where t (resp. t') is any transition such that $\lambda(t) = +s_i$ (resp. $-s_i$).

Intuitively, $\mathcal{M}_{s_i}^{+}$ is the set of markings in which the signal s_i is changeable from 0 to 1, and $\mathcal{M}_{s_i}^{1}$ is the set of markings in which the signal s_i is stable to 1. Similarly for $\mathcal{M}_{s_i}^{-}$ and $\mathcal{M}_{s_i}^{0}$. The set of states corresponding to a set of markings $\mathcal{M}$ is denoted by $\widehat{\mathcal{M}}$. For signal s, we can obtain the following sets of states.

$$\widehat{\mathcal{M}}_{\mathrm{s}}^{+} = \{0001, 0000\}, \tag{5.6}$$

$$\widehat{\mathcal{M}}_{\mathrm{s}}^{1} = \{1001, 1000, 1101, 1100, 1111, 1011\}, \tag{5.7}$$

$$\widehat{\mathcal{M}}_{\mathrm{s}}^{-} = \{1110, 1010\}, \text{ and} \tag{5.8}$$

$$\widehat{\mathcal{M}}_{\mathrm{s}}^{0} = \{0010, 0110, 0111, 0011, 0101, 0100\}. \tag{5.9}$$

Definition 5.4 (On-set, Off-set) Two sets of states *On-set* and *Off-set* for signal s, denoted by On-set(s) and Off-set(s), are defined as follows.

$$\text{On-set}(s) = \widehat{\mathcal{M}}_{s}^{+} \cup \widehat{\mathcal{M}}_{s}^{1}, \text{and} \tag{5.10}$$

$$\text{Off-set}(s) = \widehat{\mathcal{M}}_{s}^{-} \cup \widehat{\mathcal{M}}_{s}^{0}. \tag{5.11}$$

Plus and minus logic functions f_s+ and f_s- of signal s are represented by the disjunction of states in On-set(s) and Off-set(s), respectively[4]. Thus we can obtain the following logic functions for signal s from (5.6), (5.7), (5.8) and (5.9).

$$f_{\mathrm{s}}+ = \overline{\mathrm{Ri}}\,\overline{\mathrm{Ro}} + \mathrm{s}\overline{\mathrm{Ro}} + \mathrm{sAi} \tag{5.12}$$

$$f_{\mathrm{s}}- = \bar{\mathrm{s}}\mathrm{Ro} + \mathrm{Ro}\overline{\mathrm{Ai}} + \bar{\mathrm{s}}\mathrm{Ri} \tag{5.13}$$

The STG in Figure 5.4 have 2 input signals Ri and Ai, 1 internal signal s, and 1 output signal Ro. Thus after deriving logic functions of signal Ro, we can obtain an asynchronous circuit. A RS flip-flop implementation is shown in Figure 5.6.

Therefore we need the set of states to derive logic functions.

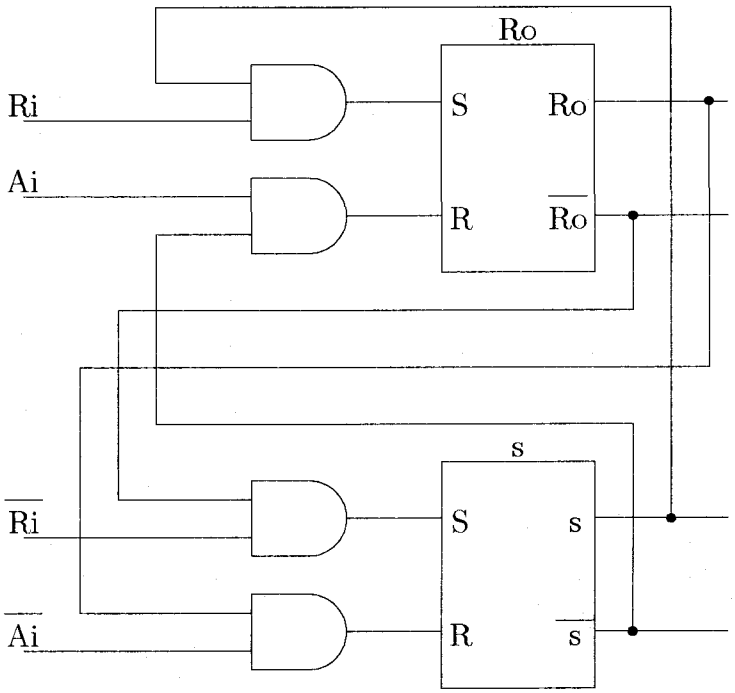

Figure 5.6: RS flip-flop implementation.

5.3 Synthesis of Asynchronous Circuits with Unfoldings

This section discusses a method to derive logic functions from STG's by using unfoldings.

Different from the reachability verification, we need the set of states to derive logic functions, namely we have to search the whole state space. Constructing the SG is not so expensive when the given STG is small. In some small nets, we can construct the SG in shorter time than constructing the unfolding. The method proposed in this section thus aims to divide the state space into small pieces whom we can obtain easily. The concurrent relation between nodes in unfoldings is used in order to obtain special sub-nets of the unfolding.

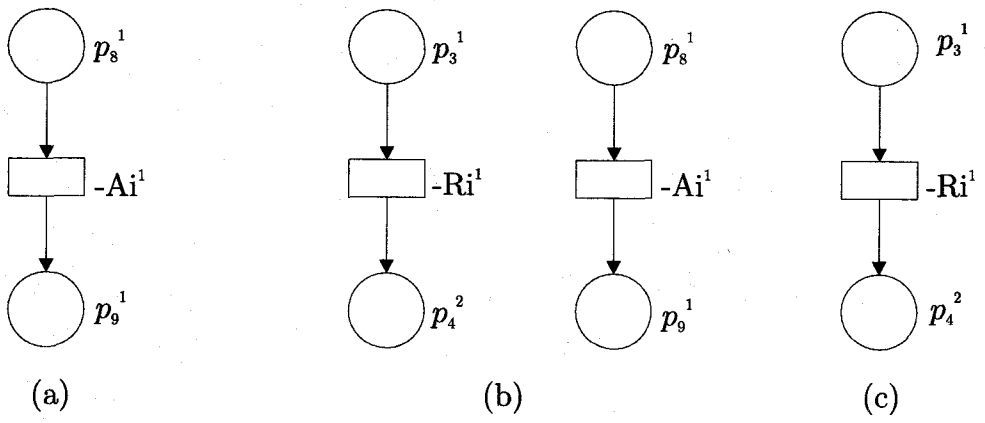

Figure 5.7: Examples of P(T)-Net's.(a) is T-Net($+s^1$), P-Net(p_1^1), P-Net(p_2^1). (b) is P-Net(p_5^1). (c) is T-Net($-s^1$).

5.3.1 P(T)-Net's

This subsection introduces new concepts P-Net's and T-Net's, and studies their characteristics.

Definition 5.5 (P-Net) A P-Net of a place p is a subnet P-Net$(p) = (P_p, T_p, F_p)$ of an unfolding, where

$$T_p = \{t \in T' | t//p\}, \tag{5.14}$$

$$P_p = \{p' \in P' | \exists t \in T_p : p' \in {}^\bullet t \vee p' \in t^\bullet\}, \text{and} \tag{5.15}$$

$$F_p = \{(x_1, x_2) \in F' | x_1, x_2 \in P_p \cup T_p\}. \tag{5.16}$$

Definition 5.6 (T-Net) A T-Net of a transition t is a subnet T-Net$(t) = (P_t, T_t, F_t)$ of an unfolding, where

$$T_t = \{t' \in T' | t'//t\}, \tag{5.17}$$

$$P_t = \{p' \in P' | \exists t' \in T_t : p' \in {}^\bullet t' \vee p' \in t'^\bullet\}, \text{and} \tag{5.18}$$

$$F_t = \{(x_1, x_2) \in F' | x_1, x_2 \in P_t \cup T_t\}. \tag{5.19}$$

In Figure 5.7, P-Net's and T-Net's for the unfolding in Figure 5.4(b) are shown. Transitions in a P-Net(p) or T-Net(t) can fire in the unfolding when a token is placed in place p or when transition t is enabled, respectively.

Lemma 5.1 In an unfolding, for each place $p' \in P_p$ in a P-Net(p) $= (P_p, T_p, F_p)$ with $\bullet p \neq \emptyset$, we can write

$$\begin{cases} p' \in (\Rightarrow \bullet p)\bullet & \text{if} \quad \bullet p' \cap T_p = \emptyset, \text{and} \\ p' \notin (\Rightarrow \bullet p)\bullet & \text{if} \quad \bullet p' \cap T_p \neq \emptyset. \end{cases} \tag{5.20}$$

Proof: (1) Case $\bullet p' \cap T_p = \emptyset$. Since $p' \in P_p$,

$$\exists t \in p' \bullet \text{ such that } t//p. \tag{5.21}$$

From equation (5.21),

$$\neg(p' \Rightarrow p). \tag{5.22}$$

(a) If $p' \in M'_0$, then, from equation (5.22) and Definition 3.2, $p' \in (\Rightarrow \bullet p)\bullet$.
(b) If $\bullet p' \Leftarrow p$, then $p \Rightarrow \bullet p' \Rightarrow p' \Rightarrow t$. This contradicts equation (5.21).
(c) If $\bullet p' \sharp p$, then $t \sharp p$. This contradicts equation (5.21).
(d) Assume $\bullet p' // p$. This contradicts that $\bullet p' \cap T_p = \emptyset$.
(e) If $\bullet p' \Rightarrow p$, then, from equation (5.22) and Definition 3.2, $p' \in (\Rightarrow \bullet p)\bullet$.
Consequently, $p' \in (\Rightarrow \bullet p)\bullet$.

(2) Case $\bullet p' \cap T_p \neq \emptyset$. That implies $\bullet p' // p$, therefore $\bullet p' \notin (\Rightarrow \bullet p)$. From Definition 3.2, $p' \notin (\Rightarrow \bullet p)\bullet$. ∎

Lemma 5.2 In an unfolding, let $M'[t > (\Rightarrow t)\bullet$. For each place $p' \in P_t$ in a T-Net(t) $= (P_t, T_t, F_t)$, we can write,

$$\begin{cases} p' \in M' & \text{if} \quad \bullet p' \cap T_t = \emptyset \\ p' \notin M' & \text{if} \quad \bullet p' \cap T_t \neq \emptyset. \end{cases} \tag{5.23}$$

Proof: The proof of this lemma is similar to that of Lemma 5.1. Therefore it is omitted here. ∎

When a marking of P-Net(p) is equal to $*P_p$, we denote this marking by M_0^p, and define the state V_0^p as below.

$$V_0^p = \begin{cases} S((\Rightarrow \bullet p)\bullet) & \text{if } \bullet p \neq \emptyset \\ S(M_0') & \text{if } \bullet p = \emptyset \end{cases} \tag{5.24}$$

When there is a token on all source places in T-Net(t), we denote this marking by M_0^t, and define the initial state V_0^t as

$$V_0^t = S(M', V_0'), \tag{5.25}$$

where $M'[t> (\Rightarrow t)\bullet$.

Theorem 5.1 The set of states of an unfolding, in which there is a token on place p, is equal to the set of states in the SG of P-Net(p) with initial marking M_0^p and initial state V_0^p.

Proof: (1) Case where $\bullet p \neq \emptyset$. In the unfolding, let us describe $(\Rightarrow \bullet p)\bullet$ as follows,

$$(\Rightarrow \bullet p)\bullet = \{p\} \cup P_0 \cup P_1, \tag{5.26}$$

where $P_0 = M_0^p$ and $P_1 = (\Rightarrow \bullet p) \bullet - \{p\} - P_0$. From Lemma 5.1, $P_1 \cap P_p = \emptyset$, therefore

$$\forall p_j \in P_1 : p_j \notin P_p. \tag{5.27}$$

From Definition 5.5 and equation (5.27),

$$\forall p_j \in P_1, \forall t \in p_j\bullet : p \sharp t, p \Rightarrow t \text{ or } t \Rightarrow p. \tag{5.28}$$

Hence t is not enabled when a token is placed on p. Since $(\Rightarrow \bullet p)\bullet$ is the first marking, in which place p have a token, the sets of states are equal.

(2) We can prove similarly in the case where $\bullet p = \emptyset$. ∎

Theorem 5.2 The set of states of an unfolding, in which a transition t is enabled, is equal to the set of states in the SG of T-Net(t) with initial marking M_0^t and initial state V_0^t.

Proof: The proof of this theorem is similar to that of Theorem 5.1. Therefore it is omitted here. ∎

Theorem 5.1 and 5.2 show that we can obtain the set of states in which a token is placed on a place p (resp. states in which a transition t is enabled), from the SG of P-Net(p) (resp. T-Net(t)). Hereafter an initial marking of a P-Net (resp. T-Net) is given by M_0^p (resp. M_0^t), and an initial state of a P-Net (resp. T-Net) is given by V_0^p (resp. V_0^t).

Example 5.1 In the unfolding in Figure 5.4(b), the set $\mathcal{M}(p_1^1)$ of reachable markings, in which a token is placed on place p_1^1, is given by the following equation,

$$\mathcal{M}(p_1^1) = \{\{p_1^1, p_8^1\}, \{p_1^1, p_9^1\}\}. \tag{5.29}$$

And its corresponding set of states is given by

$$\widehat{\mathcal{M}}(p_1^1) = \{1001, 1000\}. \tag{5.30}$$

P-Net(p_1^1) is shown in Figure 5.7(a), and its initial state is given by the following equation,

$$\begin{aligned} V_0^{p_1^1} &= S((\Rightarrow \bullet p_1^1)\bullet, V_0') && (5.31)\\ &= S(\{p_1^1, p_8^1\}, 0001) && (5.32)\\ &= 1001. && (5.33) \end{aligned}$$

Therefore from the SG of P-Net(p_1^1), we can obtain the set of states,

$$\widehat{\mathcal{M}}(p_1^1) = \{1001, 1000\}. \tag{5.34}$$

5.3.2 Tree

This subsection introduces new concept *tree*, and studies its characteristics.

The set of states of a P-Net of each series place is mutually disjoint. We define a set of places, called a tree, to search all state space.

Definition 5.7 (Tree) In an unfolding, a set $\Gamma \subseteq P'$ of places is called a *tree* when

$$\exists p \in \Gamma : p \in M'_0, \text{ and} \tag{5.35}$$

$$\forall p \in \Gamma : \forall t \in p\bullet, t\bullet \cap \Gamma \neq \emptyset. \tag{5.36}$$

Although a choice of a tree Γ is not unique, the reachable states can be obtained from any Γ as shown in the following theorem.

Theorem 5.3 For a given STG, the set of all states of the STG is equal to the union of the set of all states of P-Net(p), where a place p belongs to a tree Γ of its unfolding, i.e.,

$$S(R(N, M_0), V_0) = \bigcup_{p \in \Gamma} S(R(\text{P-Net}(p), M_0^p), V_0^p) \tag{5.37}$$

Proof: It has been proved in Lemma 3.1 and Lemma 3.2 that the set of all markings of an STG can be derived by the set of all markings of its unfolding. From the definition of a tree, for any marking M reachable from M_0 of the STG, there exists a place p' with $p' \in \Gamma \wedge \mathcal{L}(p') \in M$ in the unfolding. Then from Theorem 5.1, we can derive any state of the STG from its unfolding. ∎

5.3.3 Next State Function Derivation

The set $\widehat{\mathcal{M}}_s^+$(resp. $\widehat{\mathcal{M}}_s^-$) is the set of states in which the transition t with $\lambda(t) = +s$ (resp. $-s$) is enabled. We can obtain the set $\widehat{\mathcal{M}}_s^+$ and $\widehat{\mathcal{M}}_s^-$ from the SG of T-Net(t). Consequently, the following theorem can be obtained.

Theorem 5.4 For a given STG, logic functions of a signal s can be derived by the Algorithm 5.1.

Proof: The proof of this theorem is trivial. Thus it is omitted here. ∎

Theorem 5.4 provide us with a means to derive logic functions by dividing into smaller set of states. The main problem on deriving logic function is the state space explosion problem. This division method will help us to derive logic functions.

Algorithm 5.1 *Deriving a Logic Function for Signal s*

input: an STG.
output: On-set(s) and Off-set(s).
begin
Construct an unfolding from the STG.;
Find a tree Γ in the unfolding.;
foreach place $p \in \Gamma$
  Construct the SG of P-Net(p).;
  foreach state in the SG
    if the value of signal s of the state is 1
      Add the state to On-set(s);
    else
      Add the state to Off-set(s);
    endif
  endforeach
endforeach
foreach transition t with $\lambda(t) = +s$ (resp. $-s$)
  Add all states in the SG of T-Net(t) to On-set(s) (resp. Off-set(s)).;
  Delete all states in the SG of T-Net(t) from Off-set(s) (resp. On-set(s)).;
endforeach
end

Figure 5.8: An algorithm to derive a logic function.

Example 5.2 Consider a case of deriving logic functions of signal s for the STG in Figure5.4(a). A tree of the unfolding in Figure5.4(b) is $\Gamma = \{p_8^1, p_9^1, p_6^1, p_7^1, p_8^2, p_9^2\}$. We can obtain the following sets of states for each places. In each state, the values of signal s, Ri, Ro and Ai are listed in this order.

$$\widehat{\mathcal{M}}(p_8^1) = \{0001, 1001, 1101, 1111, 1011\} \tag{5.38}$$
$$\widehat{\mathcal{M}}(p_9^1) = \{0000, 1000, 1100, 1110, 1010\} \tag{5.39}$$
$$\widehat{\mathcal{M}}(p_6^1) = \{0110, 0010\} \tag{5.40}$$
$$\widehat{\mathcal{M}}(p_7^1) = \{0111, 0011\} \tag{5.41}$$
$$\widehat{\mathcal{M}}(p_8^2) = \{0101, 0001\} \tag{5.42}$$
$$\widehat{\mathcal{M}}(p_9^2) = \{0100, 0000\} \tag{5.43}$$

Here, $\widehat{\mathcal{M}}(p) = S(R(\text{P-Net}(p), M_0^p), V_0^p)$. After calculating SG of all the P-Net's, On-set(s) and Off-set(s) are obtained as follows.

$$\text{On-set(s)} = \{1001, 1101, 1111, 1011, 1000, 1100, 1110, 1010\} \tag{5.44}$$
$$\text{Off-set(s)} = \{0001, 0000, 0110, 0010, 0111, 0011, 0101, 0100\} \tag{5.45}$$

Instantiations of +s and −s in the unfolding in Figure 5.4(b) are $+\text{s}^1$ and $-\text{s}^1$. The sets of states of each T-Net's are as follows.

$$\widehat{\mathcal{M}}(+\text{s}^1) = \{0001, 0000\} \tag{5.46}$$
$$\widehat{\mathcal{M}}(-\text{s}^1) = \{1110, 1010\} \tag{5.47}$$

Here, $\widehat{\mathcal{M}}(t) = S(R(\text{T-Net}(t), M_0^t), V_0^t)$. Therefore at the end of Algorithm 5.1, On-set(s) and Off-set(s) are obtained as follows.

$$\text{On-set(s)} = \{1001, 1101, 1111, 1011, 1000, 1100, 0001, 0000\} \tag{5.48}$$
$$\text{Off-set(s)} = \{0110, 0010, 0111, 0011, 0101, 0100, 1110, 1010\} \tag{5.49}$$

In Figure5.9, the Karunaugh map for signal s is shown. A state in On-set(s) (resp. Off-set(s)) is expressed by 1 (resp. 0) in the map. Consequently, we can derive logic functions of signal s from the map as follows.

$$f_\text{s}+ = \overline{\text{Ri}}\,\overline{\text{Ro}} + \text{s}\overline{\text{Ro}} + \text{sAi} \tag{5.50}$$
$$f_\text{s}- = \bar{\text{s}}\text{Ro} + \text{Ro}\overline{\text{Ai}} + \bar{\text{s}}\text{Ri} \tag{5.51}$$

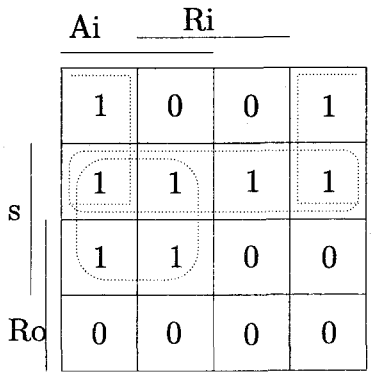

Figure 5.9: The Karunaugh map for signal s.

5.3.4 Several Techniques to Improvement

This subsection discusses a method to extend the Algorithm 5.1.

Congruent Relation

Let us introduce a concept of *congruent* relation to eliminate redundant state space construction.

Definition 5.8 (Congruent) In an OCN, nodes x_1 and x_2 are called in *congruent* relation, denoted by

$$x_1 \cong x_2, \tag{5.52}$$

if two sets of transitions, which are in concurrent relation with nodes x_1 and x_2, respectively, are the same.

From Definitions 5.5, 5.6 and 5.8, it is obvious that if nodes are in congruent relation, P(T)-Net's of each node have the same structure. Therefore reachability graphs of these P(T)-Net's are the same, since the set of source places is given for an initial marking of a P(T)-Net.

For a given P(T)-Net, we can derive the SG by constructing the reachability graph and assigning a state to each marking from a given initial state. Thus when nodes x_1 and x_2 are in congruent relation, we can use the reachability graph of the

P(T)-Net of node x_1 for that of node x_2. Constructing the reachability graph is the most expensive in deriving logic functions. Therefore great time reduction can be expected by eliminations of construction of a reachability graph.

There are many cases in which nodes are in congruent relation. Here we show three cases in which nodes are in congruent relation. Note that in each case checking congruent relation is done in a few steps.

Lemma 5.3 In an unfolding $= (P, T, F; M_0; \mathcal{L})$, a place $p \in P$ and a transition $t \in T$ are in congruent relation, when

$$\bullet t = \{p\}. \tag{5.53}$$

Proof: Let us describe T as follows,

$$\begin{aligned} T &= T_p^\sharp \cup T_p^{//} \cup T_p^{\Rightarrow} \cup T_p^{\Leftarrow} \\ &= T_t^\sharp \cup T_t^{//} \cup T_t^{\Rightarrow} \cup T_t^{\Leftarrow} \cup \{t\}, \end{aligned} \tag{5.54}$$

where $T_p^\sharp$ (resp. $T_t^\sharp$) is the set of transitions in conflict with p (resp. t), $T_p^{//}$ (resp. $T_t^{//}$) is the set of transitions concurrent with p (resp. t), $T_p^{\Rightarrow}$ (resp. $T_t^{\Rightarrow}$) is the set of transitions which are predecessors of p (resp. t) and $T_p^{\Leftarrow}$ (resp. $T_t^{\Leftarrow}$) is the set of transitions which are successors of p (resp. t). From $\bullet t = \{p\}$, we can obtain

$$T_p^{\Rightarrow} = T_t^{\Rightarrow}, \text{ and} \tag{5.55}$$

$$T_p^\sharp \subseteq T_t^\sharp. \tag{5.56}$$

Consider a case where $|p\bullet| = 1$, namely $p\bullet = \{t\}$. From $p\bullet = \{t\}$,

$$T_p^{\Leftarrow} = T_t^{\Leftarrow} \cup \{t\}. \tag{5.57}$$

For each transition $t' \in T_t^\sharp$, there exist distinct transitions $t_1, t_2 \in T$ such that $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ and $(t_1, t), (t_2, t') \in F^*$. From $p\bullet = \{t\}$, $t_1 \neq t$, therefore t_1 is a predecessor of p. That implies $T_t^\sharp \subseteq T_p^\sharp$, so that, from equation (5.56),

$$T_t^\sharp = T_p^\sharp. \tag{5.58}$$

From equations (5.54), (5.55),(5.57) and (5.58),

$$T_p^\sharp \cup T_p^\Rightarrow \cup T_p^\Leftarrow = T_t^\sharp \cup T_t^\Rightarrow \cup T_t^\Leftarrow \cup \{t\}, \tag{5.59}$$

namely, $T_p^{//} = T_t^{//}$.

In a case where $|p\bullet| > 1$, the set $T_p^\Leftarrow$ can be expressed by

$$T_p^\Leftarrow = T_t^\Leftarrow \cup \{t\} \cup T_o, \tag{5.60}$$

where each pair of sets $T_t^\Leftarrow$, $\{t\}$ and T_o is mutually disjoint. For each transition $t'' \in T_o$, there exists a transition $t''' \in T_o$ such that $(t''', t'') \in F^*$, $t''' \in p\bullet$ and $t''' \neq t$, thus $t\sharp t''$, namely,

$$T_o \subseteq T_t^\sharp. \tag{5.61}$$

Therefore, from equations (5.55), (5.56), (5.60) and (5.61),

$$\begin{aligned} T_p^\sharp \cup T_p^\Rightarrow \cup T_p^\Leftarrow &\subseteq T_t^\sharp \cup T_t^\Rightarrow \cup T_t^\Leftarrow \cup \{t\} \cup T_o \\ &= T_t^\sharp \cup T_t^\Rightarrow \cup T_t^\Leftarrow \cup \{t\}, \end{aligned} \tag{5.62}$$

namely,

$$T_t^{//} \subseteq T_p^{//}. \tag{5.63}$$

For a transition $t_3 \in T_p^{//}$, from $\bullet t = \{p\}$, a predecessor of t is p or is a predecessor of p, thus

$$\neg(t_3 \Rightarrow t).p \tag{5.64}$$

Similarly,

$$\neg(t \Rightarrow t_3). \tag{5.65}$$

Assume that $t\sharp t_3$, then there exist distinct transitions $t_4, t_5 \in T$ such that $\bullet t_4 \cap \bullet t_5 \neq \emptyset$ and $(t_4, t), (t_5, t_3) \in F^*$. t_4 is t or is a predecessor of p since a predecessor of t is p or is a predecessor of p. When t_4 is t, t_5 is a successor of p, thus t_3 is a successor

of p. This contradicts that $t_3 \in T_p^{//}$. When t_4 is a predecessor of p, $t_3 \in T_p^{\sharp}$. This contradicts that $t_3 \in T_p^{//}$. Therefore

$$\neg(t \sharp t_3). \tag{5.66}$$

Equations (5.64), (5.65) and (5.66) imply

$$T_p^{//} \subseteq T_t^{//}, \tag{5.67}$$

so that $T_p^{//} = T_t^{//}$.

Consequently p and t are in congruent relation. ∎

Lemma 5.4 In an unfolding $= (P, T, F; M_0; \mathcal{L})$, a place $p \in P$ and a transition $t \in T$ are in congruent relation, when

$$t\bullet = \{p\}. \tag{5.68}$$

Proof: The set of transitions ($T_p^{\sharp}$, etc.) are defined similarly. Since $t\bullet = \{p\}$ and $\bullet p = \{t\}$, it is obvious that $T_p^{\sharp} = T_t^{\sharp}$, $T_p^{\Leftarrow} = T_t^{\Leftarrow}$, $T_p^{\Rightarrow} = T_t^{\Rightarrow} \cup \{t\}$ and $T_p^{//} = T_t^{//}$. Therefore p and t are in congruent relation. ∎

Lemma 5.5 In an unfolding $= (P, T, F; M_0; \mathcal{L})$, places $p_1, p_2 \in P$ are in congruent relation, if

$$\bullet t = \{p_1\} \text{ and } t\bullet = \{p_2\}, \tag{5.69}$$

where $\bullet p_2 = \{t\}$.

Proof: From Lemma 5.3,

$$T_{p_1}^{//} = T_t^{//}. \tag{5.70}$$

From Lemma 5.4,

$$T_{p_2}^{//} = T_t^{//}. \tag{5.71}$$

Therefore

$$T^{//}_{p_1} = T^{//}_{p_2}, \tag{5.72}$$

so that p_1 and p_2 are in congruent relation. ∎

Example 5.3 In the case of the unfolding in Figure 5.4(b), we can find following congruent relations.

$$\begin{array}{lllllllll}
+\mathrm{s}^1 & \cong & p_1^1 & \cong & +\mathrm{Ri}^1 & \cong & p_2^1 & \cong & \mathrm{Ro}^1 \\
p_3^1 & \cong & -\mathrm{Ri}^1 & \cong & p_4^2 & & & & \\
p_8^1 & \cong & -\mathrm{Ai}^1 & \cong & p_9^1 & & & & \\
-\mathrm{s}^1 & \cong & p_6^1 & \cong & +\mathrm{Ai}^1 & \cong & p_7^1 & \cong & -\mathrm{Ro}^1 \\
p_8^2 & \cong & -\mathrm{Ai}^2 & \cong & p_9^2 & & & &
\end{array} \tag{5.73}$$

When we have to obtain the sets of states for P-Net(p_1^1), P-Net(p_2^1), and T-Net($+\mathrm{s}^1$), we have only to construct the reachability graph of one of these nets.

Optimal Tree Selection

In Algorithm 5.1, we construct SG's for each place in a tree and for each transition whose firing means a signal transition of a given signal s. We show a method to find an optimal tree by using congruent relation positively. The congruent relation is defined by the equivalence of sets and is an equivalent relation so that we can define the equivalent class of places, called *C-class*. From the condition of Lemma 5.5 and the fact that each place has at most one input transition, each C-class is a partial ordered set including the greatest element, where a is greater than b if $a \Rightarrow b$. When $\bullet t = \{p_1\}$ and $t\bullet = \{p_2\}$, if p_2 is in a tree, p_1 must be in the tree, from the condition 2 of Definition 5.7. Therefore for each C-class, only the SG of the greatest element is needed to construct. To optimize the selection of a tree, introduce a notion of *weight* for each place. A weight $w(p)$ of a place p is given according to the following criteria:

- For a given signal s, let T_s be the set of transitions with $\lambda(t) = *s^1$.

[1] $*s$ is $+s$ or $-s$.

Algorithm 5.2 *Finding an Optimal Tree*

input: an unfolding.

output: an optimal tree.

begin

For each place $p \in P'$, decide weight $w(p)$.;

For each place $p \in M_0$, call function Tree-place(p).;

A place $p \in M_0$ of which the value of the function Tree-place is minimum is the root place of an optimal tree.

end

Figure 5.10: An algorithm to find an optimal tree.

- For a C-class P, if there exist a transition $t \in T_s$ and a place $p \in P$ with $t \cong p$, for each place $p' \in P$, $w(p') = 0$.

- For a C-class P, if there does not exist a transition $t \in T_s$ and a place $p \in P$ with $t \cong p$, for the greatest place $p' \in P$, $w(p') = g(p')$, for other places $p'' \in P$, $w(p'') = 0$.

Ideally the function g should be a cost to construct an SG. Practically, $g(p)$ is expressed as a function of the number of nodes that are concurrent with p in this thesis. A weight of a tree Γ is given by the sum of weights of places in Γ. The problem is to find a tree whose weight is minimum. Since an unfolding is an acyclic directed graph, we have only to find optimal sub trees recursively in order to find an optimal tree. An algorithm to find an optimal tree is shown in Figure 5.10. In Figures 5.11 and 5.12, functions which are used in Algorithm 5.2 are shown.

Example 5.4 Consider a case of deriving logic functions of signal s for the STG in Figure 5.4(a). If the function g is given by $g(p) = 2^x$, where x is the number of

Function 5.1 *Tree-place* (p : *place*) : *weight* ;
begin
foreach transition $t \in p\bullet$
Tree-transition(t).;
endforeach
return the sum of all value of the Tree-transition and $w(p)$.;
end

Figure 5.11: Function Tree-place.

Function 5.2 *Tree-transition* (t : *transition*) : *weight* ;
begin
foreach place $p \in t\bullet$
Tree-place(p).;
endforeach
Decide a place p', such that the value of Tree-place(p') is minimum, as one of successors of $\bullet t$.;
return the value of Tree-place(p').;
end

Figure 5.12: Function Tree-transition

transitions concurrent with p, a weight of each place is given as follows:

$$w(p_4^1) = w(p_{10}^1) = w(p_8^2) = 2, \tag{5.74}$$
$$w(p_3^1) = 32, \tag{5.75}$$
$$w(p_8^1) = 16, \tag{5.76}$$
$$w(p_5^1) = w(p_{10}^2) = 4, \tag{5.77}$$

and for other places the weight is 0. From Algorithm 5.2, an optimal tree of Figure 5.4(b) is

$$\Gamma = \{p_4^1, p_1^1, p_2^1, p_5^1, p_6^1, p_7^1, p_8^2, p_9^2\}. \tag{5.78}$$

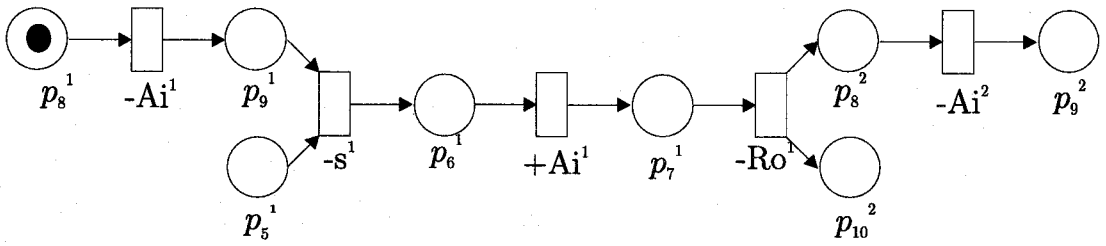

Figure 5.13: P-Net(p_3^1).

In this case, we can eliminate the construction of reachability graph for the P-Net of place p_1^1, p_2^1, p_6^1, p_7^1 and p_9^2. The number of states of P-Net(p_4^1) is 2, that of T-Net($+s^1$) is 2, that of P-Net(p_5^1) is 4, that of T-Net($-s^1$) is 4 and that of P-Net(p_8^2) is 2. Therefore total number of searched states is 14. When we do not use this optimization, we have to search 24 states. Therefore this optimization can be seen as a state reduction method.

Nesting

Though P(T)-Net's are smaller than the unfolding, they are not always much smaller than the STG. See Figure 5.13, where P-Net(p_3^1) is shown. In this case, the net has still 8 places and 5 transitions. The original STG has 10 places and 8 transitions.

Algorithm 5.1 provides a method to divide state space. Therefore, nesting the algorithm will be useful for this problem. Let us consider the case that we construct the SG of P(T)-Net's by using the algorithm. We can list the following properties.

- The P(T)-Net is an OCN, therefore we can omit constructing unfolding.
- We can also omit the step with respect to T-Net's, because we need only the whole state space of the given P(T)-Net.
- By the similar reason, we do not have to classify the states into On-set and Off-set, and have only to sum up the states.

In Figure 5.14, a nesting algorithm to obtain the set of states of a P(T)-Net is shown.

Algorithm 5.3 *Deriving a set of states of P(T)-Net's*
input: a P(T)-Net.
output: the set of states of the net.
begin
Find a tree Γ in the net.;
foreach place $p \in \Gamma$
Construct the SG of P-Net(p).;
Sum up the sets of SG's.;
return the set of states.;
end

Figure 5.14: Nesting algorithm.

Example 5.5 Let us apply the algorithm to P-Net(p_3^1). If we choose the set

$$\Gamma = \{p_8^1, p_9^1, p_6^1, p_7^1, p_8^2, p_9^2\} \tag{5.79}$$

as a tree, we can divide the net into 6 P-Net's. Although these nets have no nodes, they have an initial state. Namely, each nets has only one state. Now, we can derive six states for P-Net(p_3^1).

As stated above, we can avoid the state space explosion problem by nesting the algorithm. If a divided net is still large, we continue to apply the algorithm to the net.

5.4 Experimental Results

Table 5.1 shows the experimental results. "STG" is the name of STG's, "sig" is the number of signals, "tr" is the number of transitions and "states" is the number of states of its SG. The examples used for our experiments have been obtained from [46, 36, 22]. In the column "$\overline{\text{OSN}}$", "O$\overline{\text{SN}}$", "$\overline{\text{O}}$S$\overline{\text{N}}$", "$\overline{\text{OS}}$N" and "OSN", average time to derive a pair of (plus and minus) logic functions of one signals shown in

Table 5.1: Experimental results(1)

STG	sig	tr	states	$\overline{\text{OSN}}$	$\text{O}\overline{\text{SN}}$	$\overline{\text{O}}\text{S}\overline{\text{N}}$	$\overline{\text{OS}}\text{N}$	OSN
master-read	14	28	2108	9.8	9.3	9.6	6.4	2.5
pla3.3	14	28	9856	77.4	68.2	65.6	21.8	6.4
pla3.4	15	30	14464	121.0	110.0	100.3	29.7	10.1
pla3.5	16	32	78720	–a	–a	–a	114.9	34.8
vme3	19	53	10234	46.5	42.6	35.5	32.3	3.4
din10	30	60	6.0×10^7	–a	–a	–a	–b	909.8
dme20	40	80	2.2×10^7	–a	–a	–a	–b	78.5
dme40	80	160	4.5×10^{13}	–a	–a	–a	–b	768.4

second. "O" means that we select optimized tree, and "$\overline{\text{O}}$" means that we used an arbitrary selected tree. "S" means that redundant state space construction was skipped by using the congruent relation, and "$\overline{\text{S}}$" means that the state space was constructed for each P(T)-Net's. "N" means that the P(T)-net's were divided into smaller P-Net's when they were still large, and "$\overline{\text{N}}$" means that the state space was constructed even if a P(T)-Net was large. As for a weight function, we used a function $w(n) = 2^t$, where n is a place or a transition and t is the number of transitions that are in concurrent with n in this experiment. Since there is no systematic way to decide the weight function and we prefer smaller P(T)-Net's, we used the function. Since congruent relations given by Lemmas 5.3, 5.4 and 5.5 were not sufficient, we kept the information of transitions in P(T)-Net's for each place and used the information to check the congruent relation. Nesting was started when a sum of number of transitions and places exceeded a given threshold. In this experiment, we took 30 as a threshold. All the times have been measured on a SUN SPARCstation 4 with 32MB main memory. Experiments on nets whose "time" value is "–a" failed because of the lack of memory. "–b" means that the experiments were stopped compulsory because it did not finish in one day.

Experimental results show that

Table 5.2: Total number of searched state

	number of states
SG	10234
$\overline{\text{OSN}}$	15090
$\overline{\text{O}}\text{S}\overline{\text{O}}$	14945
$\text{O}\overline{\text{SN}}$	8340
$\overline{\text{OS}}\text{N}$	1708
OSN	83

- Algorithm 4.1 is not sufficient to avoid the state space explosion,
- Techniques "optimal tree selection", "congruent relation" and "nesting" are not sufficient alone, and
- by combining these techniques, we can avoid the state space explosion problem.

As for the possibility of the state space reduction, for example, the number of states of vme3 is 10234. For a specified signal, total number of searched states in each case is shown in Table 5.2. From this table, proposed method can be seem an efficient method with respect to state space reduction.

Table 5.3 shows the results obtained by the previous methods and the proposed method. "STG" is the name of STG's, "sig" is the number of signals, "tr" is the number of transitions and "states" is the number of states of its SG. In column "from [22]", results by using tool "Petrify" is shown. Shown values are the sum of "trav", "init" and "enc" in the output file of the tool. In the column "SG", time to construct the whole SG is shown in second. In the column "unfold", average time to derive a pair of (plus and minus) logic functions of one signal is shown in second. The weight function, nesting threshold and conditions for congruent relation were the same with the previous experiment. All the times have been measured on a SUN SPARCstation 4 with 32MB main memory. Experiments on these nets whose "time" value is denoted by "–a" failed because of the shortage of memory. "–b" means that the experiments were stopped compulsory because it was not finished in

Table 5.3: Experimental results(2)

STG	sig	tr	states	from [22]	SG	unfold
master-read	14	28	2108	99.8	8.6	2.5
pla3.3	14	28	9856	174.7	183.7	6.4
pla3.4	15	30	14464	244.2	396.5	10.1
pla3.5	16	32	78720	364.6	–a	34.8
vme3	19	53	10234	452.9	246.1	3.4
din10	30	60	6.0×10^7	4590.9	–a	909.8
dme20	40	80	2.2×10^7	1806.0	–a	78.5
dme40	80	160	4.5×10^{13}	–b	–a	768.4

one day. From the experimental result, the proposed method is more efficient than constructing the whole SG in order to derive logic functions.

5.5 Concluding Remarks

This chapter discussed the problem of synthesizing asynchronous circuits from STG's. In order to obtain next state logic functions from a given STG, the whole state space must be constructed. An algorithm to obtain next state logic functions was proposed. The concepts of P-Net's, T-Net's, the tree, the congruent relation, the optimal tree selection and nesting were introduced. The proposed algorithm utilizes these concepts, and provides with a method to avoid the state space explosion problem by dividing the state space into smaller pieces. Experimental results showed effectiveness of the proposed algorithm.

Chapter 6

Conclusions

This thesis studied the problem of verifying discrete event systems which were modeled by Petri nets by using their unfoldings.

Chapter 3 studied constructing unfoldings. Previous proposed unfoldings called M-unfolding and K-unfolding were compared first, some problems on these unfoldings were pointed out. The problem is how to reduce the size of M-unfolding without loss of generality.

Next, a new condition for a transition was proposed, and it was proved that the condition is a condition for a transition to be a cutoff point. The condition, however, depends on the order of coping transitions, thus an improved algorithm to construct unfoldings was proposed to avoid the ordering problem. The algorithm solves the ordering problem by using a stack. By using a condition that was proposed by McMillan and the new condition together, the reduction of unfoldings was successful in term of constructing time as well as required space. Experimental results show a significant effectiveness of the proposed method.

Chapter 4 considered a computational problem of the reachability and the upper bound problem.

First, a concept of concurrent-relation graphs was introduced. The graph represents concurrent relation between instances in an unfolding. Two graph division methods were introduced, and it was shown that maximal complete subgraphs was preserved by the division methods.

Secondly, it was shown that the reachability problem results in finding a maximal complete subgraphs in a concurrent-relation graph induced by the set of places, and an algorithm to find the maximal complete subgraph was shown. Experimental results showed effectiveness of the proposed algorithm.

Finally, it was shown that the upper bound problem results in finding the maximum complete subgraph in a concurrent-relation graph induced by a place, and an algorithm to find the maximum complete subgraph was proposed. Experimental results showed effectiveness of the proposed algorithm.

Chapter 5 discussed synthesizing asynchronous circuits. In order to obtain next state logic functions from a given STG, the whole state space must be constructed.

An algorithm to obtain next state logic functions was proposed. The concepts of P-Net's, T-Net's, the tree, the congruent relation, the optimal tree selection and nesting were introduced. The proposed algorithm utilizes these concepts, and provides with a method to avoid the state space explosion problem by dividing the state space into smaller pieces. Experimental results showed effectiveness of the proposed algorithm.

The verification methodology with Petri net unfoldings leaves possible directions of future research listed as follows:

- Unfoldings will be used to verify other properties of Petri nets, e.g., reversibility, coverability, etc.
- The methodology will be useful for the control theory of discrete event systems.
- CSC property verification should be done by using unfoldings.

This thesis has been studied unfoldings and Petri net's verification methodology by using unfoldings. The effectiveness of unfoldings was shown by experimental results of verifying the reachability problem and the upper bound problem and deriving next state logic functions.

References

[1] S. B. Akers, "Binary Decision Disgrams," *IEEE Transactions on Computers*, vol.C-27, no.6, pp.509–516, 1978.

[2] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol.C-35, no.8, pp.677–691, August 1986.

[3] E. Best, "Structure Theory of Petri Nets: the Free Choice Hiatus," *Lecture Notes in Computer Science*, vol.254, pp.168–206, June 1987.

[4] T. A. Chu, "Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications," in *Proceedings of ICCD'87*, pp.220–223, October 1987.

[5] J.M. Couvreur and E. Paviot-Adet, "New Structural Invariants for Petri Net Analysis," *Lecture Notes in Computer Science*, vol.815, pp.199–218, June 1994.

[6] J. Esparza and M. Silva, "Circuits, Handles, Bridges and Nets," *Lecture Notes in Computer Science*, June 1989.

[7] J. Esparza and M. Silva, "Top-down Synthesis of Live and Bounded Free Choice Nets," *Lecture Notes in Computer Science*, June 1990.

[8] J. Esparza, S. Römer and W. Vogler, "An Improvement of McMillan's Unfolding Algorithm," *Technical Report, Technische Universitat München*, August 1995.

[9] N. Funabiki and S. Nishikawa, "Comparisons of Energy-Descent Optimization Algorithms for Maximum Clique Problems," *IEICE Transactions Fundamentals*, vol.E79-A, no.4, pp.452–460, April 1996.

[10] H. J. Genrich and R. M. Shapiro, "Formal Verification of an Arbiter Cascade," *Lecture Notes in Computer Science*, vol.616, pp.205–223, June 1992.

[11] M. Hack, "Analysis of Production Schemata by Petri Nets," *M. S. thesis*, TR-94, Project MAC, MIT, 1972.

[12] K. Hiraishi and M. Nakano, "On Symbolic Model Checking in Petri Nets", *IEICE Transactions on Fundamentals*, vol.E78-A, no.11, pp.1479–1486, November 1995.

[13] K. Hiraishi, "Reduced State Space Representation for Unbounded Vector State Spaces," *Lecture Notes in Computer Science*, vol.1091, pp.230–248, June 1996.

[14] C. A. R. Hoare, "Communicating Sequential Processes," Prentice Hall, 1985.

[15] M. Iri, I. Shirakawa, Y. Kajitani and S. Shinoda, *Graph Theory with Exercise*, Corona Publishing, 1983. (in Japanese)

[16] H. Kagotani and T. Nanya, "On Performance Enhancement of Two-Phase Quasi-Delay-Insensitive Circuits," *IEICE Transactions Information & System*, vol.J78-D-I, no.4, pp.416–423, April 1995.

[17] M. Kishinevsky, A. Kondratyev, A. Taubin and V. Varshavsky, "Concurrent Hardware: The Theory and Practice of Self-Timed Design," John Wiley and Sons, London, 1993.

[18] M. A. Kishinevsky, A. Kondratyev and A. Taubin, "Specification and Analysis of Self-Timed Circuits," *Journal of VLSI Signal Processing*, no.7, pp.117–135, 1994.

[19] A. Kondratyev and A. Taubin, "On Verification of the Speed - Independent Circuits by STG unfoldings," *Technical Report 94-2-001, The University of Aizu*, 1994.

[20] A. Kondratyev, M. Kishinevsky and A. Yakovlev, "Monotonous cover transformations for speed-independent implementation of asynchronous circuits," *Technical Report , The University of Aizu*, July 1994.

[21] A. Kondratyev, A. Taubin and Sergey Ten, "Verification of Asynchronous Circuits by Petri Net Unfoldings," In *Proceedings of ETFA'94*, pp.404–413, 1994.

[22] A. Kondratyev, J. Cortadella, M. Kishinevsky, E. Pastor, O. Roig and A. Yakovlev, "Checking Signal Transition Graph Implementablity by Symbolic BDD Traversal," in *Proceedings EDTC-95*, pp325–332, March 1995.

[23] A. Kondratyev, M. Kishivevsky and A. Yakovlev, "On Hazard-Free Implementation of Speed-Independent Circuits," in *Proceedings of ASP-DAC'95*, pp.241–248, August 1995.

[24] A. Kondratyev, M. Kishinevsky, A. Taubin and Sergei Ten, "A Structural Approach for the Analysis of Petri Nets by Reduced Unfoldings," *Lecture Notes in Computer Science*, vol.1091, pp.346–365, June 1996.

[25] S.R. Kosaraju, "Decidability of reachability in vector addition systems," in *Proceedings of the 14th Annual ACM Symp. on Theory of Computing*, pp.267–281, May 1982.

[26] D.-I. Lee, "Analysis and Synthesis of Petri Nets by State Machine Decomposition," *Ph.D Thesis*, Osaka University, January 1993.

[27] J. Martinez and M. Silva, "A Simple and Fast Algorithm to Obtain all Invariants of a Generalized Petri Net," *Lecture Notes in Computer Science*, vol.52, pp.301–310, June 1982.

[28] E.W. Mayr, "An algorithm for the general Petri net reachability problem," *SIAM Journal on Computing*, vol.13, no.3, pp.441–460, August 1984.

[29] K. I. McMillan, "Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits," *Lecture Notes in Computer Science*, vol.663, pp.164–177, June 1993.

[30] T. H.-Y. Meng, R. W. Brodersen, and D. G. Messershumitt, "Automatic Synthesis of Asynchronous Circuits from High-Level Specifications," *IEEE Transactions on Computer-Aided Design*, vol.8, no.11, pp.1185–1205, November 1989.

[31] G. Memmi and G. Roucairol, "Linear algebra in net theory," *Lecture Notes in Computer Science*, vol.84 [15], pp.213-223, 1980.

[32] R. Milner, "Calculus for Communication Systems," *Lecture Notes in Computer Science*, vol.92, 1980.

[33] T. Miyamoto, D.-I. Lee and S. Kumagai, "An Efficient Method to Derive Logic from Signal Transition Graphs for Asynchronous Circuits," *Proceedings of JTC'96*, pp.766–769, July 1995.

[34] T. Miyamoto, D. I. Lee and S. Kumagai, "An Efficient State Space Search for the Synthesis of Asynchronous Circuits by Subspace Construction," *IEICE Transactions on Fundamentals*, vol.E78-A, pp.1504–1510, November 1995.

[35] T. Miyamoto and S. Kumagai, "An Efficient Algorithm for Deriving Logic Functions of Asynchronous Circuits," *Proceedings of Async'96*, pp.30–35, March 1996.

[36] T. Miyamoto and S. Kumagai, "An Efficient Algorithm for Deriving Logic Functions of Asynchronous Circuits," *IEICE Transactions on Fundamentals*, vol.E79-A, pp.818–824, June 1996.

[37] T. Miyamoto and S. Kumagai, "A Graph Theoretic Approach to Reachability Problem with Petri Net Unfoldings," *IEICE Transactions on Fundamentals*, vol.E79-A, pp.1809–1816, November 1996.

[38] T. Miyamoto and S. Kumagai, "On Deriving Logic Functions of Asynchronous Circuits by STG Unfoldings," *IEICE Transactions on Information & Systems*, vol.E80-D, to appear, March 1997.

[39] T. Murata, "State equation controllability, and maximal machings of Petri nets," *IEEE Transactions on Automatic Control*, vol.A-22, no.3, pp.412–416, June 1977.

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

[41] C. J. Myers and T. H.-Y. Meng, "Synthesis of Timed Asynchronous Circuits," *IEEE Transactions on VLSI Systems*, vol.1, no.2, pp.106–119, June 1993.

[42] M. Nakagawa, S. Kumagai, T. Miyamoto and D.-I. Lee, "Equivalent Net Reduction for Firing Sequence," *IEICE Transactions on Fundamentals,* vol.E78-A, pp.1447–1457, November 1995.

[43] T. Nanya, Y. Ueno, H. Kagotani, M. Kuwako and A. Takamura, "TITAC : Design of a quasi-delay-insensitive microprocessor," *IEEE Design & Test of Computers*, vol.11, no.2, pp.50-63, 1994.

[44] M. Notomi and T. Murata, "Hierarchically Organized Petri Net State Space for Reachability and Deadlock Analysis," in *Proceedings of IPPS'92*, pp. 616–622, March 1992.

[45] E. Pastor and J. Cortadella, "An Efficient Unique State Coding Algorithm for Signal Transition Graphs," in *Proceedings ICCD'93*, pp.174–177, October 1993.

[46] E. Pastor and J. Cortadella, "Polynomial Algorithms for the Synthesis of Hazard-free Circuits from Signal Transition Graphs," in *Proceedings ICCAD'93*, pp.250–254, November 1993.

[47] E. Pastor, O. Roig, J. Cortadella and R.M. Badia, "Petri Net Analysis Using Boolean Manipulation," *Lecture Notes in Computer Science*, vol.815, pp.416–435, June 1994.

[48] C. A. Petri, "Communication with Automata," in *Griffis Air Force Base*, New York, Technical Report RADC-TR-65-377, vol.1, 1966.

[49] O. Roig, J. Cortadella and Enric Pastor, "Verification of Asynchronous Circuits by BDD-based Model Checking of Petri Nets," *Lecture Notes in Computer Science*, vol.935, pp.374–391, June 1995.

[50] A. Semenov and A. Yakovlev, "Combining partial orders and symbolic traversal for efficient verification of asynchronous circuits", in *Proceedigns of ASP-DAC'95*, pp.567–573, August 1995.

[51] M. Silva, "Petri Nets in Automation and Computer Engineering," Kluwer Academin Press, 1989.

[52] I. Suzuki and T. Murata, "A method for stepwise refinements and abstractions of Petri nets," *Journal of Computer and System Science*, vol.27, no.1, pp.51–76, January 1983.

[53] S. Takai, "State Feedback Control of Discrete Event Systems," *Ph.D Thesis*, Osaka University, January 1995.

[54] A. V. Yakovlev, "On Limitations and Extensions of STG Model for Designing Asynchronous Control Circuits," in *Proceedigns ICCD'92*, pp.396–400, October 1992.

[55] A. Taubin, A. Kondratyev, M. Kishinevsky and S. Ten, "Deadlock Prevention Using Petri Net Unfoldings," in *Proceedigns of CESA'96, Symposimu on Discrete Events and Manufacturing Systems*, pp.426–431, July 1996.

[56] E. Teruel and M. Silva, "Liveness and Home States in Equal Conflict Systems," *Lecture Notes in Computer Science*, vol.691, pp.415–432, June 1993.

[57] E. Teruel and M. Silva, "Well-formedness of Equal Conflict Systems," *Lecture Notes in Computer Science*, vol.815, pp 491–510, June 1994.

[58] A. Valmari, "Compositionality in State Space Verification Methods," *Lecture Notes in Computer Science*, vol.1091, pp.29–56, June 1996.

[59] A. V. Yakovlev, M. A. Kishinevsky, A. Kondratyev and L. Lavagno, "OR Causality : Modelling and Hardware Implementation," *Lecture Notes in Computer Science*, vol.815, pp.568–587, June 1994.

[60] M. Yoeli, "Specification and Verification of Asynchronous Circuits Using Marked Graphs," 1987.

List of Publications by the Author

I. Transactions

1. T. Miyamoto, D.-I. Lee and S. Kumagai, "An Efficient State Space Search for the Synthesis of Asynchronous Circuits by Subspace Construction," *IEICE Transactions Fundamentals,* vol.E78-A, pp.1504–1510, November 1995.

2. M. Nakagawa, S. Kumagai, T. Miyamoto and D.-I. Lee, "Equivalent Net Reduction for Firing Sequence," *IEICE Transactions Fundamentals,* vol.E78-A, pp.1447–1457, November 1995.

3. T. Miyamoto and S. Kumagai, "An Efficient Algorithm for Deriving Logic Functions of Asynchronous Circuits," *IEICE Transactions Fundamentals,* vol.E79-A, pp.818–824, June 1996.

4. T. Miyamoto and S. Kumagai, "A Graph Theoretic Approach to Reachability Problem with Petri Net Unfoldings," *IEICE Transactions Fundamentals,* vol.E79-A, pp.1809–1816, November 1996.

5. T. Miyamoto and S. Kumagai, "On Deriving Logic Functions of Asynchronous Circuits by STG Unfoldings," *IEICE Transactions Information & Systems,* vol.E80-D, to appear, March 1997.

II. International Conferences

1. T. Miyamoto, D.-I. Lee and S. Kumagai, "An Efficient Method to Derive Logic from Signal Transition Graphs for Asynchronous Circuits," *Proceedings of JTC'96,* pp.766–769, July 1995.

2. T. Miyamoto and S. Kumagai, "An Efficient Algorithm for Deriving Logic Functions of Asynchronous Circuits," *Proceedings of Async'96,* pp.30–35, March 1996.

3. T. Miyamoto and S. Kumagai, "A Multi Agent Net Model of Autonomous Distributed Systems," *Proceedings of SESA'96, Symposium on Discrete Events and Manufacturing Systems* pp.619–623, July 1996.

4. S. Kumagai and T. Miyamoto, "An Agent Net Approach To Autonomous Distributed Systems," *Proceedings of SMC'96,* pp.3204–3209, October 1996.

III. Technical Reports and Convention Records

1. T. Miyamoto, S. Takai, D.-I. Lee and S. Kumagai, "Feedback control for safeness and fairness of discrete event systems," *Technical Report of IEICE,* CAS92–48, September 1992 (in Japanese).

2. T. Miyamoto and S. Kumagai, "An Efficient Algorithm for Deriving Logic Functions of Asynchronous Circuits," *Technical Report of IEICE,* CST95-30, Janunary 1996 (in Japanese).

3. T. Nobata, T. Miyamoto and S. Kumagai, "An agent net model of autonomous distributed systems," *Technical Report of IEICE,* CST95-30, January 1996 (in Japanese).