

國立交通大學

網路工程研究所

碩士論文

一個縮短發生圖建立時間的技術



A Technique for Reducing Occurrence Graph Building Time

研究生：林友涵

指導教授：王豐堅 教授

中華民國九十六年八月

一個縮短發生圖建立時間的技術
A Technique for Reducing Occurrence Graph Building Time

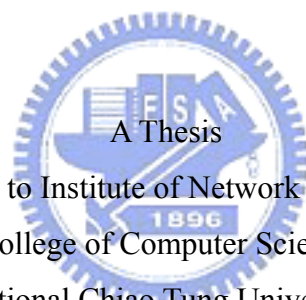
研究生：林友涵

Student : Yo-Han Lin

指導教授：王豐堅

Advisor : Feng-Jian Wang

國立交通大學
網路工程研究所
碩士論文



A Thesis
Submitted to Institute of Network Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Computer Science

August 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年八月


一個縮短發生圖建立時間的技術

研究生：林友涵

指導教授：王豐堅 博士

國立交通大學
網路工程研究所
碩士論文

摘要



隨著資訊時代的進步，各類系統愈來愈複雜，分析這些系統正確性的方法好壞變得很重要。執行一個方法的需求時間也是這個方法好壞的一個關鍵點。派翠網是一種可以被用來分析許多系統的模型，而發生圖是派翠網的狀態圖。一個派翠網的發生圖可以被用來分析許多這個派翠網的特性。這篇論文提出了一個技術：當在特定條件下融合兩個派翠網間的轉移點時利用它們的發生圖來縮短建立新發生圖的時間。如此一來當一些使用發生圖的分析方法需要取得發生圖時即可節省時間。因為這個技術一次只融合兩個派翠網，融合三個以上的派翠網的方法以及影響也會被討論。另外，文章中也提出這個技術的兩個較明顯的應用方向。

關鍵字： 派翠網、轉移點融合、發生圖、狀態圖、遞增分析、工作流程

A Technique for Reducing Occurrence Graph Building Time

Student: Yo-Han Lin

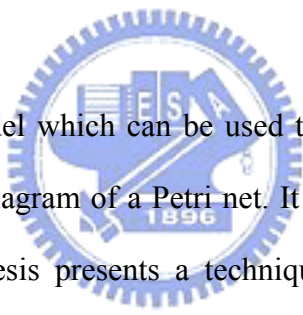
Advisor: Dr. Feng-Jian Wang

Institute of Network Engineering

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

Abstract

The logo of National Chiao Tung University is a circular emblem with a blue border. Inside the circle, there is a stylized representation of a building or a gear-like structure. The text 'NCTU' is written in the center, and '1896' is at the bottom. The logo is semi-transparent and overlaid on the abstract text.

Petri net is a kind of model which can be used to analyze many kinds of systems. An occurrence graph is the state diagram of a Petri net. It can be used to analyze many kinds of properties of the net. This thesis presents a technique that reduces the occurrence graph building time by applying the occurrence graphs of two Petri nets whose transitions to be merged. Since the technique merges two Petri nets a time only, the policy of merging more than two nets is discussed in this thesis. Two significant applications are also indicated.

Keywords: Petri net, transition mergence, occurrence graph, marking graph, state diagram, incremental analysis, workflow.

誌謝

本篇論文的完成，首先要感謝我的指導教授王豐堅博士兩年來不斷的指導與鼓勵，讓我在軟體工程及工作流程的技術上得到很多豐富的知識，使我可以在派翠網路上找到靈感。另外，也非常感謝我的畢業口試評審委員朱治平博士以及留忠賢博士，提供許多寶貴的意見，補足我論文裡不足的部分。

其次，我要感謝實驗室的學長姊們在研究生涯上的指導與照顧，尤其是靜慧學姊的督促與幫忙以及建德學長提供的寶貴經驗，讓我學得許多做研究的方法和技巧，得以順利的撰寫論文。

最後，我要感謝我的家人，由於有你們的支持，讓我能讀書、作研究到畢業，此外，謝謝同期碩士班好友們的打氣，在我遇到挫折時能互相勉勵並度過難關。由衷地感謝你們大家一路下來陪著我走過這段研究生歲月。



Table of Contents

摘要	i
Abstract.....	ii
誌謝	iii
Table of Contents.....	iv
List of Figures.....	v
Chapter 1. Introduction.....	1
Chapter 2. Background.....	3
2.1 Motivation	3
2.2 Petri Net.....	4
2.3 Occurrence Graph.....	7
2.4 Related Works.....	9
Chapter 3. Petri Net Transition Mergence	11
3.1 Combination of Two Petri Nets.....	11
3.2 The Algorithm of Combining Two Petri Nets	17
3.3 Merging Petri Net Transitions	21
3.4 Transition Mergence Algorithm.....	32
Chapter 4. More about Transition Mergence.....	39
4.1 Merging Places and Transitions.....	39
4.2 Merging Policy	46
4.3 Possible Applications.....	49
4.3.1 The Analysis Following the Component Mergence	50
4.3.2 Incremental Analysis for Edition in a Petri Net.....	52
Chapter 5. Example	60
Chapter 6. Conclusion and Future Works.....	67
Reference.....	69
Appendix	71

List of Figures

Figure 2.1 An example of a Petri net.....	4
Figure 2.2 The marking after firing t_1	7
Figure 2.3 The O-graph of the Petri net in Figure 2.1	8
Figure 3.1 (a) An example of a Petri net; (b) the O-graph of (a).....	11
Figure 3.2 (a) The Petri net generated by combining two Petri nets in Figure 3.1 (a); (b) the O-graph of (a)	13
Figure 3.3 (a) Merge two transitions between two nets from Figure 3.1 (a); (b) the O-graph of (a).....	22
Figure 4.1 (a) The result net of merging two pairs of transitions between two subnets from Figure 3.2 (a); (b) merging two places further; (c) the O-graph of (a) and (b)	41
Figure 4.2 An example of a merging tree	46
Figure 4.3 Another example of merging tree.....	49
Figure 4.4 Merging two transitions between two WorkFlow nets.....	51
Figure 4.5 The O-graph of the whole net shown in Figure 4.4	51
Figure 4.6 A method for incremental analyzing a Petri net.....	52
Figure 4.7 (a) “And Split” division; (b) “And Join” division.....	53
Figure 4.8 “Sequence” division.....	54
Figure 4.9 (a) A Petri net PN ; (b) the O-graph of PN	55
Figure 4.10 (a) Two subnets after dividing PN ; (b) their O-graphs.....	56
Figure 4.11 (a) The net after modifying subnet 1; (b) its O-graph	56
Figure 4.12 (a) The Petri net PN' after merging subnets 1' and 2; its O-graph.....	57
Figure 4.13 A resource allocation system.....	58
Figure 4.14 Five parts of the resource allocation system in Figure 4.13.....	59
Figure 5.1 (a) An example Petri net; (b) the O-graph of the net showed in (a).....	60
Figure 5.2 (a) A Petri net generated from combine to nets; (b) the O-graph of the net showed in (a)	61
Figure 5.3 (a) A Petri net generated from doing a transition mergence; (b) the O-graph of the net showed in (a).....	62

Chapter 1. Introduction

Petri net is a kind of model which can be represented by graphical and mathematical ways. The graphical representations increase their readability while the mathematical representations define them clearly and let them can be analyzed by many kinds of methods ([3], [5], [8]). The powerful modeling ability, two kinds of representations, and many analyzing methods of Petri nets let them very practical to model and analyze many kinds of systems ([4], [5], [8]).

An occurrence graph (O-graph) is the state diagram of a Petri net. The nodes in the O-graph represent the reachable states (also called “marking”) while the arcs represent the variations of states from one to another. Since the O-graphs represent the behaviors of the Petri nets, many kinds of analyzing methods are based on them ([2], [3]).

Two Petri nets can be applied to construct a new Petri net by merging part of their transitions. This thesis presents a technique to reduce the construction time of a new O-graph of a Petri net for the merge. The work is done by reusing the O-graphs of two original nets instead of using the new net. The merge works on two Petri nets at a time, and the policy of merging more than two nets is then introduced. In a large distributed system, there are lots of components. These components can be modeled by Petri nets and our work provides an effective way for the construction and analysis of the Petri nets and their O-graph from their component. Another method can be applied in the incremental analysis of a Petri net by dividing the net into many components actively.

The rest of this thesis is organized as follows. Chapter 2 presents the motivation and

introduces Petri nets, O-graphs, and some other related works. The algorithms of merging transitions between two Petri nets and generating new O-graph are described in Chapter 3 while Chapter 4 introduces some extended applications. Chapter 5 uses an example to trace the execution of the main algorithm in Chapter 3. Chapter 6 concludes the thesis and indicates some future works.



Chapter 2. Background

This chapter introduces the motivation for merging Petri nets in Section 2.1. There are two necessary models is defined for the mergence first in Section 2.2 and 2.3. Some other related works are introduced in Section 2.4.

2.1 Motivation

When workflows are more and more complex, the ability of analyzing workflows when they are edited is needed. One characteristic of this ability is incremental analysis. For example, most code editor softwares nowadays have some incremental analysis abilities to assist a person under coding. A well known function is raising a warning of syntax error in a meaningful time when a programmer is editing the code. Like that, if workflow editor software has the ability of incremental analysis, the errors can be found and corrected in the editing phase.

Since Petri nets is a kind of models which can be applied to model and analyze workflows ([9], [10], [11]), if the ability of incremental analysis can be added on Petri nets, the related analysis for workflows can be applied incrementally, too. So, the technique of merging two existent Petri nets with their transitions is proposed in this thesis to help incremental analysis.

Because Petri nets are used to analyze not only workflows but also many other systems ([3], [8]), this technique might be applied in many other kinds of systems further.

2.2 Petri Net

Petri nets originated from the early work of Carl Adam Petri ([7]). Most readers refer to [8] when applying Petri nets.

Petri nets are a kind of models, which can be used to model many kinds of system and analyzed with techniques. Petri nets used in this thesis are also called marked place/transition nets in the Petri net taxonomy ([3]).

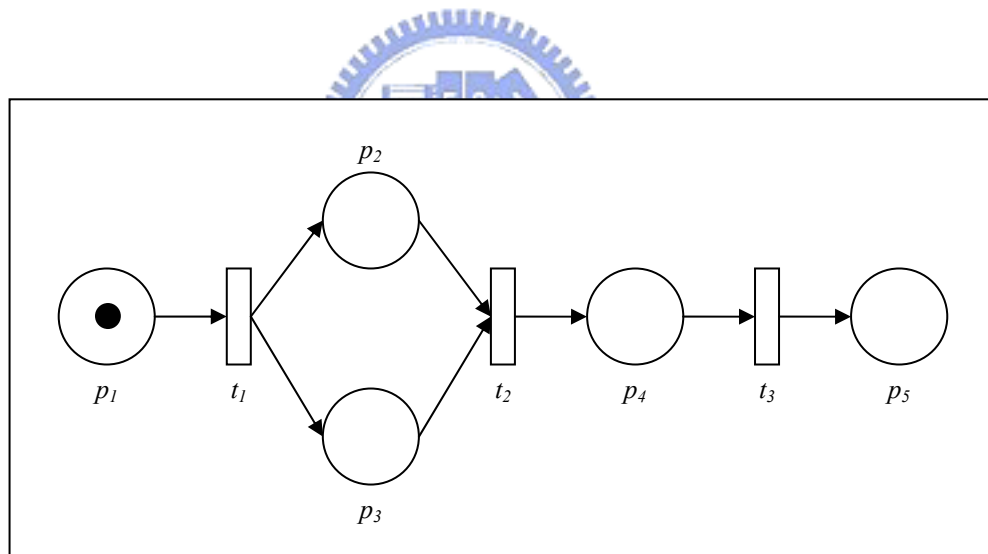


Figure 2.1 An example of a Petri net

A Petri net (also called “net” in this thesis for short) is a directed graph with two kinds of nodes, interpreted as places and transitions, such that no arc connects two nodes of the same kind. A Petri net is also equipped with an initial marking which puts tokens on some places. A marking of a net is a state of this net. Figure 2.1 shows an example of a Petri net. A circle

means a place, a rectangle means a transition, and a dot means a token. If this is the initial state of the Petri net, the initial marking of the Petri net puts a token on the place p_1 .

Definition 2.1: A Petri net is a tuple $PN = (P, T, F, m_0)$ where

- P is a finite set of **places**;
- T is a finite set of **transitions** such that $P \cap T = \emptyset$;
- F is a finite set of **directed arcs**, $F \subseteq (P \cup T) \times (P \cup T)$, satisfying $F \cap (P \times P) = F \cap (T \times T) = \emptyset$;
- m_0 is the **initial marking**, $m_0: P \rightarrow \mathbb{N}$ where $\mathbb{N} = \{0, 1, 2, \dots\}$.

This is the mathematical definition of a Petri net used in this thesis. While this thesis presents a technique about Petri Net, some precise mathematical definitions about Petri Net must be defined.



Definition 2.2:

- A **marking** of a set of place P is a mapping $m: P \rightarrow \mathbb{N}$ where $\mathbb{N} = \{0, 1, 2, \dots\}$.
- A **marking** of a Petri net $PN = (P, T, F, m_0)$ is a marking of a set of place P .

A marking of a net represents a state of this net. It is a function defined from a set of places (or a set of all places in a net) to the set of nonnegative integers which means the number of tokens on each place. For the reason of readability, a marking of a net in an example can be expressed as an array with nonnegative integers which each element in it means the number of tokens on each place. For example, the marking of the net showed in Figure 2.1 can be represented as $(1, 0, 0, 0, 0)$.

Definition 2.3: Let $PN = (P, T, F, m_0)$ be a Petri net.

- For an element $x \in P \cup T$, its **pre-set** $\bullet x$ is defined by

$$\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$$
- and its **post-set** x^\bullet is defined by

$$x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}.$$

This definition defines the notations about the input and output sets of a node (place or transition) in a net. Note that the input and output sets of a place can only be the sets of transitions and the input and output set of a transition can only be the sets of places.



Definition 2.4: A transition t is **enabled** by a marking m if m marks all places in $\bullet t$. In this case t can **occur**. Its occurrence transforms m into the marking m' , defined for each place p by

$$m'(p) \begin{cases} m(p)-1 & \text{if } p \in \bullet t - t^\bullet, \\ m(p)+1 & \text{if } p \in t^\bullet - \bullet t, \\ m(p) & \text{otherwise.} \end{cases}$$

This definition defines the words “enable” and “occur” (also called “fire”). The sentence “ m marks all places in $\bullet t$ ” indicates that the marking m puts at least one token on each place in $\bullet t$. This definition defines the behavior of Petri nets while an occurrence of a transition can transform from one marking to another in a net, that is, from one state to another. Intuitively, when a transition is fired, it removes a token from each input place of it and adds a token on each output place of it.

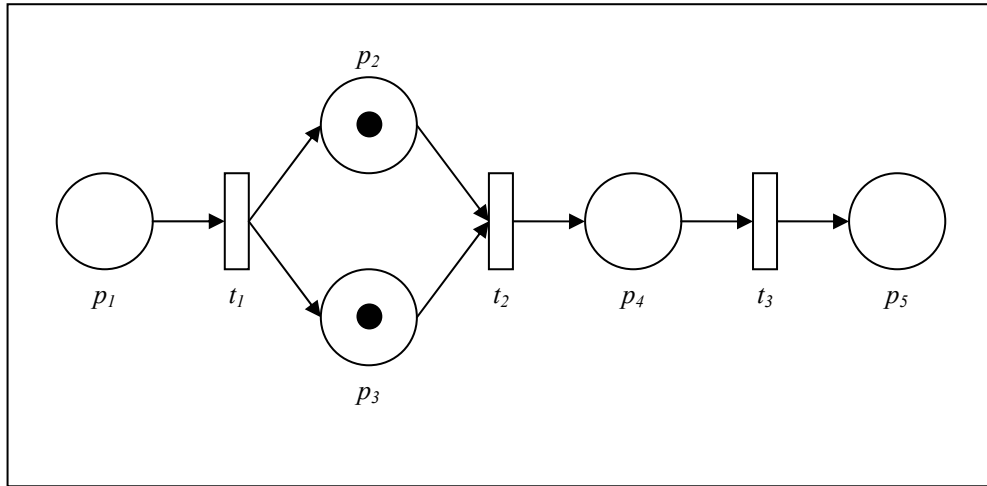


Figure 2.2 The marking after firing t_1

For example, in the net showed in Figure 2.1, if t_1 is fired, the marking of this net becomes Figure 2.2 – a token is removed from p_1 and two tokens is added on p_2 and p_3 separately.



2.3 Occurrence Graph

An occurrence graph (O-graph) can also be called a marking graph because it is a graph uses markings as its nodes. An O-graph of a Petri net is a graph uses all reachable markings from the initial marking as its nodes and uses all possible behaviors as its arcs. A behavior represents a transition being fired on a marking and the marking being transformed to another. Since O-graphs are state diagrams of Petri nets, they provide a very straightforward and easy-to-use method to analyze many properties of Petri nets.

Definition 2.5: An **arc-labeled directed graph** is a tuple $DG = (V, A)$ where

- V is a set of **nodes** (or **vertices**);
- A is a set of **arcs** (or **edges**), $A \subseteq V \times L \times V$ where L is a given set of some labels.

Definition 2.6: The occurrence graph of a Petri net (P, T, F, m_0) is an arc-labeled directed graph $OG = (V, A)$ where

- V is the set of all reachable markings from m_0 ;
- $A = \{(m_1, t, m_2) \in V \times T \times V \mid m_2 \text{ is the marking after firing } t \text{ on } m_1\}$.

These two definitions define the O-graph of a net. The O-graph uses a transition of the net as the label of an arc because a behavior can only correspond to a transition.

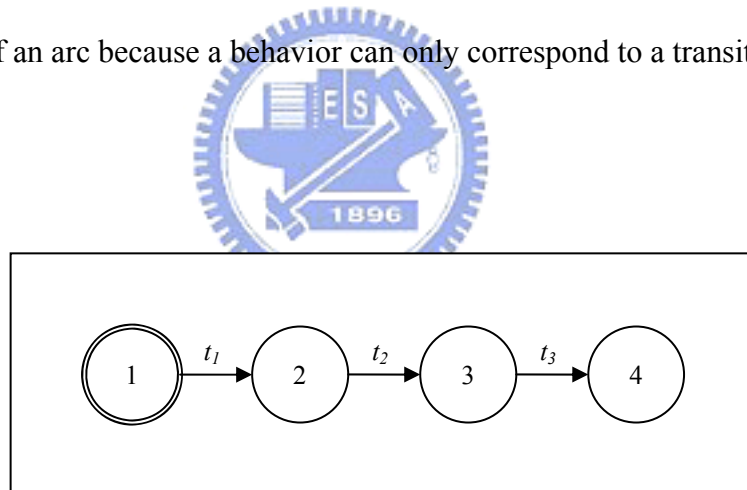


Figure 2.3 The O-graph of the Petri net in Figure 2.1

Figure 2.3 shows the O-graph of the net showed in Figure 2.1. Each circle denoted by a number is a node (i.e. a marking) and the double circle is the initial marking. The numbers in the circles separate these nodes and each of them corresponds to a marking: $1 \rightarrow (1, 0, 0, 0, 0)$, $2 \rightarrow (0, 1, 1, 0, 0)$, $3 \rightarrow (0, 0, 0, 1, 0)$, $4 \rightarrow (0, 0, 0, 0, 1)$.

2.4 Related Works

Julia Padberg [12] applied an extension of high-level replacement systems ([14]) to Petri nets in order to achieve an integration of transformations with the preservation of safety properties. This article proves that the transformations of Petri nets under three kinds of rules can preserve safety properties (something bad should never happen). A safety property can be expressed by a temporal logic formula in terms of numbers of tokens on places. In [13], she extended safety preservation to the transformations of algebraic high-level nets, which is a kind of high-level Petri nets.

Piotr Chrzastowski-Wachtel [6] proposed five basic refinement rules (sequential place split, sequential transition split, OR-split, AND-split, Loop) to refine workflow nets (WF-net, defined in [10], WF-net is a kind of Petri net) using a top-down manner. If a WF-net is started from a single place with no transitions and only the five refinement rules are proposed on it, the resulting WF-net is sound (defined in [6] and [10]). Since the refinement rules are not enough to produce every kinds of WF-net, this article also presents two kinds of non-refinement rules called “communication” and “synchronization” that can also modify WF-nets without losing soundness property.

The above works can preserve some kinds of properties when modifying Petri nets according to some rules. They have an advantage that there is no analysis needed to analyze these properties after modification. But, they also have shortcomings. First, the modifying rules are restricted for some specific properties, e.g., the flexibility is restricted and the integrations are more complicated. Second, although some properties can be preserved after several modifications containing some characteristics, these properties may still exist after

other kinds of modifications.

Since the size of an O-graph is often huge, Pierre Molinaro [16] presented a technique that can generate a huge O-graph in a reasonable amount of time and memory space. This technique is based on the Vector Decision Diagram (VDD) data structure, which is inspired by Binary Decision Diagrams (BDDs) introduced in [18]. A tool ([17]) based on the technique is also proposed.



Chapter 3. Petri Net Transition Mergence

This chapter presents the algorithms merging pairs of Petri net (also called “net” for short) transitions. The discussions are done in both formal and informal ways from the combination¹ of two Petri nets (without merging any pair of transition) to transition mergence². In order to simplify the understandability, a sample case based on two identical nets shown in Figure 3.1 (a) is used through this chapter. Figure 3.1 (b) shows the O-graph of the net in (a) where the number marked on each marking indicates the token on what place in (a). Section 3.1 shows the way to combine two nets and their occurrence graphs (O-graph) while Section 3.2 shows transition mergence with a deletion-based algorithm. Section 3.3 presents a construction-based algorithm corresponding to the above two algorithms.

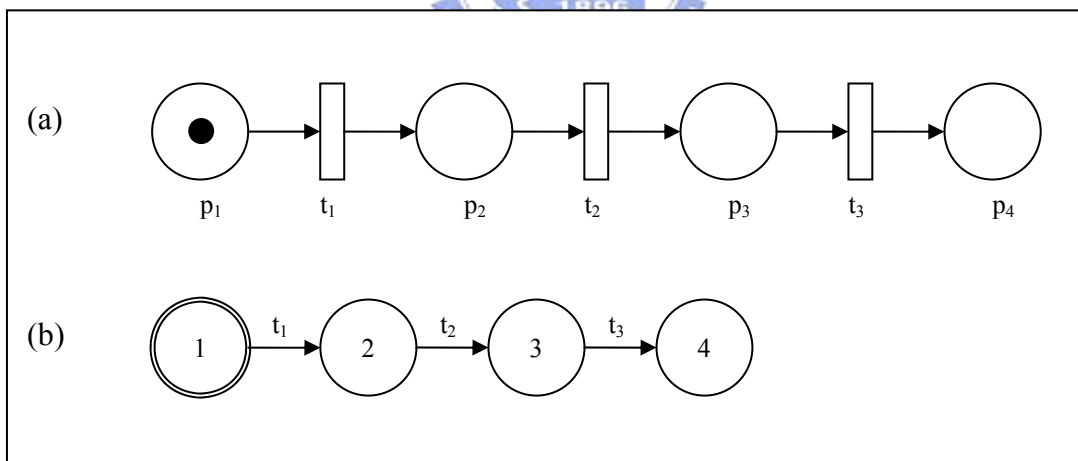


Figure 3.1 (a) An example of a Petri net; (b) the O-graph of (a)

3.1 Combination of Two Petri Nets

¹ Which is defined in Definition 3.1

² Which is defined in Definition 3.5

Our discussion starts from combining two distinct nets to a new one without merging any transition. Figure 3.2 (a) is a simple example of combining two identical nets showed in Figure 3.1 (a). The O-graph of the result net is presented in Figure 3.2 (b). Instead of built from the result net (Figure 3.2 (a)), this O-graph can be built as a matrix graph from original O-graphs directly. For the matrix graph, each row corresponds to the O-graph of subnet 1 while each column presents subnet 2. As shown in Figure 3.2 (b), the two subnets have no mutual intersection, so the variety of marking on the nets can be handled as two isolated parts, denoted by *left* and *right*. In the matrix graph, each node with a *left/right* symbol represents the marking variety of subnet 1 and 2 separately. For example, the node denoted as “1/1” represents the marking that puts a token on p_{1-1} and a token on p_{2-1} ; the node denoted as “1/2” represents the marking that puts a token on p_{1-1} and a token on p_{2-2} . Different with the markings, each arc is denoted by the corresponding transition that is associated with two numbers, which the first represents the subnet it belongs and the second separates the transition from others in the same subnet. For example, if the transition t_{2-1} is fired on the marking denoted as “1/1”, a token moves from p_{2-1} to p_{2-2} while another is still placed on p_{1-1} and the marking of the result net is transformed from “1/1” to “1/2”.

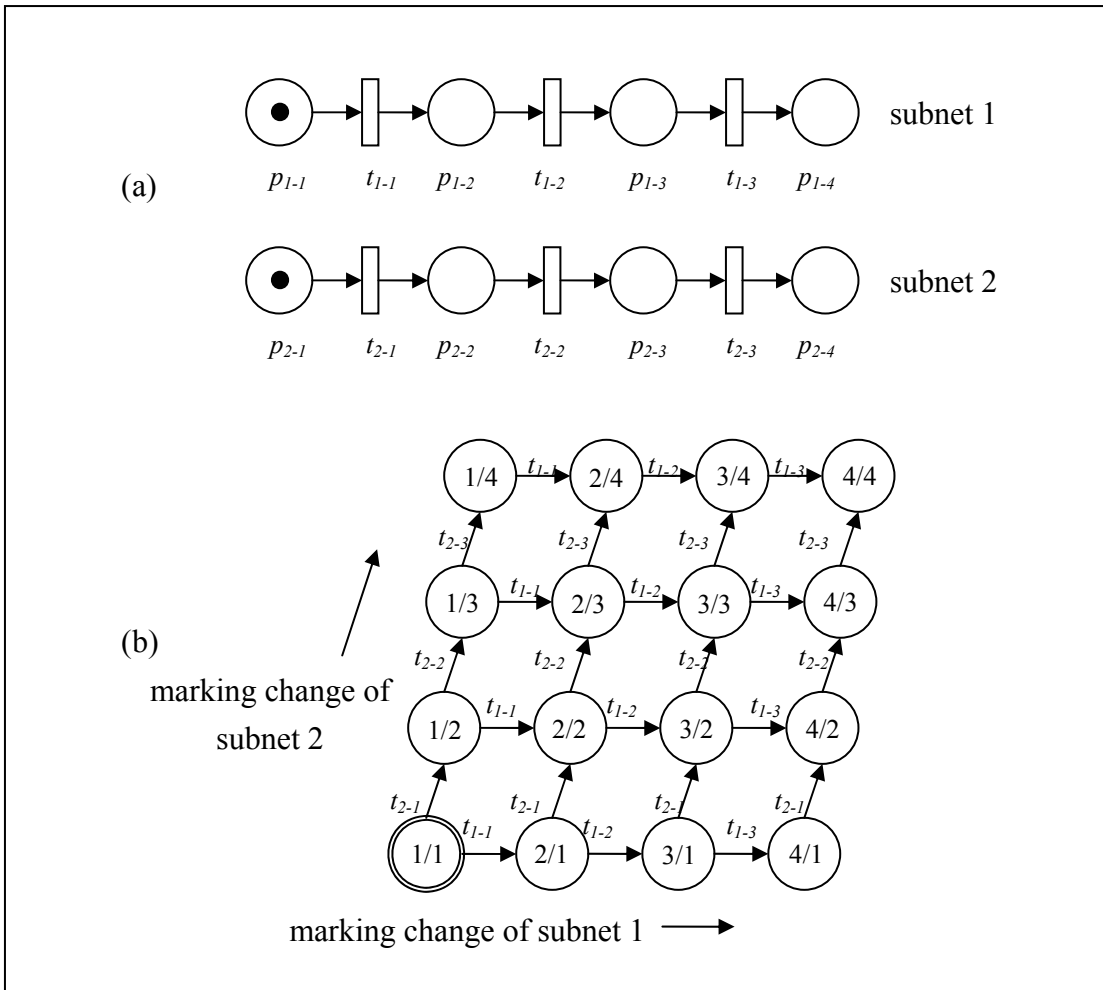


Figure 3.2 (a) The Petri net generated by combining two Petri nets in Figure 3.1 (a); (b) the O-graph of (a)

Since the above two subnets are distinct, a transition belonging to a subnet firing represents a marking change on the direction of the subnet.

We give a formal definition for “combine” as below.

Definition 3.1: Two distinct Petri nets $PN_1 = (P_1, T_1, F_1, m_{0,1})$ and $PN_2 = (P_2, T_2, F_2, m_{0,2})$ are combined to a new Petri net $PN = (P, T, F, m_0)$, where

- (1) $P = P_1 \cup P_2$;
- (2) $T = T_1 \cup T_2$;
- (3) $F = F_1 \cup F_2$;
- (4) $\forall p \in P_1: m_0(p) = m_{0,1}(p)$;
- (5) $\forall p \in P_2: m_0(p) = m_{0,2}(p)$.

PN can be called as the combination of PN_1 and PN_2 .

With the example (Figure 3.2) mentioned above, the combination functions are

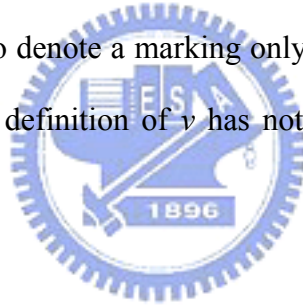
- (1) $P = \{p_{1-1}, p_{1-2}, p_{1-3}, p_{1-4}\} \cup \{p_{2-1}, p_{2-2}, p_{2-3}, p_{2-4}\}$
- (2) $T = \{t_{1-1}, t_{1-2}, t_{1-3}\} \cup \{t_{2-1}, t_{2-2}, t_{2-3}\}$
- (3) F is the union of all the arcs that connect places and transitions in subnet 1 and subnet 2.
- (4) The token numbers on these places $p_{1-1}, p_{1-2}, p_{1-3}, p_{1-4}$ about m_0 (the initial marking of the net combined from the two subnet) are the same as those about $m_{0,1}$ (the initial marking of subnet 1).
- (5) The token numbers on these places $p_{2-1}, p_{2-2}, p_{2-3}, p_{2-4}$ about m_0 are the same as those about $m_{0,2}$ (the initial marking of subnet 2). Items (4) and (5) indicate that m_0 is merged from $m_{0,1}$ and $m_{0,2}$ without other modifications.

The relationship between the O-graphs of PN_1, PN_2 and PN can be discussed now.

Definition 3.2: Let two Petri nets $PN_1 = (P_1, T_1, F_1, m_{0,1})$ and $PN_2 = (P_2, T_2, F_2, m_{0,2})$ be distinct, the O-graphs $OG_1 = (V_1, A_1)$ and $OG_2 = (V_2, A_2)$ belong to PN_1 and PN_2 separately, and M be the set of all markings of $P_1 \cup P_2$. The marking mapping function M_{PN_1,PN_2} which is a function defined from $V_1 \times V_2$ into M , and $\forall v_i \in V_1$ and $v_j \in V_2$:

- (1) $\forall p \in P_1: M_{PN_1,PN_2}(v_i, v_j)(p) = v_i(p);$
- (2) $\forall p \in P_2: M_{PN_1,PN_2}(v_i, v_j)(p) = v_j(p).$

Although this definition uses the notations v_i and v_j to denote the nodes in OG_1 and OG_2 , the two notations still denote the markings of PN_1 and PN_2 , that is, the functions defined from a set of places (P_1 or P_2) to the set of nonnegative integer. The reason for using v instead of m to denote a marking here is that the markings in this definition are treated as the nodes of the O-graphs. In the rest, using v to denote a marking only means that the marking is treated as a node in an O-graph while the definition of v has nothing different with the definition of a marking.



M_{PN_1,PN_2} is a function which merges two reachable markings of PN_1 and PN_2 separately together. For example, while using the marking (1, 0, 0, 0) (node “1” in Figure 3.1 (b)) of the subnet 1 in Figure 3.2 (a) and the marking (0, 1, 0, 0) (node “2” in Figure 3.1 (b)) of the subnet 2 as inputs, the function M of the two subnets outputs the marking (1, 0, 0, 0, 0, 1, 0, 0) (the numbers in the array specify the numbers of tokens of the places in this order ($p_{1-1}, p_{1-2}, p_{1-3}, p_{1-4}, p_{2-1}, p_{2-2}, p_{2-3}, p_{2-4}$)) (node “1/2” in Figure 3.2 (b)).

Proposition 3.1: If $PN = (P, T, F, m_0)$ is the combination of $PN_1 = (P_1, T_1, F_1, m_{0,1})$ and $PN_2 = (P_2, T_2, F_2, m_{0,2})$ and the O-graphs $OG = (V, A)$, $OG_1 = (V_1, A_1)$, and $OG_2 = (V_2, A_2)$ belong to PN , PN_1 , and PN_2 separately, M_{PN_1,PN_2} is a one-to-one and onto function defined from $V_1 \times V_2$ into V

Because PN_1 and PN_2 are distinct, their O-graphs are orthogonal. When PN is only the combination of PN_1 and PN_2 without other modifications, its reachable markings can be generated by combining two reachable markings of PN_1 and PN_2 separately, that is, M_{PN_1,PN_2} .

Definition 3.3: Let two Petri nets $PN_1 = (P_1, T_1, F_1, m_{0,1})$ and $PN_2 = (P_2, T_2, F_2, m_{0,2})$ be distinct, the O-graphs $OG_1 = (V_1, A_1)$ and $OG_2 = (V_2, A_2)$ belong to PN_1 and PN_2 separately, and M be the set of all markings of $P_1 \cup P_2$. The arc mapping function R_{PN_1,PN_2} is a function defined from $(A_1 \times V_2) \cup (A_2 \times V_1)$ into $M \times (T_1 \cup T_2) \times M$, and

- (1) $\forall v \in V_1, a = (v_1, t, v_2) \in A_2: R_{PN_1,PN_2}(a, v) = (M_{PN_1,PN_2}(v, v_1), t, M_{PN_1,PN_2}(v, v_2));$
- (2) $\forall v \in V_2, a_1 = (v_1, t, v_2) \in A_1: R_{PN_1,PN_2}(a, v) = (M_{PN_1,PN_2}(v_1, v), t, M_{PN_1,PN_2}(v_2, v)).$



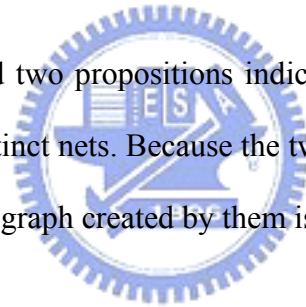
The function R_{PN_1,PN_2} maps the behaviors (arcs in the O-graph) of the nets PN_1 and PN_2 to the behaviors of the combination of PN_1 and PN_2 .

Proposition 3.2: If $PN = (P, T, F, m_0)$ is the combination of $PN_1 = (P_1, T_1, F_1, m_{0,1})$ and $PN_2 = (P_2, T_2, F_2, m_{0,2})$ and the O-graphs $OG = (V, A)$, $OG_1 = (V_1, A_1)$, and $OG_2 = (V_2, A_2)$ belong to PN , PN_1 , and PN_2 separately, R_{PN_1,PN_2} is a one-to-one and onto function defined from $(A_1 \times V_2) \cup (A_2 \times V_1)$ into A .

Because the behaviors in OG_1 are not influenced by what state (marking) PN_2 is at since PN_1 and PN_2 are distinct from each other and each marking in OG_1 can be combined with every markings in OG_2 (i.e. each marking in OG_1 can be mapped to n markings in OG by

M_{PN_1,PN_2} , where n is the number of markings in OG_2 , the markings in OG mapped to the same marking in OG_1 must have those arcs which start from that marking in OG_1 (i.e. each arc a_1 in A_1 has the corresponding arcs $(a_1 \times V_2)$ in A). For example, because transition t_1 is enabled by marking “1” in Figure 3.1, transition t_{1-1} is enabled by “1/1”, “1/2”, “1/3”, and “1/4” in Figure 3.2. The arcs in OG_2 have the same property as those in OG_1 , too. Except those arcs in A which have the relations with A_1 or A_2 (those arcs in the set $R_{PN_1,PN_2}((A_1 \times V_2) \cup (A_2 \times V_1))$), there are no other arcs being made in A since PN_1 and PN_2 have no change (i.e. there are no new markings or transitions to create new behaviors). So R_{PN_1,PN_2} is a one-to-one and onto function defined from $(A_1 \times V_2) \cup (A_2 \times V_1)$ into A .

These two definitions and two propositions indicate a way of building the O-graph for the net combined from two distinct nets. Because the two functions M and R are onto function, it can be guaranteed that the O-graph created by them is complete.



3.2 The Algorithm of Combining Two Petri Nets

These are some structures about Petri net used in the algorithm of Petri net combination:


```

Structure PetriNet {
    Place[]      p;
    Transition[] t;
    FlowRelation[] f;
    Ograph      OG;
}

Structure Place {
    int initialMarking;
    FlowRelation[] fIn;
    FlowRelation[] fOut;
}

Structure Transition {
    FlowRelation[] fIn;
    FlowRelation[] fOut;
}

Structure FlowRelation {
    Null* start;    // point to Place or Transition
    Null* end;      // point to Transition or Place
}

```

Although FlowRelation.start and FlowRelation.end are null pointers, they only point to Place or Transition and do not both point to Place or Transition at the same time to satisfy the definition of Petri net that one directed arc does not connect the same kind of nodes (place or transition).

These are some structures about O-graph:

```

Structure Ograph {
    Vertex[]    v;
    Arc[]       a;
    Vertex[]    pointerTable1D;    // pointer to the vertex by ID
    Vertex[][]  pointerTable2D;    // pointer to the vertex by tmpID1 and 2
}

Structure Vertex {
    Marking m;
    Arc[]   arcOut;
    Arc[]   arcIn;
    // the three kinds of ID will be used in the algorithm
    int ID;    // All vertexes have the different ID from 0 to size - 1.
              // The vertex with the ID "0" is the initial marking.
    // The two kinds of tmpID will map this vertex to the two original markings
    // like an inverse function of  $M$ .
    int tmpID1;
    int tmpID2;
}

Structure Arc {
    Vertex    vStart;
    Vertex    vEnd;
    Transition t;
}

```

Vertex.ID gives each vertex a unique integer ID in an O-graph while Vertex.tmpID1 and Verex.tmpID2 map each vertex (node) to its original vertexes (like an inverse function of M) in two subnets. Ograph.pointerTable1D and Ograph.pointerTable2D point to the vertexes by the three kinds of ID, so they can be used to find the vertexes that are needed in the algorithms without doing unnecessary search.

This is the algorithm of the Petri net combination:

Algorithm 3.1: **PNCombining(PetriNet $PN1$, PetriNet $PN2$)** – combines two Petri nets $PN1$ and $PN2$ and returns the result net.

```

1 // Creat a new empty net then combine  $PN1$  and  $PN2$  to it.
2 PetriNet  $PN$  = new PetriNet;
3
4 // merge the Petri net part
5  $PN.p$  = clone( $PN1.p$  +  $PN2.p$ ); // initial marking (token number in the place) can
// also be copy in this stage.
6  $PN.t$  = clone( $PN1.t$  +  $PN2.t$ );
7  $PN.f$  = clone( $PN1.f$  +  $PN2.f$ );
8
9 // merge the O-graph part
10 int newID = 0;
11 // create the vertexes;
12  $PN.OG.pointerTable2D$  = new Vertex[ $PN1.OG.v.size$ ][ $PN2.OG.v.size$ ];
13 FOR i from 0 to  $PN1.OG.pointerTable1D.size - 1$ 
14      $vx$  =  $PN1.OG.pointerTable1D[i]$ ;
15     FOR j from 0 to  $PN1.OG.pointerTable1D.size - 1$ 
16          $vy$  =  $PN2.OG.pointerTable1D[j]$ ;
17         Vertex  $v$  = new Vertex;
18          $v.m$  =  $vx.m$  +  $vy.m$ ;
19          $v.tmpID1$  = i;
20          $v.tmpID2$  = j;
21          $v.ID$  = newID;
22          $PN.OG.v$  +=  $v$ ;
23          $PN.OG.pointerTable1D[newID]$  =  $v$ ;
24          $PN.OG.pointerTable2D[i][j]$  =  $v$ ;
25         newID++;
26     END
27 END
28

```

```

29 // create the arcs
30 FOR ALL ax ∈ PN1.OG.a
31     FOR ALL vy ∈ PN2.OG.v
32         Arc a = new Arc;
33         a.vStart = pointerTable2D[ax.vStart.ID][vy.ID];
34         PN.OG.pointerTable2D[ax.vStart.ID][vy.ID].arcOut += a;
35         a.vEnd = pointerTable2D[ax.vEnd.ID][vy.ID];
36         PN.OG.pointerTable2D[ax.vEnd.ID][vy.ID].arcIn += a;
37         a.t = ax.t;
38         PN.OG.a += a;
39     END
40 END
41
42 FOR ALL ay ∈ PN2.OG.a
43     FOR ALL vx ∈ PN1.OG.v
44         Arc a = new Arc;
45         a.vStart = pointerTable2D[vx.ID][ay.vStart.ID];
46         PN.OG.pointerTable2D[vx.ID][ay.vStart.ID].arcOut += a;
47         a.vEnd = pointerTable2D[vx.ID][ay.vEnd.ID];
48         PN.OG.pointerTable2D[vx.ID][ay.vEnd.ID].arcIn += a;
49         a.t = ax.t;
50         PN.OG.a += a;
51     END
52 END
53
54 RETURN PN;

```



Lines 4 to 7 in the algorithm unite the places, transitions and flow relations together separately in two input nets to the result net. Lines 11 to 27, like the marking mapping function M , create new vertexes from the original O-graphs. It also builds the vertex IDs and the pointer tables. Lines 29 to 52, like the arc mapping function R , create the arcs from the original O-graphs. The codes between line 30 and line 40 correspond to $(A_1 \times V_2)$ in Definition 3.3 while the codes between line 42 and line 52 correspond to $(A_2 \times V_1)$.

3.3 Merging Petri Net Transitions

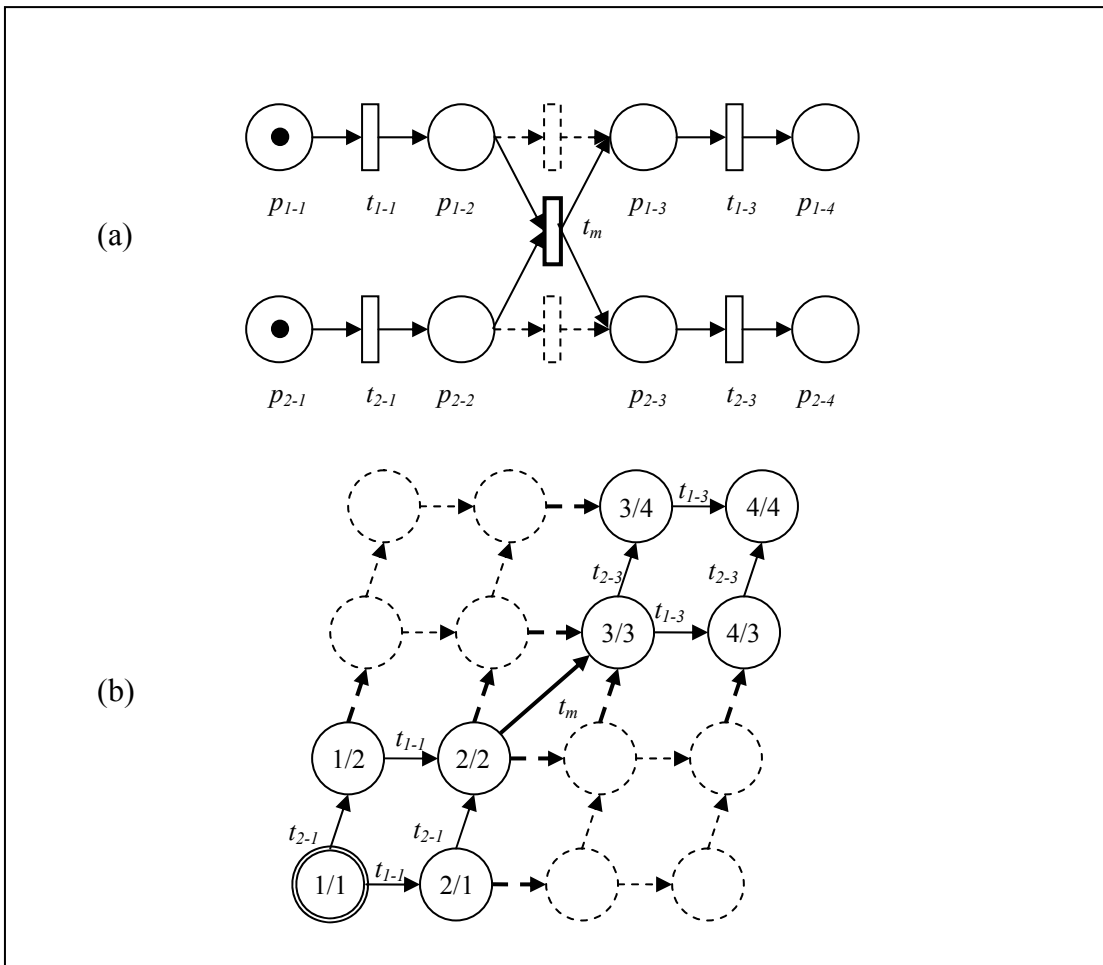
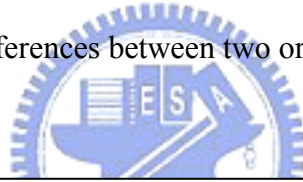


Figure 3.3 (a) Merge two transitions between two nets from Figure 3.1 (a); (b) the O-graph of (a)

Merging two transitions represents inserting synchronization for them. Figure 3.3 (a) is a simple example of merging two transitions in Figure 3.2 (a), where the dashed transitions are merged as a new transition t_m . Figure 3.3 (b) is the O-graph of Figure 3.3 (a) where the node “1/1” with double circle is the initial marking. This O-graph can be generated from the O-graph in Figure 3.2 (b) by removing the dashed nodes and arcs and adding the arc of double black (“2/2”, t_m , “3/3”). When the two dashed transitions are merged, the arcs corresponding to them (the dashed double black arcs in Figure 3.3 (b)) must be removed. Because the node

“2/2” enables the two dashed transitions (the node is the starting point of two arcs corresponding to these two transitions separately), a new arc (the arc of double black in Figure 3.3 (b)) corresponding to the new transition t_m and starting from this node must be added. Node “3/3” are the end of a path that starts from “2/2” and passes through two arcs corresponding to the two dashed transitions separately, so it is the end point of the new arc of double black. Other dashed nodes and arcs are removed since they are not reachable from the initial marking “1/1” after removing the dashed double black arcs and adding the arc of double black.

Above example indicates a rough sketch of the method that generates the new O-graph. Before the discussion of the algorithm generating the new net for merging transitions and its O-graph, we first define the differences between two original subnets and the new net.



Definition 3.4 (merging relation): Let two distinct sets of transitions T_1 and T_2 be held by two distinct Petri nets PN_1 and PN_2 respectively. For a *merging relation* MR from T_1 to T_2 , $MR \subseteq T_1 \times T_2$, and $\forall (ta_1, ta_2), (tb_1, tb_2) \in MR$: if $(ta_1, ta_2) \neq (tb_1, tb_2)$ then $ta_1 \neq tb_1$ and $ta_2 \neq tb_2$.

A merging relation is a relation from the set of all transitions in a net to that in another. All elements in a merging relation are distinct. A transition mergence which is defined in Definition 3.5 merges transitions between two nets according a merging relation.

Definition 3.5 (transition mergence): Let $PN_1 = (P_1, T_1, F_1, m_{0,1})$ and $PN_2 = (P_2, T_2, F_2, m_{0,2})$ be two Petri nets. A Petri net $PN = (P, T, F, m_0)$ generated by a transition mergence according to a merging relation MR from T_1 to T_2 , where

- (1) $P = P_1 \cup P_2$;
- (2) $T = (T_1 \cup T_2 \cup MR) - T'$, where $T' = \{t \mid \exists (t_1, t_2) \in MR: t_1 = t \text{ or } t_2 = t\}$;
- (3) $F = (F_1 \cup F_2 \cup F_1'' \cup F_2'') - (F_1' \cup F_2')$, where

$$F_1' = \{f \mid \exists (t_1, t_2) \in MR: (p_1, t_1) = f \in F_1 \text{ or } (t_1, p_1) = f \in F_1\},$$

$$F_2' = \{f \mid \exists (t_1, t_2) \in MR: (p_2, t_2) = f \in F_2 \text{ or } (t_2, p_2) = f \in F_2\},$$

$$F_1'' = \{f \mid (\exists (t_{a1}, t_{a2}) = t_a \in MR \text{ and } \exists (p_{a1}, t_{a1}) \in F_1; f = (p_{a1}, t_a)) \text{ or}$$

$$(\exists (t_{b1}, t_{b2}) = t_b \in MR \text{ and } \exists (t_{b1}, p_{b1}) \in F_1; f = (t, p_{b1}))\}, \text{ and}$$

$$F_2'' = \{f \mid (\exists (t_{a1}, t_{a2}) = t_a \in MR \text{ and } \exists (p_{a2}, t_{a2}) \in F_2; f = (p_{a2}, t_a)) \text{ or}$$

$$(\exists (t_{b1}, t_{b2}) = t_b \in MR \text{ and } \exists (t_{b2}, p_{b2}) \in F_2; f = (t, p_{b2}))\};$$
- (4) $\forall p \in P_1: m_0(p) = m_{0,1}(p)$;
- (5) $\forall p \in P_2: m_0(p) = m_{0,2}(p)$.

In this definition, both original transitions to be merged are deleted and a corresponding transition is generated as follows. Let two transitions t_1 and t_2 be merged as a transition t in a transition mergence, and then all of t_1 and t_2 are transferred to input/output places of t . A pair of transitions to be merged can not appear inside any of both original nets. There might be more than one pair of transitions to be merged, and the merged transitions in any two pairs must be distinct. Item (4) and (5) indicate that the initial marking of P is the same as P_1 and P_2 's initial marking correspondingly.

The content of the box shown below is the data structures about merging relation used in the following algorithm of transition mergence.

```

Structure MergingRelation {
    TransitionPair[] tp;
}

```

```

Structure TransitionPair {
    Transition t1;
    Transition t2
}

```

Algorithm 3.2: PNTransitionMergenceV1(PetriNet $PN1$, PetriNet $PN2$, MergingRelation MR)

```

1   PetriNet  $PN$  = PNCombining(PetriNet  $PN1$ , PetriNet  $PN2$ );
2
3   FOR  $i$  from 0 to ( $MR.tp.size - 1$ )
4       // make new transition in  $PN$ 
5       Transition  $t$  = mergeTransition( $PN$ ,  $MR.tp[i]$ );
6       // remove or make the arcs
7       FOR ALL ( $arc1 \in PN.OG.a$ ) && ( $arc1.t == MR.tp[i].t1$ )
8           IF ( $\exists arc2 \in arc1.vStart.arcOut, arc2.t == MR.tp[i].t2$ )
9               Arc  $a$  = new Arc;
10               $a.t = t$ ;
11               $a.vStart = arc1.vStart$ ;
12               $arc1.vStart.arcOut += a$ ;
13               $a.vEnd = PN.OG.pointerTable2D[arc1.vEnd.tmpID1][arc2.vEnd.tmpID2]$ ;
14               $PN.OG.pointerTable2D[arc1.vEnd.tmpID1][arc2.vEnd.tmpID2].arcIn += a$ ;
15               $PN.OG.a += a$ ;
16          END
17          removePath( $PN.OG$ ,  $arc1$ );
18      END
19      FOR ALL ( $arc2 \in PN.OG.a$ ) && ( $arc2.t == MR.tp[i].t2$ )
20          removePath( $PN.OG$ ,  $arc2$ );
21      END
22  END
23  regenerateID( $PN.OG$ );
24
25  RETURN  $PN$ ;

```


This algorithm does a transition merge according to the merging relation MR between two Petri nets $PN1$ and $PN2$ by a deletion-based way and returns the result Petri net. All the inputs of it must conform to the definitions about transition mergences.

This algorithm based on Algorithm 3.1 modifies the result O-graph returned from Algorithm 3.1 like the example mentioned at the beginning of this section. Line 1 in this algorithm calls Algorithm 3.1 and gets its result Petri net. The loop from line 3 to line 22 deals one transition pair ($MR.tp[i]$) a time. Line 5 calls the function `mergeTransition` (Algorithm 3.2.1) to delete the old transitions to be merged (i.e. in the transition pair $MR.tp[i]$), add the new transition generated from merging the transition pair, and redirect the flow relations in the nets that connect with them (for example, modify from the net in Figure 3.2 (a) to that in Figure 3.3 (a)). Among the loop, two small loops from line 7 to line 21 remove the arcs corresponding to the old transitions in the transition pair $MR.tp[i]$ and add new arcs on the markings that enable the two old transitions (For example, add the arc t_m in Figure 3.3 (b)). The function `removePath` (Algorithm 3.2.2) removes the input arc from the input O-graph and if the marking following this arc is not pointed by other arcs, it will be removed and its output arcs will be removed by using `removePath` function recursively. The function `regenerateID` (Algorithm 3.2.3) in line 23 can reconstruct an ID for each vertex for later use. This can prevent any unconnection (if a vertex with $ID = i$ exists in OG then a vertex with $ID = i - 1$ also exists for all $i \geq 1$) in the ID system to make sure that another algorithm using this Petri net as its input to work correctly.

These are the three algorithms used in Algorithm 3.2

Algorithm 3.2.1: **mergeTransition(PetriNet PN , TransitionPair tp)**

```
Transition t = new Transition;
FOR ALL fin  $\in$   $tp.t1.fIn$ 
    fin.end = t;
    t.fIn += fin;
END
FOR ALL fout  $\in$   $tp.t1.fOut$ 
    fout.start = t;
    t.fOut += fout;
END
 $PN.t[]$  -=  $tp.t1$ ;
FOR ALL fin  $\in$   $tp.t2.fIn$ 
    fin.end = t;
    t.fIn += fin;
END
FOR ALL fout  $\in$   $tp.t2.fOut$ 
    fout.end = t;
    t.fOut += fout;
END
 $PN.t$  -=  $tp.t2$ ;
 $PN.t$  += t
RETURN t;
```



Algorithm 3.2.2: **removePath(Ograph OG , Arc arc)**

```
 $arc.vEnd.arcIn$  -=  $arc$ ;
 $OG.a$  -=  $arc$ ;
IF ( $arc.vEnd.arcIn == \emptyset$ )
    FOR ALL  $a \in arc.vEnd.arcOut$ 
        removePath( $OG$ ,  $a$ );
    END
     $OG.v$  -=  $arc.vEnd$ ;
     $OG.pointerTable1D[arc.vEnd.ID] = null$ ;
END
```

Algorithm 3.2.3: **regenerateID(Ograph OG)**

```
int i = 0;
FOR j from 0 to  $OG.pointerTable1D.size - 1$ 
    v =  $OG.pointerTable1D[j]$ ;
    v.ID = i;
     $OG.pointerTable1D[i] = v$ ;
    i++;
END
```

Proposition 3.3: If PN is generated by a transition merge between PN_1 and PN_2 according to a merging relation MR where $MR = \emptyset$, then PN is the combination of PN_1 and PN_2 .

This proposition shows the relation and difference between Petri net combination and a transition merge. Actually, the difference is only at the merging relation MR . A transition merge merges one or more pair of transitions between two Petri nets while a combination only deems two distinct Petri nets as one.

Definition 3.6:

- Let PN be generated by a transition merge between PN_1 and PN_2 according to a merging relation MR and PN_b be generated by combining PN_1 and PN_2 , then PN is also deemed as the Petri net after doing a transition merge according to the merging relation MR from PN_1 to PN_2 , and PN_b is also deemed as the Petri net before doing this transition merge.
- With above $MR, PN, PN_b, PN_1, PN_2, OG$, the O-graph of PN , is deemed as the O-graph after doing a transition merge according to the merging relation MR from PN_1 to PN_2 , and OG_b , the O-graph of PN_b , is deemed the O-graph before doing this transition merge.

This definition defines other denominations of PN , PN_b , OG , and OG_b that will be used later for easy understanding.

Proposition 3.4: Let $PN = (P, T, F, v_0)$ be the Petri net after doing transition merge according to the merging relation MR from PN_1 to PN_2 , $PN_b = (P_b, T_b, F_b, v_{b0})$ be the Petri net before doing this transition merge, $OG = (V, A)$ be the O-graph of PN , and $OG_b = (V_b, A_b)$ be the O-graph of PN_b .

- (1) $v_0 = v_{b0}$;
- (2) $V \subseteq V_b$;
- (3) $\forall a = (v_1, t, v_2) \in A$: if $t \notin MR$, then $a \in A_b$;
- (4) $\forall a = (v_1, t, v_2) \in A$: if $t \in MR$, then $a \notin A_b$;
- (5) $\forall a_b = (v_1, t_b, v_2) \in A_b$: if $(\exists (t_1, t_2) \in MR: t_1 = t_b \text{ or } t_2 = t_b)$, then $a_b \notin A$;
- (6) $v_{b0} \in V$;
- (7) $\forall v_b \in V_b$: if $(\nexists (v_1, t, v_b) \in A)$ and $v_b \neq v_{b0}$, then $v_b \notin V$;
- (8) $\forall a_b = (v_1, t, v_2) \in A_b$: if $v_1 \notin V$, then $a_b \notin A$;
- (9) $\forall v_1 = M_{PN_1, PN_2}(v_{a1}, v_{a2}) \in V, (t_1, t_2) \in MR$: iff $(\exists v_3 = M_{PN_1, PN_2}(v_{b1}, v_{a2}) \in V_b: (v_1, t_1, v_3) \in A_b)$ and $(\exists v_4 = M_{PN_1, PN_2}(v_{a1}, v_{b2}) \in V_b: (v_1, t_2, v_4) \in A_b), \exists (v_1, (t_1, t_2), v_2) \in A$ where $v_2 = M_{PN_1, PN_2}(v_{b1}, v_{b2})$;
- (10) $\forall a_b = (v_1, t, v_2) \in A_b$: if $v_1 \in V$ and $(\nexists (t_1, t_2) \in MR: t_1 = t \text{ or } t_2 = t)$, then $a_b \in A$;
- (11) $\forall v_b \in V_b$: if $\exists (v_1, t, v_b) \in A$, then $v_b \in V$.

Proof:

- (1) Because the definitions of the initial markings of PN and PN_b are the same (see Definition 3.1 and Definition 3.5), $v_0 = v_{b0}$.
- (2) Since the firing behavior of each new transition t is the same as the behavior of firing the two original transitions merged as t one by one, the action of merging transitions does not produce new reachable markings. So, V is the subset of V_b .

- (3) Because V is the subset of V_b , all the arcs starting at the markings belonging to V but not corresponding to the new transitions can be found in A_b .
- (4) Because the transitions belonging to MR are new transitions that can not be found in T_b , the arcs corresponding to them can not be found in A_b .
- (5) The transitions that are merged must be removed after the transition mergence, thus each arc corresponding to them does not exist in the O-graph after the transition mergence.
- (6) $v_0 \in V$ and $v_{b0} = v_0$, therefore $v_{b0} \in V$.
- (7) Since there is no arc pointing to the marking v_b in OG , such markings do not exist in OG but the initial marking.
- (8) If v_l does not exist in the new O-graph OG , the arcs starting from it to v_2 can not occur in OG .
- (9) The new transition (t_1, t_2) can be enabled by a marking only when both original transitions t_1 and t_2 are enabled by the marking. Each this kind of marking v_l must add an output arc a corresponding to (t_1, t_2) and no other arc corresponding to (t_1, t_2) must be added in OG . Since the firing behavior of (t_1, t_2) is the same as the behavior of firing t_1 and t_2 one by one, the output marking of arc a is $M_{PN_1, PN_2}(v_{b1}, v_{b2})$ when the input marking of arc a is $M_{PN_1, PN_2}(v_{a1}, v_{a2})$, firing t_1 on v_{a1} in PN_1 changes the marking to v_{b1} , and firing t_2 at v_{a2} in PN_2 changes the marking to v_{b2} .
- (10) Because v_l is still a reachable marking of PN and t is still the transition in PN with the same input and output places, when t can be fired at v_l and changes the marking to v_2 in PN_b , the same behavior can occur in PN .
- (11) This is straightforward in the definition of graph.

To help understand the proposition, we made the items that mapping the notations in this proposition to the targets in the example shown at the beginning of the section.

- PN : the net showed in Figure 3.3 (a)
- OG : the O-graph showed in Figure 3.3 (b)
- PN_b : the net showed in Figure 3.2 (a)
- OG_b : the O-graph showed in Figure 3.2 (b)
- MR : $\{(t_{1-2}, t_{2-2})\}$ (note that $t_m = (t_{1-2}, t_{2-2})$)
- In item (3), a can be (“1/1”, t_{1-1} , “2/1”), (“1/2”, t_{1-1} , “2/2”), (“1/1”, t_{2-1} , “1/2”), (“2/1”, t_{2-1} , “2/2”), (“3/3”, t_{1-3} , “4/3”), (“3/4”, t_{1-3} , “4/4”), (“3/3”, t_{2-3} , “3/4”), or (“4/3”, t_{2-3} , “4/4”)
- In item (4), a can be (“3/3”, t_m , “3/4”)
- In item (5), a_b can be (“2/1”, t_{1-2} , “3/1”), (“2/2”, t_{1-2} , “3/2”), (“2/3”, t_{1-2} , “3/3”), (“2/4”, t_{1-2} , “3/4”), (“1/2”, t_{2-2} , “1/3”), (“2/2”, t_{2-2} , “2/3”), (“3/2”, t_{2-2} , “3/3”), or (“4/2”, t_{2-2} , “4/3”)
- In item (7), v_b which satisfies the “if” condition can be “1/3”, “1/4”, “2/3”, “2/4”, “3/1”, “3/2”, “4/1”, or “4/2”
- In item (8), a_b which satisfies the “if” condition can be (“1/3”, t_{1-1} , “2/3”), (“1/4”, t_{1-1} , “2/4”), (“2/3”, t_{1-2} , “3/3”), (“2/4”, t_{1-2} , “3/4”), (“1/3”, t_{2-3} , “2/4”), (“2/3”, t_{2-3} , “2/4”), (“3/1”, t_{1-3} , “4/1”), (“3/2”, t_{1-3} , “4/2”), (“3/1”, t_{2-1} , “3/2”), (“4/1”, t_{2-1} , “4/2”), (“3/2”, t_{2-2} , “3/3”), or (“4/2”, t_{2-2} , “4/3”)
- In item (9), for the mapping from $(v_1, (t_1, t_2), v_2)$ to (“2/2”, (t_{1-2}, t_{2-2}) , “3/3”), v_{a1} is the marking (0, 1, 0, 0) in subnet 1 and v_{a2} is the marking (0, 1, 0, 0) in subnet 2. When v_{b1} is the marking (0, 0, 1, 0) in subnet 1 and v_{b2} is the marking (0, 0, 1, 0) in subnet 2, the “if” condition can be satisfied, so (“2/2”, (t_{1-2}, t_{2-2}) , “3/3”) $\in A$
- In item (10), using the mapping from a_b to (“2/1”, t_{2-1} , “2/2”) as an example, when “2/1” $\in V$ and t_{2-1} is not belonging to any pair in MR , (“2/1”, t_{2-1} , “2/2”) $\in A$

- In item (11), using the mapping from v_b to “2/2” as an example, when (“2/1”, t_{2-1} , “2/2”) $\in A$, “2/2” $\in V$

In Proposition 3.4 (2) ~ (4), it can be observed that V can be built by removing some markings in V_b and A can be built by removing some arcs in A_b and adding some arcs corresponding to the new transitions. Items (5) and (8) detect the arcs to be removed, while item (10) certifies the removes with (5) and (8) are complete. Item (7) detects the markings to be removed, while (11) certifies the removes with (7) are complete. Item (9) can show the arcs to be added for a new transition between two markings.

Proposition 3.4 can prove the correctness of the deletion-based algorithm of transition mergence (Algorithm 3.2). Line 8 to line 16, in this algorithm, adds the arcs corresponding to the new transition discussed in item (9) of this proposition. Line 17 and line 20 use Algorithm 3.2.2 to remove the arcs and the markings for items (5), (7), and (8) in this proposition when the O-graph before the transition mergence contains no loop. In the algorithm, if all arcs pointing to a marking are removed then it is removed; if the input marking of an arc is removed then the arc is removed. So, if the O-graphs containing loops, all markings and arcs in these loops are unable to be removed even when they are unreachable from the initial marking. Although some methods which can check the reachability of the nodes can deal with the problem, they are not needed to be added in the technique since the problem does not exist in another construction-based algorithm discussed in Section 3.4.

3.4 Transition Mergence Algorithm

The construction-based algorithm for transition mergence is discussed in this section, where the two algorithms discussed in last two sections are used to explain the concept of transition mergences.

Algorithm 3.3: PNTransitionMergenceV2(PetriNet PN_1 , PetriNet PN_2 , MergingRelation MR)

```

1  generate result Petri net  $PN$ ;
2   $s1pool = \{\emptyset\}$  is a set of marking;
3   $s2pool = \{\emptyset\}$  is a set of marking;
4  add  $M_{PN_1,PN_2}(m_1, m_2)$  into O-graph  $OG$  and  $s1pool$  where  $m_1$  is the initial marking of  $PN_1$  and
    $m_2$  is the initial marking of  $PN_2$ ;
5  WHILE ( $s1pool$  or  $s2pool$  is not empty)
6      WHILE ( $s1pool$  is not empty)
7          remove a marking  $v = M_{PN_1,PN_2}(v_1, v_2)$  from  $s1pool$ ;
8          FOR ALL ( $a_1 = (v_1, t_1, w_1) \in$  the set of output arcs of  $v_1$  in  $PN_1$ )
9              IF ( $t_1$  is to be merged)
10                 label  $a_1$  on  $v$ ;
11             ELSE
12                 add arc  $R_{PN_1,PN_2}(a_1, v_2)$  into  $OG$ ;
13                 IF ( $M_{PN_1,PN_2}(w_1, v_2)$  does not exist in  $OG$ )
14                     add  $M_{PN_1,PN_2}(w_1, v_2)$  into  $OG$  and  $s1pool$ ;
15                 END
16             END
17         END
18         add  $v$  into  $s2pool$ ;
19     END

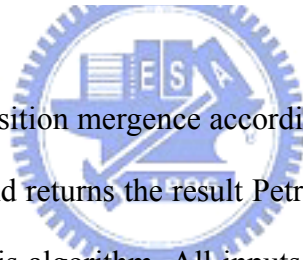
```



```

20   WHILE ( $s2pool$  is not empty)
21       remove a marking  $v = M_{PN1,PN2}(v_1, v_2)$  from  $s2pool$ ;
22       FOR ALL ( $a_2 = (v_2, t_2, w_2) \in$  the set of output arcs of  $v_2$  in  $PN_2$ )
23           IF ( $t_2$  is to be merged)
24               IF ( $v$  has been labeled by an arc  $a_1 = (v_1, t_1, w_1)$  where  $(t_1, t_2) \in MR$ )
25                   add arc  $(v, (t_1, t_2), M_{PN1,PN2}(w_1, w_2))$  into  $OG$ ;
26                   IF ( $M_{PN1,PN2}(w_1, w_2)$  does not exist in  $OG$ )
27                       add  $M_{PN1,PN2}(w_1, w_2)$  into  $OG$  and  $s1pool$ ;
28                   END
29               END
30           ELSE
31               add arc  $R_{PN1,PN2}(a_2, v_1)$  into  $OG$ ;
32               IF ( $M_{PN1,PN2}(v_1, w_2)$  does not exist in  $OG$ )
33                   add  $M_{PN1,PN2}(v_1, w_2)$  into  $OG$  and  $s1pool$ ;
34               END
35           END
36       END
37   END
38   return  $PN$  and  $OG$ 

```



This algorithm does a transition mergence according to the merging relation MR between two Petri nets $PN1$ and $PN2$ and returns the result Petri net and its O-graph, which is built by a construction-based way in this algorithm. All inputs must conform to the definitions about transition mergences. The detail of this algorithm is shown in the appendix Algorithm A.

This algorithm builds the new O-graph OG starting from the initial marking and continuing generating the arcs and the markings that can reach from the markings in OG (i.e. the arcs conforming with Proposition 3.4 (9) or (10) and the markings conforming with Proposition 3.4 (14)). It has not the problem about loops existing in Algorithm 3.2 because the unreachable markings are never put into OG to build unreachable paths to them self.

Line 1 in the algorithm constructs places, transitions, and the flow relations of PN while the rest of the Lines generate OG , the O-graph of PN . Lines 2 and 3 initial two pools used in

the rest of this algorithm. *s1pool* records the markings that have not been examined with the O-graph belonging to PN_1 while *s2pool* records the markings that have not been examined with the O-graph belonging to PN_2 but have already been examined with that belonging to PN_1 . Line 4 creates the initial marking merged from the two initial markings in two original nets then put it into *s1pool*. Lines 5 to 37 describe a loop which jumps out when *s1pool* and *s2pool* are both empty, which indicates that all current markings in *OG* are already examined with two original nets and *OG* has been completely built.

The loop from line 6 to line 19 is executed if any marking exists in *s1pool*. It removes a marking $v = M_{PN_1,PN_2}(v_1, v_2)$ from *s1pool* at line 7 and deals with this marking at a time. The loop from line 8 to line 17 checks an arc a_1 that starts at v_1 in PN_1 corresponding to v at a time. If a_1 is corresponding to a transition to be merged, line 10 labels a_1 on v sets a related flag recording this transition pair and the arc $R_{PN_1,PN_2}(a_1, v)$ is not created (by Proposition 3.4 (5), the arc does not exist in the new O-graph). If a_1 is corresponding to a transition that is not to be merged (the arc satisfies the conditions of Proposition 3.4 (10)), a related new arc $R_{PN_1,PN_2}(a_1, v)$ is generated and put into *OG* at line 12. If the marking pointed by $R_{PN_1,PN_2}(a_1, v)$ (the marking satisfies the conditions of Proposition 3.4 (11)) does not exist in *OG*, it is generated and added into *OG* and *s1pool* at line 14. v is added into *s2pool* at line 18 after the examining mentioned above is done.

The loop from line 20 to line 35 is executed if any marking exists in *s2pool*. It removes a marking $v = M_{PN_1,PN_2}(v_1, v_2)$ from *s2pool* and deals with this marking at a time. Although this loop is like that from line 6 to line 19 but to check with PN_2 , there are still some different points. Lines 21 and 22 check that if the transition t_2 corresponded by a_2 must be merged with t_1 and there is an arc (v_1, t_1, w_1) labeled on v (this situation satisfies the conditions of Proposition 3.4 (9) where $v_1 = v$). If so, line 23 creates a new arc $(v, (t_1, t_2), M_{PN_1,PN_2}(w_1, w_2))$

where w_2 is the marking pointed by a_2 in PN_2 . If $M_{PN_1, PN_2}(w_1, w_2)$ does not exist in OG , it is generated and added into OG and $sIpool$ at line 25.

Since only v_l belongs to V in Proposition 3.4 items (9) and (10) and (v_l, t, v_b) belongs to A in item (11), all other conditions in these items only relate with the elements in the O-graph before the transition merge (which can create from the two original O-graphs by functions M and R), the current state of the new O-graph under the algorithm operated does not influence the examination of a marking corresponding to v_l according to the three items. So, each marking only needs to be checked with the two original nets one time. This algorithm uses the two pools to make sure that each marking in new O-graph is checked exactly one time by two original nets. Since all markings in an O-graph are reachable from its initial marking, there is no marking or arc being lost by the algorithm.

Now, Algorithm 3.3 can be compared with the original algorithm ([2]) that constructs the O-graph directly from its corresponding Petri net.

Algorithm 3.4

- input: a Petri net
 - output: its O-graph
- ```

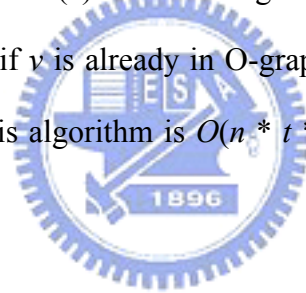
1 Waiting = ∅;
2 Node(m0);
3 WHILE (Waiting ≠ ∅)
4 select a node v1 ∈ Waiting;
5 FOR ALL (t, v2) ∈ Next(v1)
6 Node(v2);
7 Arc(v1, t, v2);
8 END
9 Waiting = Waiting - {v1};
10 END

```

$\text{Node}(v)$  is a procedure that creates a new node (marking)  $v$  and adds  $v$  into  $Waiting$ , a set

of markings, and the new O-graph. If  $v$  exists in the O-graph already, the procedure does nothing.  $\text{Next}(v_l)$  is used to denote the set of all possible “next moves” from  $v_l$ . I.e.,  $\text{Next}(v_l) = \{(t, v_2) \in T \times V \mid v_2 \text{ is the marking after firing } t \text{ on } v_l\}$ .  $\text{Arc}(v_l, t, v_2)$  is a procedure that creates a new arc  $(v_l, t, v_2)$  and adds it to the new O-graph.

Let  $n$  be the number of nodes in the result O-graph. The loop from line 3 to line 10 repeats  $n$  times because each node contained in the result O-graph should be added into *Waiting* and examined one time in this loop. Let  $t$  be the number of transitions in the Petri net. Since the set  $\text{Next}(v_l)$  must be known at line 5, each transition in the result net must be examined whether it is enabled by  $v_l$  or not. This indicates that the complexity of finding the set  $\text{Next}(v_l)$  is  $O(t)$ . Because  $\text{Node}(v)$  does nothing if  $v$  exists in the result O-graph already, the procedure  $\text{Node}(v)$  checks if  $v$  is already in O-graph and the complexity of this search is  $O(n)$ . So, the complexity of this algorithm is  $O(n * t * n)$ . This complexity is also shown in [16].



Now, check the complexity of Algorithm 3.3. Since all markings that must be added into the result O-graph should be checked one time with two subnets, the loop from line 6 to 19 and that from 20 to 36 will run  $n * 2$  times totally when considering the loop from 5 to 37. Let  $t_1$  be the number of transitions in  $PN1$  and  $t_2$  be the number of transitions in  $PN2$ . The loop from line 8 to line 17 runs  $t_1$  times at most because each marking in the O-graph of  $PN1$  can have  $t_1$  arcs as output at most. With similar reason, the loop from line 20 to line 34 runs  $t_2$  times at most. While  $t_1, t_2 \leq t$ , the complexity of the two loops are  $O(t)$ . Note that each marking usually enables a small part of all the transitions only, so the running times of the two loops usually are much fewer than  $t$ . The complexity of Algorithm 3.3 is  $O(n * t)$ .

Although the complexity of Algorithm 3.3 is smaller than the complexity of the original algorithm, there are still situations which influence the time of generating the final result Petri net by merging a set of nets while applying a transition merge have some differences with applying the original algorithm. These situations are discussed in Section 4.2.



## Chapter 4. More about Transition Mergence

This chapter introduces two kinds of extension about transition mergences and discusses two significant applications can be applied with transition mergences. Section 4.1 indicates that the O-graph can also be generated by Algorithm 3.3 with a modification when merging places with a restriction. Section 4.2 shows how to build a Petri net from merging more than two nets using transition mergences and the influences of the merging sequence. Section 4.3 discusses two significant applications can be applied with the technique for transition mergence.

### 4.1 Merging Places and Transitions



Place mergence between two Petri nets is not as easy as transition mergence. For example, the new markings unreachable in the combination net might become reachable after place mergence. Since these markings must be found, the condition is more complex. On the other hand, a restriction for the input/output transitions in a place mergence might makes the O-graph of the result net extremely like the O-graph built by a transition mergence according to these pairs of transitions.

Figure 4.1 (a) is the result net of merging two pairs of transitions between two subnets in Figure 3.2 (a) while Figure 4.1 (b) is the net after merging a pair of places in Figure 4.1 (a). The shapes of O-graphs of these two Petri nets are the same, as in Figure 4.1 (c). In Figure 4.1 (a), the input transitions of places  $p_1$  and  $p_2$  are merged together. So are their output transitions.

The numbers of tokens on both two places in the initial marking are zero. So, the token numbers in  $p_1$  and  $p_2$  are the same no matter what series of transitions are fired. Since both places have the same token number in each reachable marking in the O-graph, they can be merged together and the result place has the same token number with them in each marking. So, the difference between the two O-graphs of the two nets in (a) and (b) is only the content of each marking. The O-graph of the net in (b) can be built by removing the token numbers corresponding to  $p_1$  and  $p_2$  in each marking in the O-graph of the net in (a), and adding the same number for  $p$  to the marking.



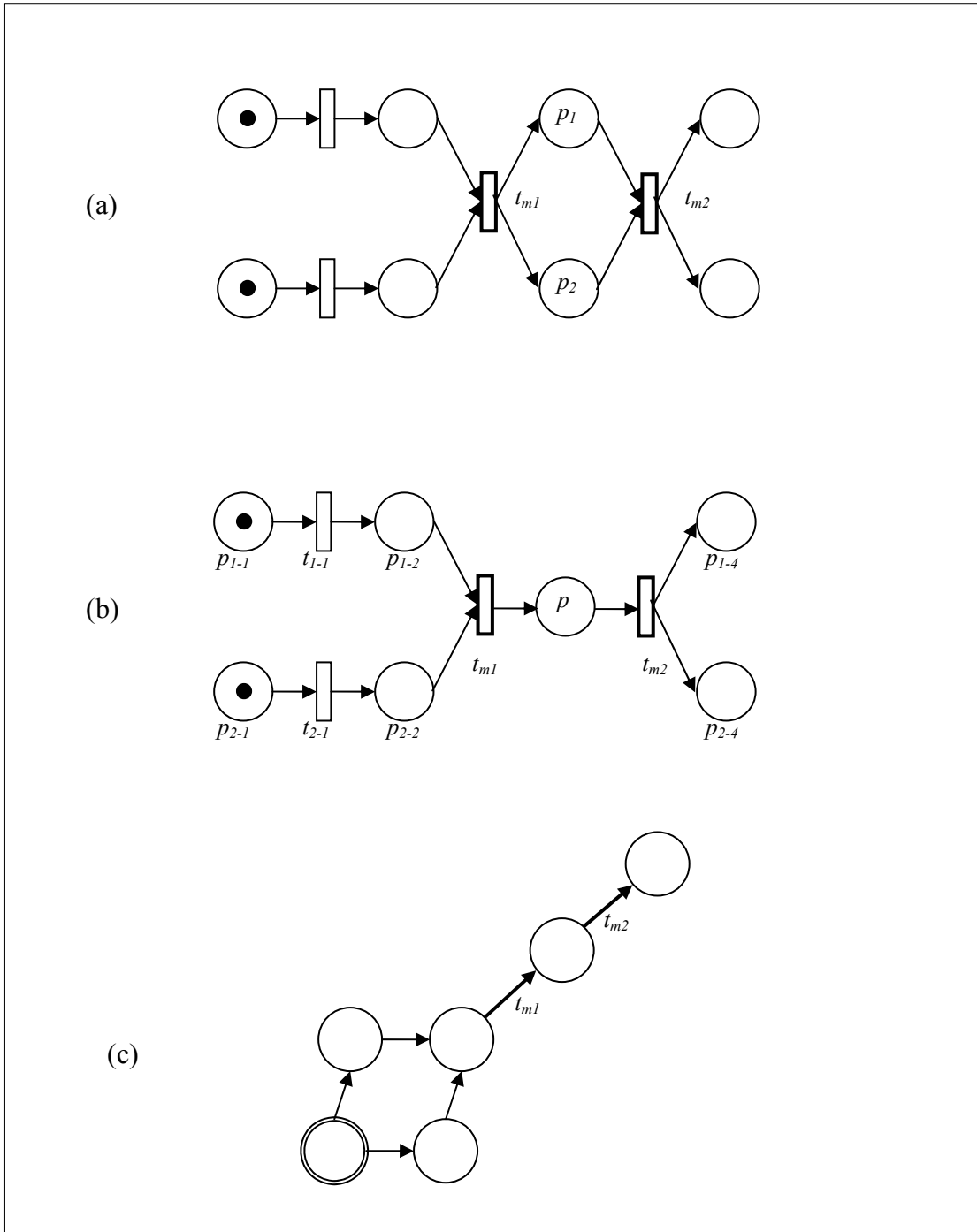


Figure 4.1 (a) The result net of merging two pairs of transitions between two subnets from Figure 3.2 (a); (b) merging two places in (a); (c) the O-graph of (a) and (b)

The definitions about merging pairs of places and transitions are shown below.



Definition 4.1 (merging relation of Petri net): Let  $PN_1 = (P_1, T_1, F_1, m_{0,1})$  and  $PN_2 = (P_2, T_2, F_2, m_{0,2})$  be two distinct Petri nets. For a merging relation of Petri net  $MRP$  from  $PN_1$  to  $PN_2$ , where

- (1)  $MRP \subseteq (P_1 \cup T_1) \times (P_2 \cup T_2)$ ;
- (2)  $\forall (e_1, e_2) \in MRP$ : if  $e_1 \in P_1$ , then  $e_2 \in P_2$ ;
- (3)  $\forall (e_1, e_2) \in MRP$ : if  $e_1 \in T_1$ , then  $e_2 \in T_2$ ;
- (4)  $\forall (a_1, a_2), (b_1, b_2) \in MRP$ : if  $(a_1, a_2) \neq (b_1, b_2)$ , then  $a_1 \neq b_1$  and  $a_2 \neq b_2$ ;
- (5)  $\forall p_1 \in P_1, p_2 \in P_2$ : if  $(p_1, p_2) \in MRP$ , then  $m_{0,1}(p_1) = m_{0,2}(p_2)$ ;
- (6)  $\forall p_1 \in P_1, p_2 \in P_2, t_1 \in T_1$ : if  $(p_1, p_2) \in MRP$  and  $(t_1, p_1) \in F_1$ , then  $\exists t_2 \in T_2$ :  $(t_2, p_2) \in F_2$  and  $(t_1, t_2) \in MRP$ ;
- (7)  $\forall p_1 \in P_1, p_2 \in P_2, t_1 \in T_1$ : if  $(p_1, p_2) \in MRP$  and  $(p_1, t_1) \in F_1$ , then  $\exists t_2 \in T_2$ :  $(p_2, t_2) \in F_2$  and  $(t_1, t_2) \in MRP$ ;
- (8)  $\forall p_1 \in P_1, p_2 \in P_2, t_2 \in T_2$ : if  $(p_1, p_2) \in MRP$  and  $(t_2, p_2) \in F_2$ , then  $\exists t_1 \in T_1$ :  $(t_1, p_1) \in F_1$  and  $(t_1, t_2) \in MRP$ ;
- (9)  $\forall p_1 \in P_1, p_2 \in P_2, t_2 \in T_2$ : if  $(p_1, p_2) \in MRP$  and  $(p_2, t_2) \in F_2$ , then  $\exists t_1 \in T_1$ :  $(p_1, t_1) \in F_1$  and  $(t_1, t_2) \in MRP$ .

This definition is modified from the definition of merging relation (Definition 3.4). It extends to the places while merging relation contains the relationships between transitions only. Besides the restriction that all elements in the relation are distinct (item (4)), there are several restrictions added as follows. The components in the relation are of the same type (items (2) and (3)). Two places contained in the relation have the same token number in the initial marking (item (5)). An input and output transitions of two places have the relation if the two places have the relation (items (6) to (9)).

Definition 4.2 (transition and place mergence): Let  $PN_1 = (P_1, T_1, F_1, m_{0,1})$  and  $PN_2 = (P_2, T_2, F_2, m_{0,2})$  be two distinct Petri nets. A Petri net  $PN = (P, T, F, m_0)$  generated by a transition and place mergence (T&P mergence) according to a merging relation of Petri net  $MRP$  from  $PN_1$  to  $PN_2$ , where

- (1)  $P = (P_1 \cup P_2 \cup P'') - P'$ , where  $P' = \{p \mid (p \in P_1 \text{ and } \exists p' \in P_2: (p, p') \in MRP) \text{ or } (p \in P_2 \text{ and } \exists p' \in P_1: (p', p) \in MRP)\}$  and  $P'' = \{p \mid p = (p_1, p_2) \in MRP \text{ where } p_1 \in P_1 \text{ and } p_2 \in P_2\}$ ;
- (2)  $T = (T_1 \cup T_2 \cup T'') - T'$ , where  $T' = \{t \mid (t \in T_1 \text{ and } \exists t' \in T_2: (t, t') \in MRP) \text{ or } (t \in T_2 \text{ and } \exists t' \in T_1: (t', t) \in MRP)\}$  and  $T'' = \{t \mid t = (t_1, t_2) \in MRP \text{ where } t_1 \in T_1 \text{ and } t_2 \in T_2\}$ ;

- (3)  $F = (F_1 \cup F_2 \cup FT_1' \cup FT_2' \cup FT_1'' \cup FT_2'' \cup FP' \cup FP'') - (F_1' \cup F_2')$ , where

$$F_1' = \{f \mid \exists (a_1, a_2) \in MRP: (a_1, b_1) = f \in F_1 \text{ or } (b_1, a_1) = f \in F_1\},$$

$$F_2' = \{f \mid \exists (a_1, a_2) \in MRP: (a_2, b_2) = f \in F_2 \text{ or } (b_2, a_2) = f \in F_2\},$$

$$FT_1' = \{(p_1, t) \mid \exists t_1 \in T_1, t_2 \in T_2, (t_1, t_2) = t \in MRP, \text{ and } (p_1, t_1) \in F_1 \text{ and } \nexists (p_1, p_2) \in MRP\},$$

$$FT_2' = \{(p_2, t) \mid \exists t_1 \in T_1, t_2 \in T_2, (t_1, t_2) = t \in MRP, \text{ and } (p_2, t_2) \in F_2 \text{ and } \nexists (p_1, p_2) \in MRP\},$$

$$FT_1'' = \{(t, p_1) \mid \exists t_1 \in T_1, t_2 \in T_2, (t_1, t_2) = t \in MRP, \text{ and } (t_1, p_1) \in F_1 \text{ and } \nexists (p_1, p_2) \in MRP\},$$

$$FT_2'' = \{(t, p_2) \mid \exists t_1 \in T_1, t_2 \in T_2, (t_1, t_2) = t \in MRP, \text{ and } (t_2, p_2) \in F_2 \text{ and } \nexists (p_1, p_2) \in MRP\},$$

$$FP' = \{(t, p) \mid \exists p_1 \in P_1, p_2 \in P_2, (p_1, p_2) = p \in MRP, \text{ and } (t_1, t_2) = t \in MRP: (t_1, p_1) \in F_1 \text{ and } (t_2, p_2) \in F_2\},$$

$$FP'' = \{(p, t) \mid \exists p_1 \in P_1, p_2 \in P_2, (p_1, p_2) = p \in MRP, \text{ and } (t_1, t_2) = t \in MRP: (p_1, t_1) \in F_1 \text{ and } (p_2, t_2) \in F_2\};$$

- (4)  $\forall p_1 \in P_1$ : if  $(\nexists p_2 \in P_2: (p_1, p_2) \in MRP)$ , then  $m_0(p_1) = m_{0,1}(p_1)$ ;
- (5)  $\forall p_2 \in P_2$ : if  $(\nexists p_1 \in P_1: (p_1, p_2) \in MRP)$ , then  $m_0(p_2) = m_{0,2}(p_2)$ ;
- (6)  $\forall (p_1, p_2) = p \in MRP$ : if  $p_1 \in P_1$  and  $p_2 \in P_2$ , then  $m_0(p) = m_{0,1}(p_1) = m_{0,2}(p_2)$ .

Like a transition mergence, all transitions to be merged (contained in a pair of transitions in the  $MRP$ ) must be removed and the new transitions generated by merging a pair of transitions must be added (item (2)). All places to be merged (contained in a pair of places in

the  $MRP$ ) must be removed and the new places generated by merging a pair of places must be added (item (1)). All the arcs connected with an old place or transition to be merged must be reconnected to the new place or transition generated by merging this old place or transition with another (item (3)). After a T&P mergence, the token number of every place not merged in the initial marking is not changed (items (4) and (5)) while the token number of a new place generated by merging a pair of old places is the same as the token number of an old place (item (6)) (note that the token numbers about the two places to be merged are the same by the definition of T&P mergence).

As the example net in Figure 4.1 (b) generated by a T&P mergence according to a merging relation of Petri net  $MRP = \{p = (p_{1-3}, p_{2-3}), t_{m1} = (t_{1-2}, t_{2-2}), t_{m2} = (t_{1-3}, t_{2-3})\}$  from subnet 1 to subnet 2 in Figure 3.2 (a),  $P' = \{p_{1-3}, p_{2-3}\}$ ;  $P'' = \{p\}$ ;  $T' = \{t_{1-2}, t_{2-2}, t_{1-3}, t_{2-3}\}$ ;  $T'' = \{t_{m1}, t_{m2}\}$ ;  $F_1' = \{(p_{1-2}, t_{1-2}), (t_{1-2}, p_{1-3}), (p_{1-3}, t_{1-3}), (t_{1-3}, p_{1-4})\}$ ;  $F_2' = \{(p_{2-2}, t_{2-2}), (t_{2-2}, p_{2-3}), (p_{2-3}, t_{2-3}), (t_{2-3}, p_{2-4})\}$ ;  $FT_1' = \{(p_{1-2}, t_{m1})\}$ ;  $FT_2' = \{(p_{2-2}, t_{m1})\}$ ;  $FT_1'' = \{(t_{m2}, p_{1-4})\}$ ;  $FT_2'' = \{(t_{m2}, p_{2-4})\}$ ;  $FP' = \{(t_{m1}, p)\}$ ;  $FP'' = \{(p, t_{m2})\}$ .

**Definition 4.3:** Let  $PN_1 = (P_1, T_1, F_1, m_{0,1})$  and  $PN_2 = (P_2, T_2, F_2, m_{0,2})$  be two distinct Petri nets,  $MR$  is a merging relation from  $T_1$  to  $T_2$ , and  $MRP$  is a merging relation of Petri net from  $PN_1$  to  $PN_2$ .  $MR$  is called the *pre-merging relation* of  $MRP$  if and only if  $(\forall (t_1, t_2) \in MR: (t_1, t_2) \in MRP)$  and  $(\forall (t_1, t_2) \in MRP: \text{if } t_1 \in T_1 \text{ and } t_2 \in T_2, \text{ then } (t_1, t_2) \in MR)$ .

If a merging relation  $MR$  is a pre-merging relation of a merging relation of Petri net  $MRP$ ,  $MR$  is the same as the set of all pairs of transitions in  $MRP$ .

Proposition 4.1: Let  $PN_1 = (P_1, T_1, F_1, m_{0,1})$  and  $PN_2 = (P_2, T_2, F_2, m_{0,2})$  be two distinct Petri nets,  $MRP$  be a merging relation of Petri net from  $PN_1$  to  $PN_2$ ,  $MR$  be the pre-merging relation of  $MRP$ ,  $PN_p$  be the Petri net generated by the transition merge according to  $MR$ ,  $PN$  be the Petri net generated by the T&P merge according to  $MRP$ ,  $OG_p = (V_p, A_p)$  be the O-graph of  $PN_p$ , and  $OG = (V, A)$  be the O-graph of  $PN$ .

Let  $VF$  be a function defined from  $V_p$  into  $V$  where  $\forall v_p \in V_p$ :

- $\forall p \in P_1 \cup P_2$ : if  $(\overline{A}(p_1, p_2) \in MRP: p_1 = p$  or  $p_2 = p)$ , then  $VF(v_p)(p) = v_p(p)$ ;
- $\forall (p_1, p_2) \in MRP$ : if  $(p_1 \in P_1$  and  $p_2 \in P_2)$ , then  $VF(v_p)((p_1, p_2)) = v_p(p_1) = v_p(p_2)$

and  $AF$  be a function defined from  $A_p$  into  $A$  where  $\forall a_p = (v_1, t, v_2) \in A_p$ :

$AF(a_p) = (VF(v_1), t, VF(v_2))$ , then

- (1)  $VF$  is a one to one and onto function defined from  $V_p$  into  $V$  and
- (2)  $AF$  is a one to one and onto function defined from  $A_p$  into  $A$ .



The difference between  $PN$  and  $PN_p$  is only at those pairs of places in  $MRP$ . Since the input and output transitions of two places in each pair of places in  $MRP$  are the same (Definition 4.1 items (6) ~ (9)) and the token numbers of them have not difference in the initial marking (Definition 4.1 item (5)), the token numbers of the two places in each reachable marking of  $PN_p$  (i.e. each marking in  $V_p$ ) are the same. The merge of the two places does not change the whole behavior. So, when merging the pairs of places in  $MRP$ , the only change from  $OG_p$  to  $OG$  is that each node should be transformed by  $VF$  and each arc should be reconnected to the new node by  $AF$ .

From Proposition 4.1, it can be known that  $OG$  can be generated from  $OG_p$  by  $VF$  and  $AF$ . Since  $OG_p$  can be generated from the O-graphs of  $PN_1$  and  $PN_2$  by Algorithm 3.3, the technique of generating  $OG$  from  $PN_1$  and  $PN_2$  is found.

## 4.2 Merging Policy

Since a T&P merge merges two Petri nets to one, the final result net of a sequence of T&P merges is a node, i.e., the root of the binary tree according to the merges. The nodes including those generated in the merges, and their binary relationships can be deemed as a merging tree.

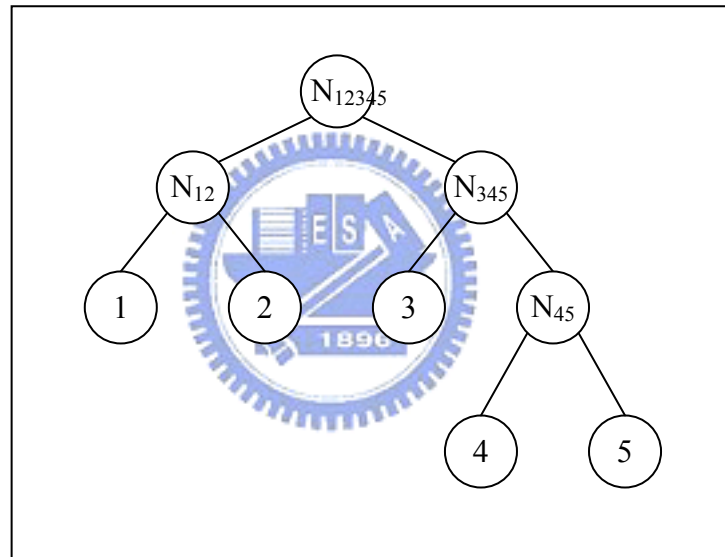
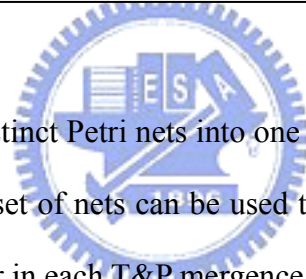


Figure 4.2 An example of a merging tree

Figure 4.2 is an example of a merging tree. Each node in the tree is a Petri net and each non-leaf node is generated by a T&P merge between its two branching nodes. For example, net (node)  $N_{12}$  is generated by a T&P merge between net 1 and net 2 while net  $N_{345}$  is generated by a T&P merge between net 3 and net  $N_{45}$ . The root (net  $N_{12345}$ ) is the final result net. The numbers following “N” indicates this node is merged from what leaf nodes.

Definition 4.4: Let  $PNS$  be a set of distinct Petri nets,  $T$  be the union of all sets of transitions in the Petri nets belonging to  $PNS$ ,  $P$  be the union of all sets of places in the Petri nets belonging to  $PNS$ , and  $F$  be the union of all sets of directed arcs in the Petri nets belonging to  $PNS$ . A *total merging relation* for  $PNS$  denoted as  $TMR$  where

- (1)  $TMR \subseteq (T \cup P) \times (T \cup P)$ ;
- (2)  $\forall t \in T, p \in P: (t, p) \notin TMR$  and  $(p, t) \notin TMR$ ;
- (3) If  $(e_1, e_2) \in TMR$ , then  $(e_2, e_1) \in TMR$ ;
- (4) If  $(e_1, e_2) \in TMR$  and  $(e_2, e_3) \in TMR$ , then  $(e_1, e_3) \in TMR$ ;
- (5)  $\forall t_1, t_2 \in T$ : if  $t_1$  belongs to the same Petri net with  $t_2$ , then  $(t_1, t_2) \notin TMR$ .
- (6)  $\forall p_1, p_2 \in P$ : if  $p_1$  belongs to the same Petri net with  $p_2$ , then  $(p_1, p_2) \notin TMR$ .
- (7)  $\forall p_1, p_2 \in P, t_1 \in T$ : if  $(p_1, p_2) \in TMR$  and  $(t_1, p_1) \in F$ , then  $\exists t_2 \in T: (t_2, p_2) \in F$  and  $(t_1, t_2) \in TMR$ ;
- (8)  $\forall p_1, p_2 \in P, t_1 \in T$ : if  $(p_1, p_2) \in TMR$  and  $(p_1, t_1) \in F$ , then  $\exists t_2 \in T: (p_2, t_2) \in F$  and  $(t_1, t_2) \in TMR$ ;



When merging a set of distinct Petri nets into one net using a series of T&P mergences, a total merging relation for this set of nets can be used to define which pairs of transitions and places must be merged together in each T&P mergence.

Definition 4.5: Let  $PNS$  and  $TMR$  be as above,  $S$  be the set of all transitions and places in  $PNS$ .

$MRP$  is a merging relation of Petri net following  $TMR$ , where

- $MRP$  is a merging relation of Petri net from  $PN_1$  to  $PN_2$  where  $PN_1$  and  $PN_2$  are merged by two distinct subsets of  $PNS$  separately;
- all elements belonging to  $S_1 \times S_2$  belong to  $TMR$ , where  $S_1$  and  $S_2$  are the subset of  $S$ .

Note: In this definition, if  $PN_1 \in PNS$ ,  $S_1 = \{e_1\}$ ; if  $PN_2 \in PNS$ ,  $S_2 = \{e_2\}$ .

Definition 4.6: Let  $PNS$  and  $TMR$  be as above.

$MT$  is a merging tree constructed based on  $TMR$ , where

- $MT$  is a merging tree;
- the set of all leaves of  $MT$  is equal to  $PNS$ ;
- $\forall NLPN \in$  the set of all non-leaf nodes of  $MT$ :  $NLPN$  is generated by a T&P merge according to a merging relation of Petri net following  $TMR$

Definition 4.7: Let  $TMR$  be as above and  $PN$  be a Petri net.  $PN$  is generated according to  $TMR$ , where  $PN$  is the root of a merging tree constructed based on  $TMR$ .

Above three definitions show how to generate a Petri net by merging a set of distinct nets according to a total merging relation for the set of nets. Note that there is one or more trees constructed based on the same total merging relation. This indicates that the sequence of T&P merge can be changed under above definitions. Figure 4.3 is a merging tree representing different sequences of T&P merge from those in Figure 4.2. If both trees are constructed based on the same total merging relation, the two roots (both denoted by “ $N_{12345}$ ”) in the two figure are the same.

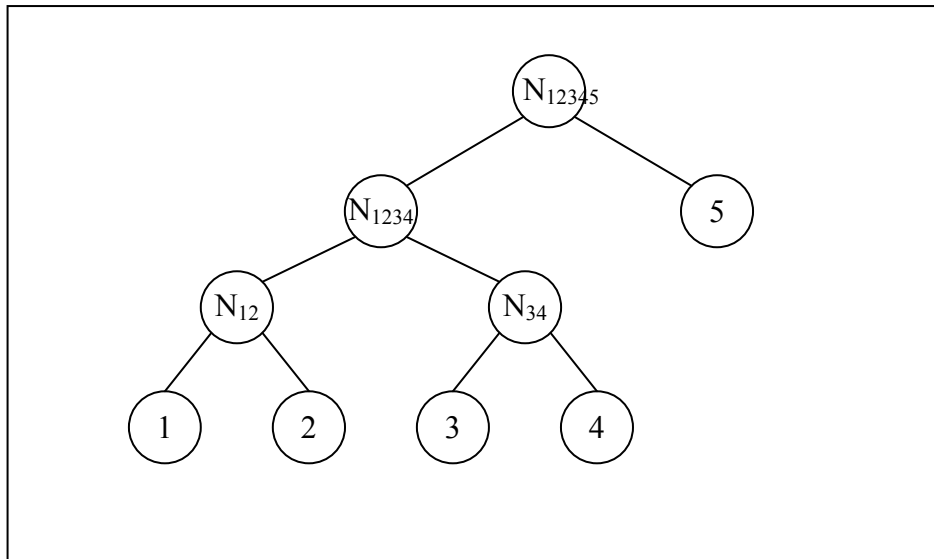


Figure 4.3 Another example of merging tree

An interesting characteristic can be observed from this example. In Figure 4.2, if net 5 is modified, nets  $N_{45}$ ,  $N_{345}$ , and  $N_{12345}$  and their O-graphs must be reconstructed when the O-graph of the final result net ( $N_{12345}$ ) is used. In Figure 4.3, once net 5 is modified, only net  $N_{12345}$  and its O-graph have to be re-constructed when they are needed. Once a net is modified and the root and its O-graph are needed, the closer the net modified is to the root, the fewer of the nets need be reconstructed. So, a merging tree containing the modified net closer to the root needs less reconstructing time.

### 4.3 Possible Applications

In the application layer, there are two significant directions that the technique for T&P mergence can be applied to: merging components into one and dividing a Petri net into a set of nets for incremental analysis.



### 4.3.1 The Analysis Following the Component Mergence

When a system is composed by several components, these components may have several synchronizations between them. Since a T&P mergence between two nets is a kind of synchronization and Petri nets can be used to model and analyze many kinds of systems ([4], [5], [8]), T&P mergences can be used to model some kinds of relationships between two components. When the generating time of the O-graph of a Petri net can be reduce, the times of many Petri net analysis methods using O-graphs ([2], [3]) might be reduced. As presented previously, it can reduce the generation time of the O-graph of a Petri net  $PN$  by merging the O-graphs of the original subnets together, instead of using  $PN$  directly. On the other hand, such mergences might be used to help to analyze the system.

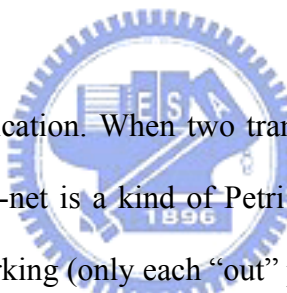


Figure 4.4 shows an application. When two transitions  $t_1$  and  $t_2$  in two *WorkFlow nets* (WF-net) (defined in [10], WF-net is a kind of Petri net) are merged, whether all reachable markings can reach the end marking (only each “out” place has a token) of the whole net must be checked. In Figure 4.5, the O-graph of the whole net generated by applying Algorithm 3.3 is displayed. With some graphical algorithms ([19]) (in this case, search according to the reverse arcs), the six reachable markings (marked with two gray oval) which can not reach the end marking can be found. Since the generation time of the O-graph can be reduced, the total analyzing time can be reduced. Similarly, the time of various analyses can be reduced.

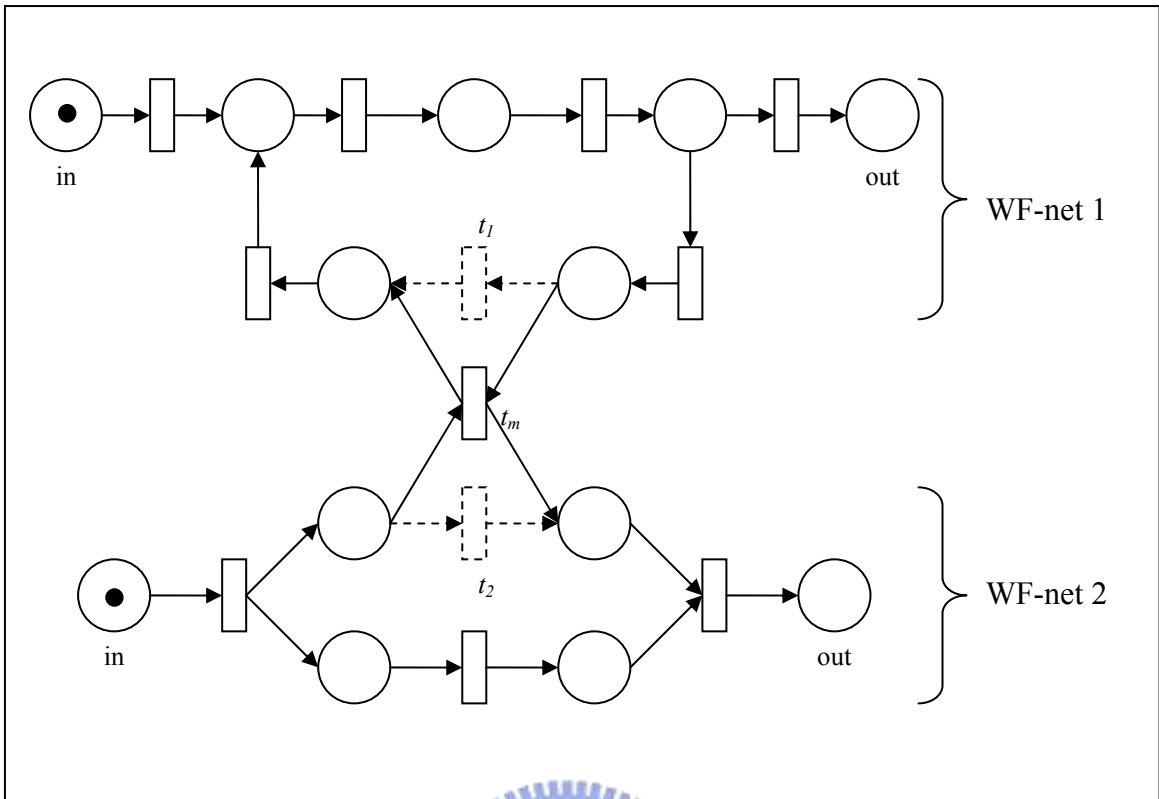


Figure 4.4 Merging two transitions between two WorkFlow nets

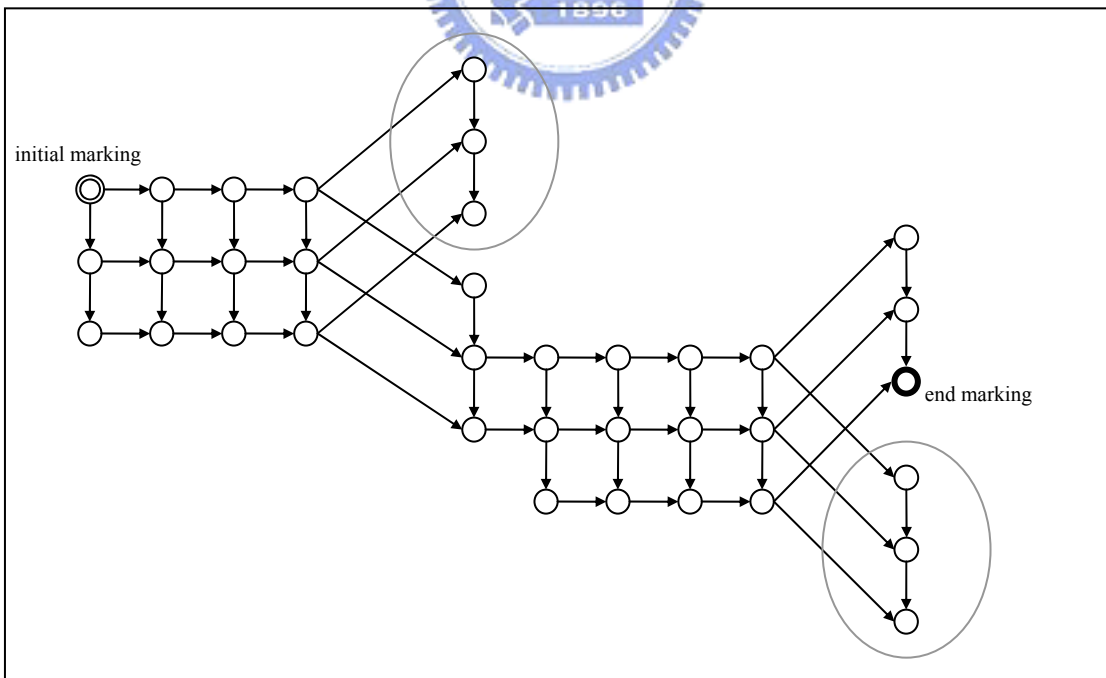


Figure 4.5 The O-graph of the whole net shown in Figure 4.4

### 4.3.2 Incremental Analysis for Edition in a Petri Net

During edition of a Petri net, the net can be deemed as the mergence of its subnets. In the merging tree corresponding to the Petri net, the O-graphs of each node can be generated by the technique for T&P mergence. When a change is happened in one or more subnets during edition, the O-graphs of these nets related to each modification must be re-constructed while the rest are not.

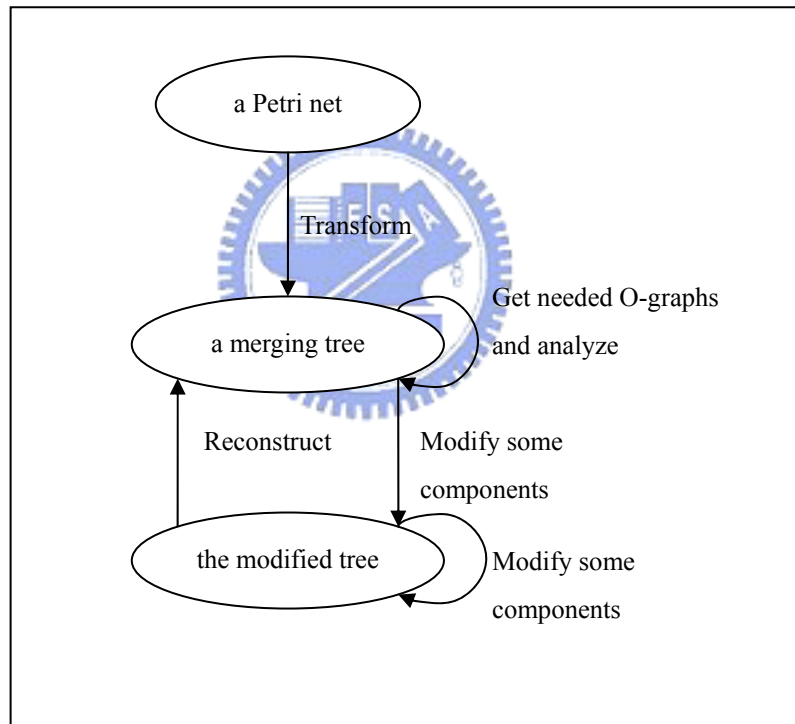


Figure 4.6 A method for incremental analyzing a Petri net

Figure 4.6 shows a method for incremental analysis using the technique for T&P mergence. Let an oval in the figure be a state. The related activities associated with the arcs are:

- Transform: transform the input Petri net into a merging tree;

- Modify some components: modify, add, or delete some nets in the tree without reconstructing their O-graphs;
- Reconstruct: reconstruct the O-graphs of the modified nets;
- Get needed O-graphs and analyze: get the O-graphs needed and analyze them.

First, a Petri net  $PN$  edited and analyzed is transformed into a merging tree whose root is  $PN$ . According to the definition of T&P mergence, each concurrent flow can be split as an isolated component. In other words, the “And Split” and “And Join” components can be divided as in Figure 4.7. When the problem of infinite markings can be dealt with, a “Sequence” can be divided as Figure 4.8 because its second half can generate infinite tokens. On the other hand, if the technique for place mergence is found, the division can be applied on places.

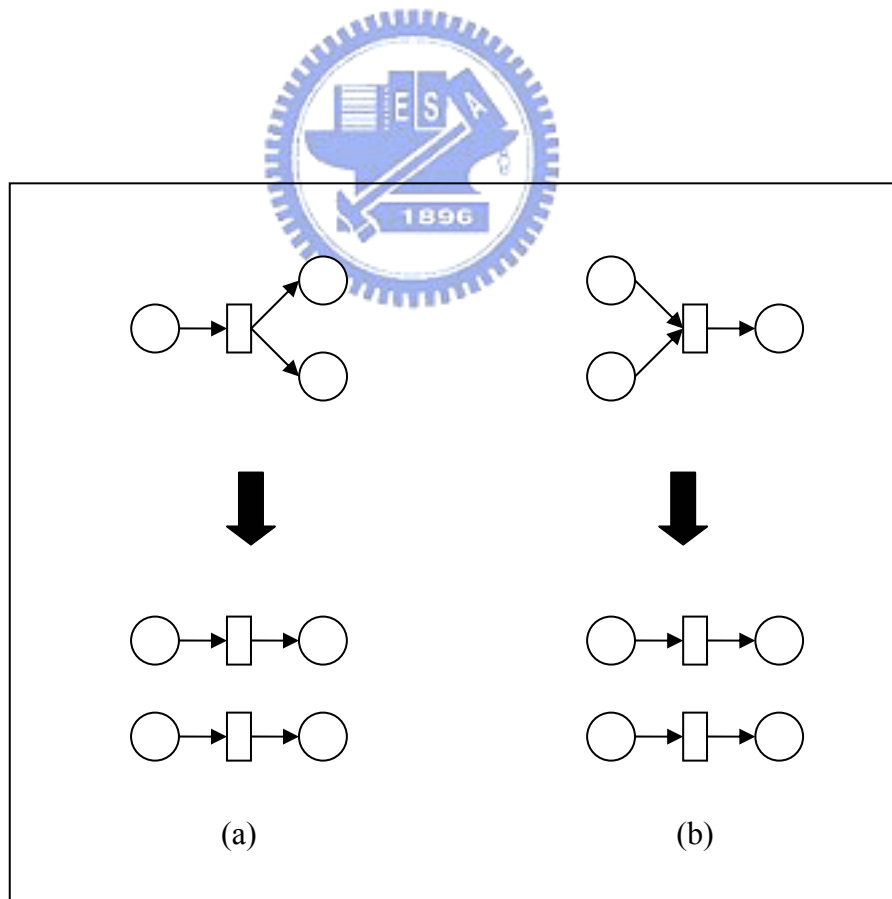


Figure 4.7 (a) “And Split” division; (b) “And Join” division

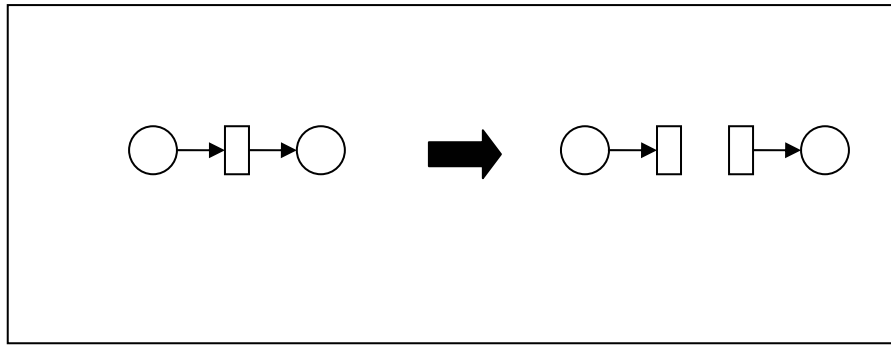
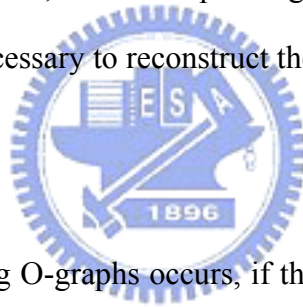


Figure 4.8 “Sequence” division

When a change happens in the structure of  $PN$ , it is necessary to perform “Modify some components” in Figure 4.6. Thus, the corresponding components in the merging tree are modified. Note that it is not necessary to reconstruct the O-graphs of the nodes of the merging tree here.



When an analysis applying O-graphs occurs, if the state is “the modified tree” in Figure 4.6, i.e., the tree has been modified but its O-graphs have not been renewed, activity “Reconstruct” is performed to rebuild the O-graphs of the modified component. This activity applies the technique for T&P merge discussed previously and then the state becomes “a merging tree”, where activity “Get needed O-graphs and analyze” can be executed to analyze the properties of  $PN$  using the O-graphs in the merging tree.

Since not all O-graphs must be reconstructed when “Reconstruct” is executed, the O-graphs not to be renewed can be preserved. This characteristic has been discussed in Section 4.2.

A simple example of the method is shown below. Figure 4.9 (a) is a Petri net  $PN$ . First, it

is divided into two subnets as in Figure 4.10 (a). A merging tree with two leaves corresponding to these two nets and a root for  $PN$  can be constructed. When  $PN$  is modified as in Figure 4.12 (a), by add a loop on top half, and it is denoted as  $PN'$  and the corresponding component, *subnet 1*, is modified as *subnet 1'* in Figure 4.11 (a). When an analysis applying O-graph of  $PN'$  occurs, this O-graph can be generated by reconstructing the O-graph of *subnet 1'* (Figure 4.11 (b)) and merging the O-graphs of the two subnets into one (Figure 4.12 (b)) with the technique for T&P mergence.

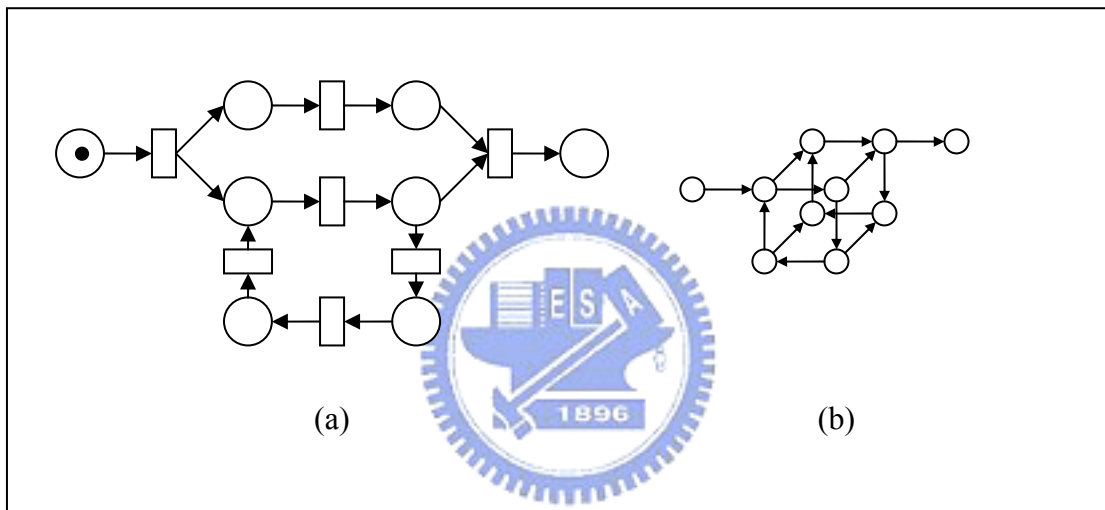


Figure 4.9 (a) A Petri net  $PN$ ; (b) the O-graph of  $PN$

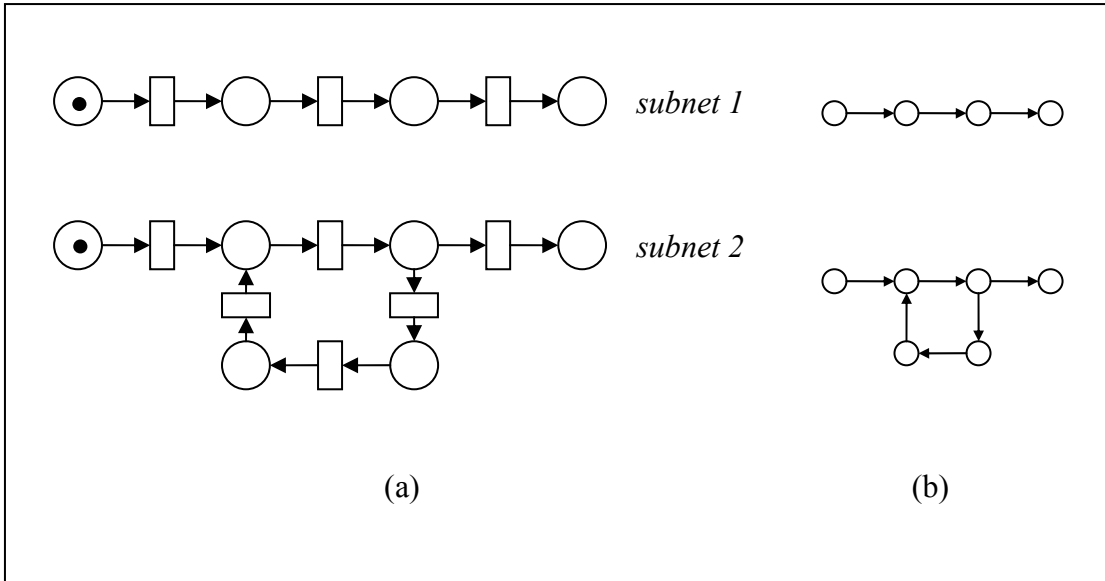


Figure 4.10 (a) Two subnets after dividing  $PN$ ; (b) their O-graphs

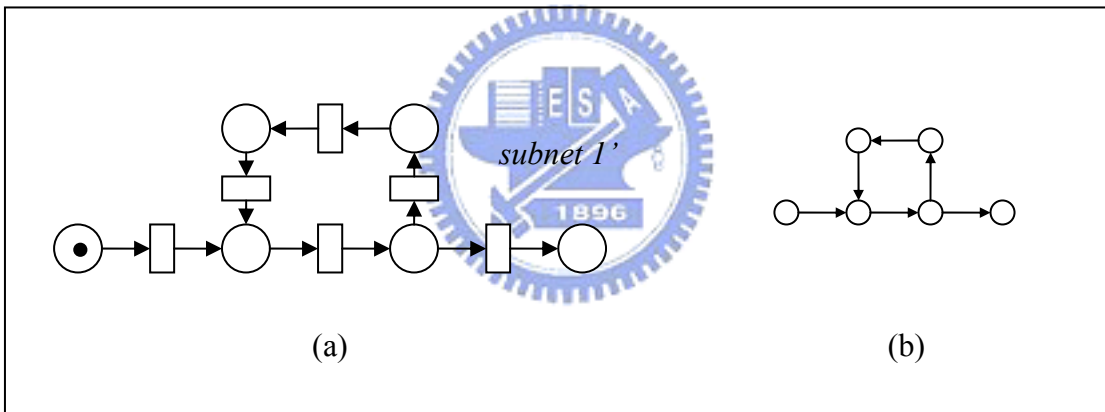


Figure 4.11 (a) The net modified from *subnet 1*; (b) its O-graph

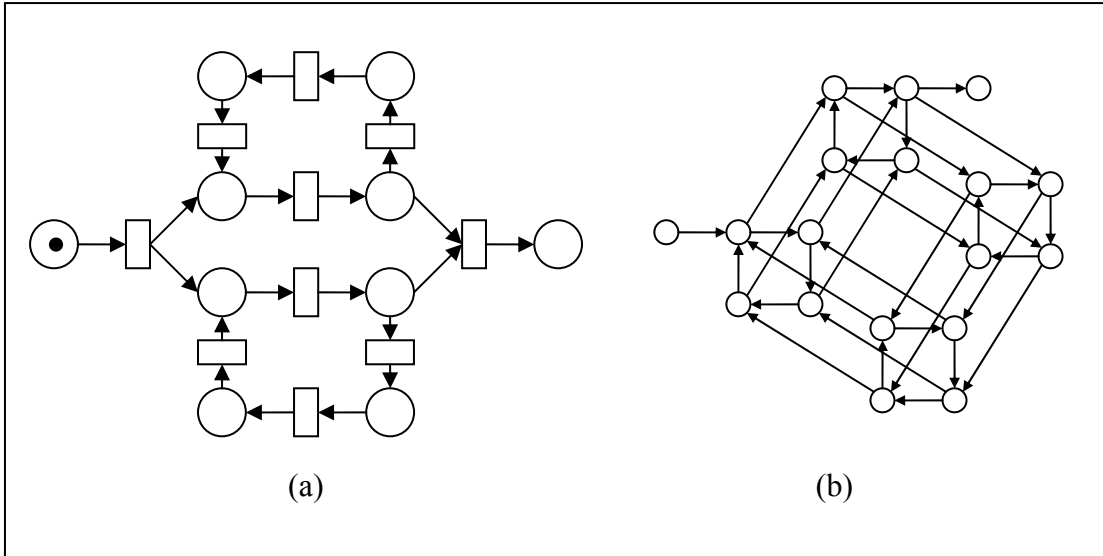


Figure 4.12 (a) The Petri net  $PN'$  merged from *subnets 1'* and *subnets 2*; its O-graph

Since the complexity of generating the final O-graph by the technique for T&P mergence has linear relation with the O-graph's marking number and quadratic relation with the reachable marking number of *subnet 1'*, it is smaller than the complexity of generating the O-graph directly from  $PN'$  which has the quadratic relation with the O-graph's marking number.



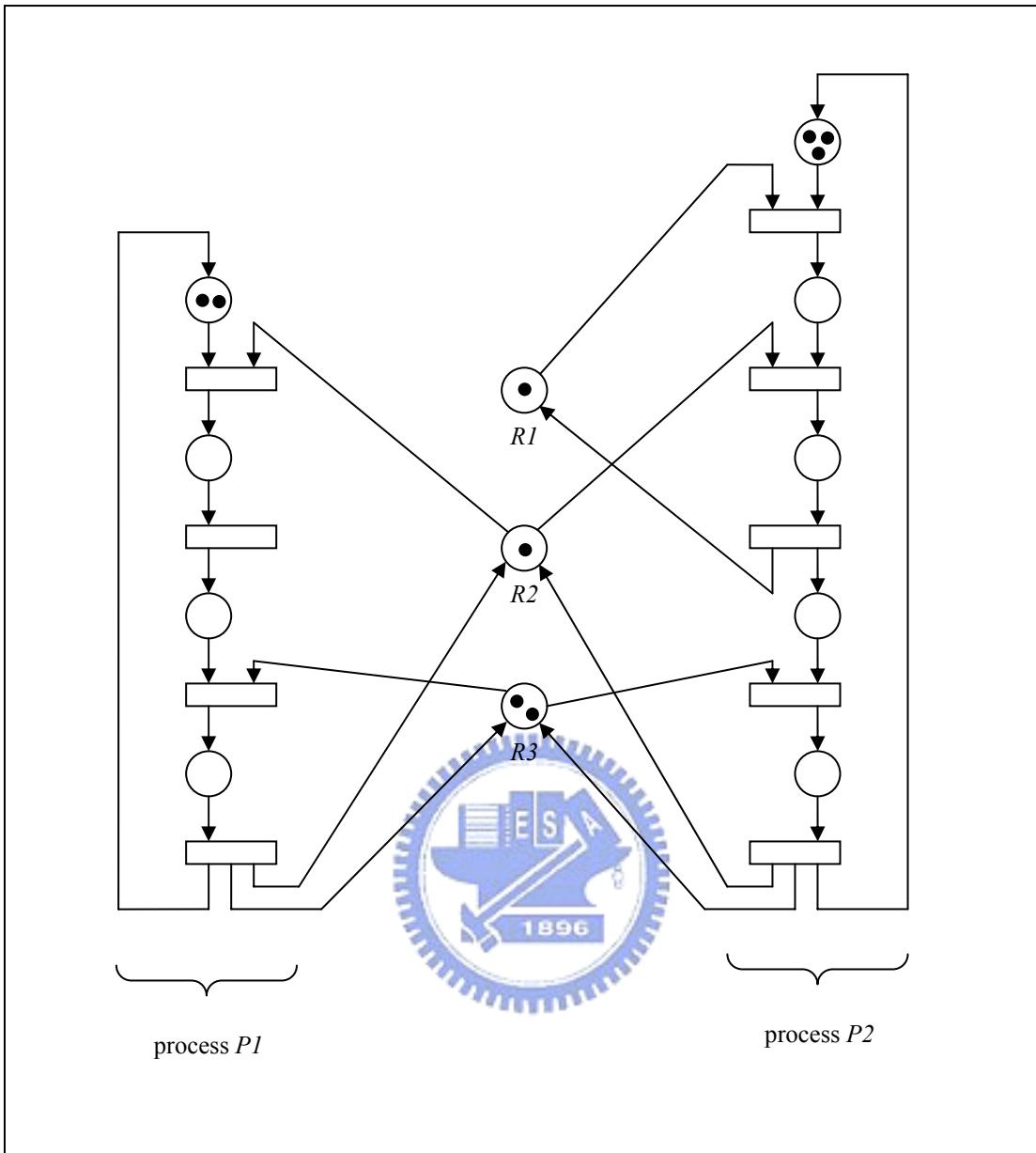


Figure 4.13 A resource allocating system

Figure 4.13 is a more complicated example. This Petri net models a resource allocating system with two concurrent processes ( $P1$  and  $P2$ ) and three resources ( $R1$ ,  $R2$ , and  $R3$ ). By the method, a merging tree using the five nets shown in Figure 4.14 as its leaves and the resource allocating system as its root can be generated at “Divide” activity and the technique for T&P mergence can be applied to.

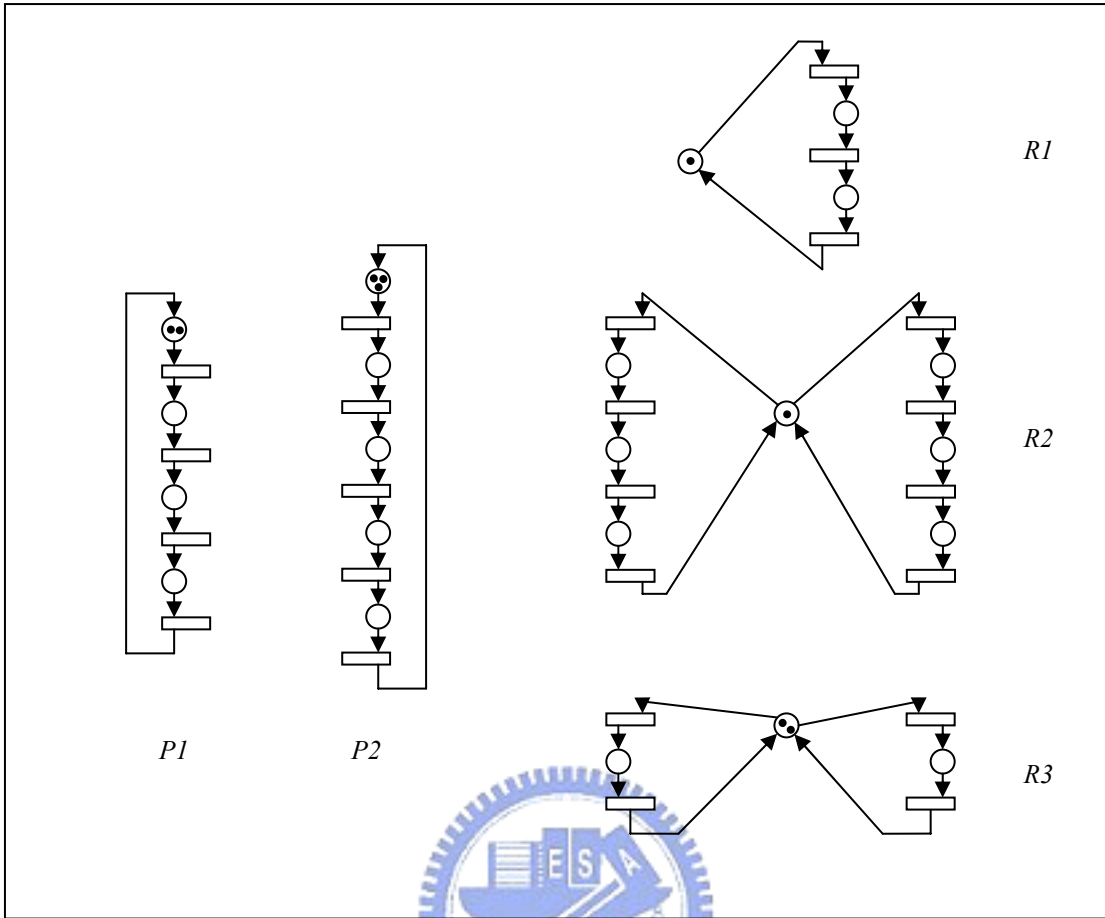


Figure 4.14 Five parts of the resource allocation system in Figure 4.13

## Chapter 5. Example

In this chapter, another example which is more complex than that in Chapter 3 is used to go through Algorithm 3.3. Figure 5.1 shows a Petri net modified from the net in Figure 3.1 (a) (an extra token is added into  $p_1$ ) and its corresponding O-graph. Figure 5.2 shows an example combination of the two nets in Figure 5.1 (a) and Figure 3.1 (a) and the O-graph of combining result. Figure 5.3 shows the net after doing a transition merge according to the merging relation  $\{(t_{1-2}, t_{2-2})\}$  and its O-graph.

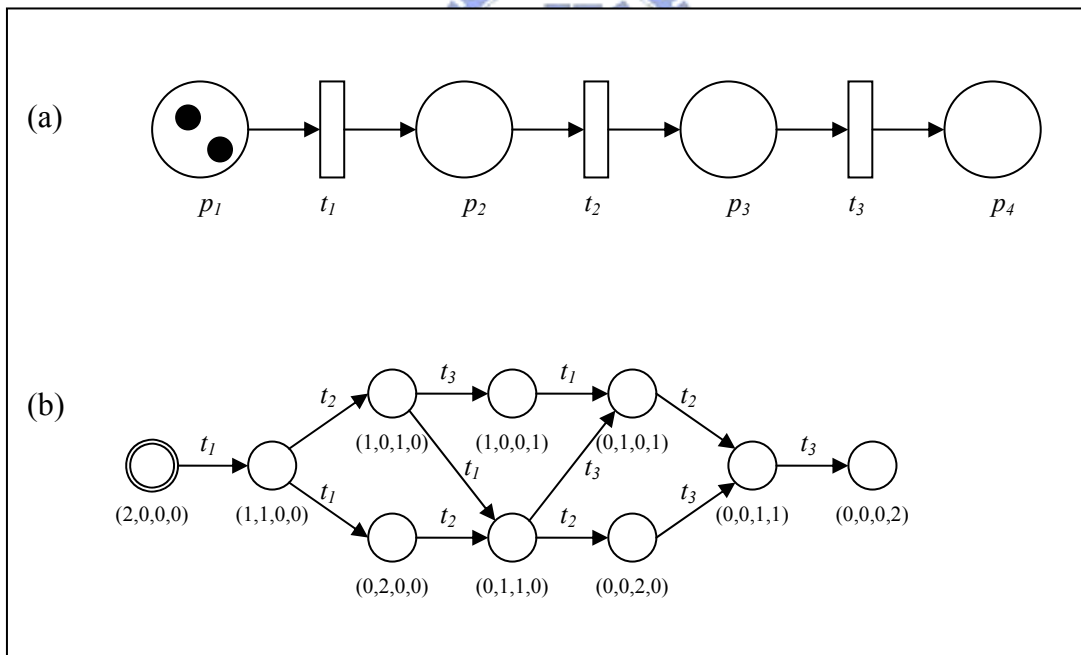


Figure 5.1 (a) An example of Petri net; (b) the O-graph of the net shown in (a)

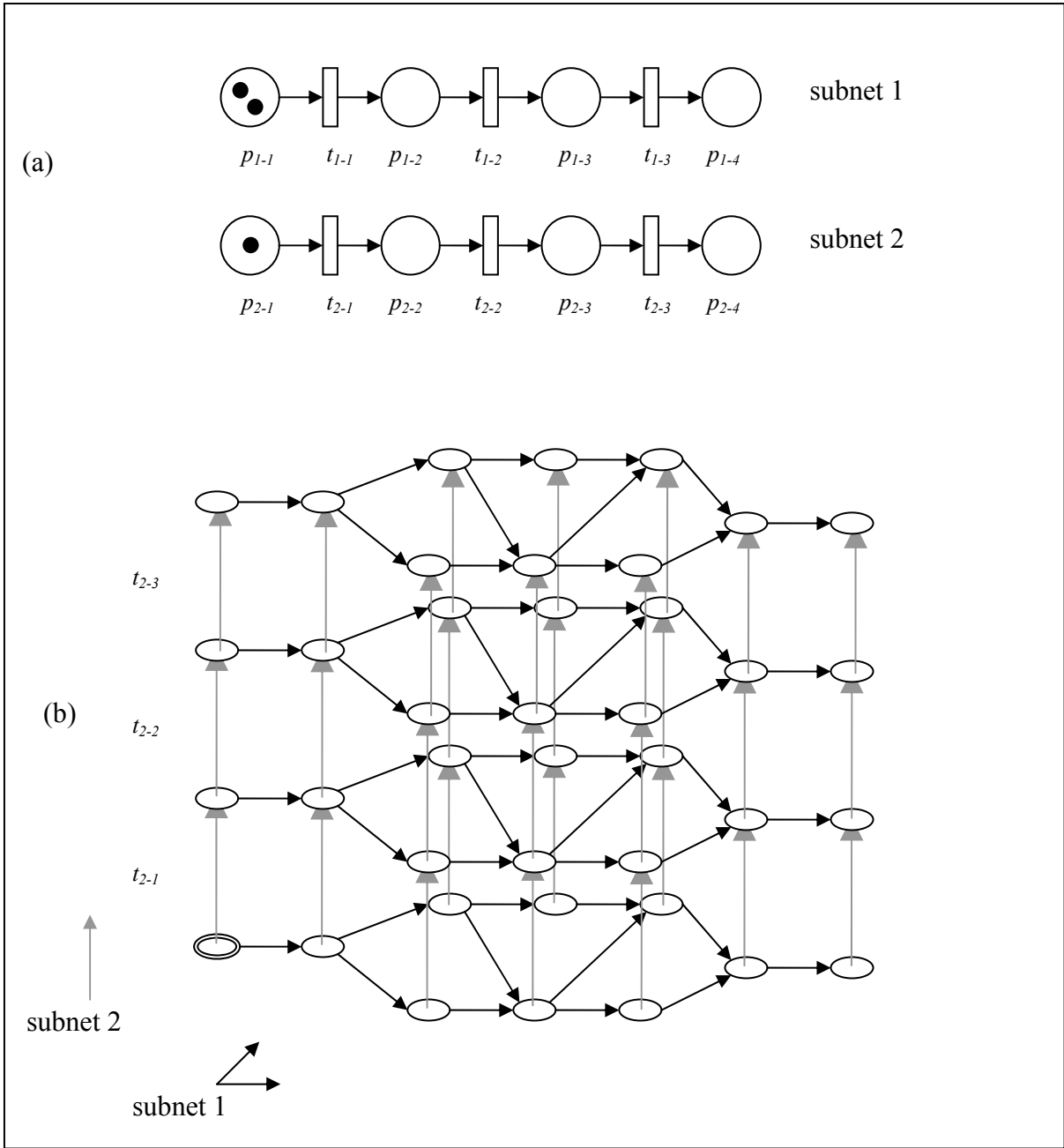


Figure 5.2 (a) A Petri net generated from combining two nets; (b) the O-graph of the net shown in (a)

In Figure 5.2 (b), there are two kinds of arcs, a black arc is corresponding to a firing of a transition in subnet 1 in Figure 5.2 (a) and a gray arc is corresponding to a firing of a transition in subnet 2. After applying Algorithm 3.1 for combining Petri nets, each marking of subnet 1 is merged with each marking of subnet 2 to become a marking in the new O-graph.

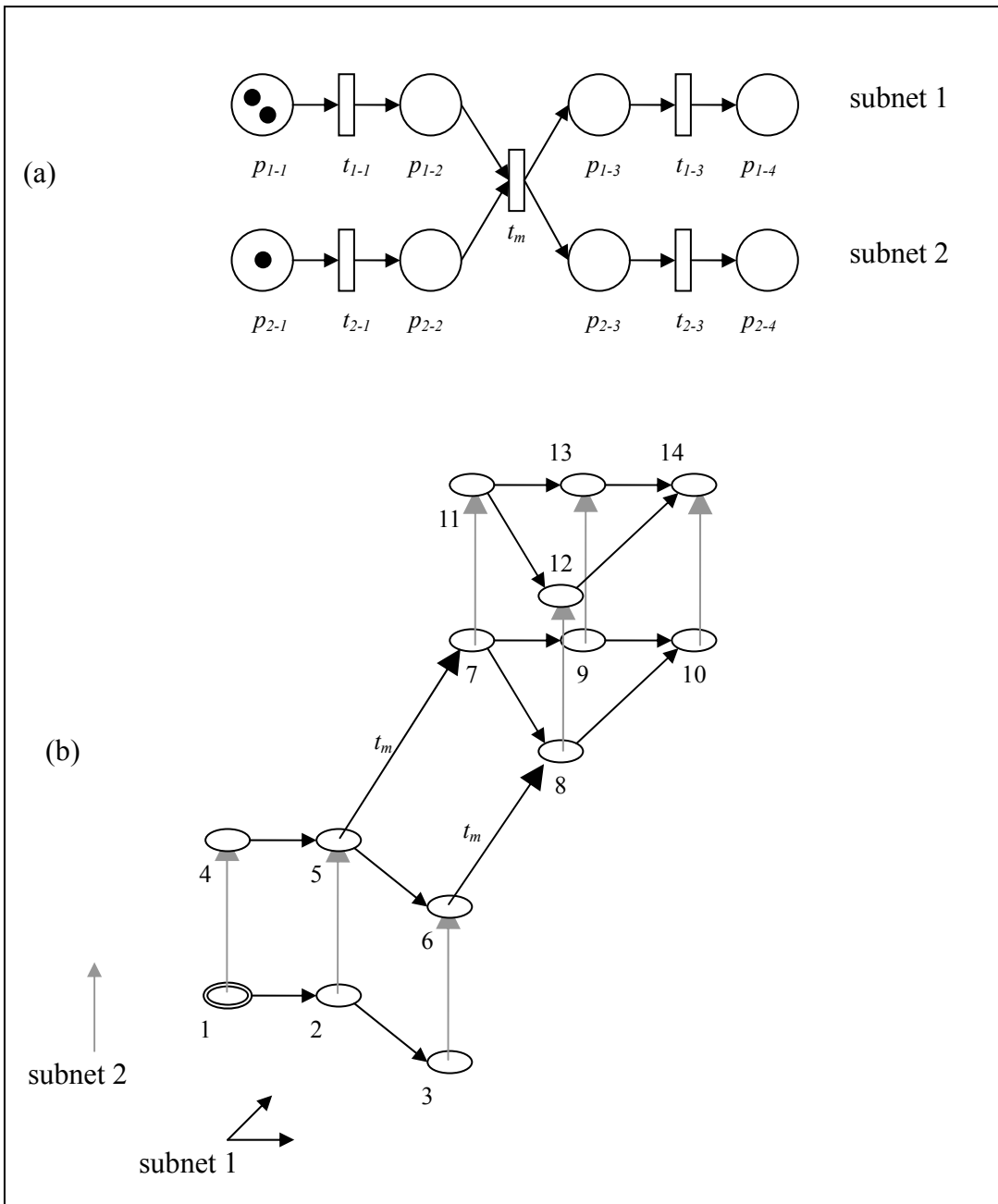


Figure 5.3 (a) A Petri net generated from doing a transition mergence; (b) the O-graph of the net showed in (a)

Figure 5.3 shows the net after merge  $t_{1-2}$  and  $t_{2-2}$  and its O-graph. The nodes in Figure 5.3

(b) are the net's markings:

“1” = (2, 0, 0, 0, 1, 0, 0, 0); “2” = (1, 1, 0, 0, 1, 0, 0, 0);

“3” = (0, 2, 0, 0, 1, 0, 0, 0); “4” = (2, 0, 0, 0, 0, 1, 0, 0);

“5” = (1, 1, 0, 0, 0, 1, 0, 0); “6” = (0, 2, 0, 0, 0, 1, 0, 0);

“7” = (1, 0, 1, 0, 0, 0, 1, 0); “8” = (0, 1, 1, 0, 0, 0, 1, 0);

“9” = (1, 0, 0, 1, 0, 0, 1, 0); “10” = (0, 1, 0, 1, 0, 0, 1, 0);

“11” = (1, 0, 1, 0, 0, 0, 0, 1); “12” = (0, 1, 1, 0, 0, 0, 0, 1);

“13” = (1, 0, 0, 1, 0, 0, 0, 1); “14” = (0, 1, 0, 1, 0, 0, 0, 1)

(the numbers in the arrays specify the numbers of tokens of the places in this order ( $p_{1-1}, p_{1-2}, p_{1-3}, p_{1-4}, p_{2-1}, p_{2-2}, p_{2-3}, p_{2-4}$ )).

The O-graph,  $OG$ , generating steps are shown below:

- (1) Generating the initial marking “1” = (2, 0, 0, 0, 1, 0, 0, 0) refers the two initial marking (2, 0, 0, 0) and (1, 0, 0, 0) of the two subnet. The result is put into  $OG$  and s1pool.

s1pool = {“1”}, s2pool = {}

- (2) Marking “1” = (2, 0, 0, 0, 1, 0, 0, 0) is gotten from s1pool and what arcs starting from (2, 0, 0, 0) in subnet 1 are checked.

- a. The arc ((2, 0, 0, 0),  $t_{1-1}$ , (1, 1, 0, 0)) is found. When  $t_{1-1}$  is not in the merging relation and node (1, 1, 0, 0, 1, 0, 0, 0) is not in  $OG$ , arc (“1” = (2, 0, 0, 0, 1, 0, 0, 0),  $t_{1-1}$ , “2” = (1, 1, 0, 0, 1, 0, 0, 0)) and node “2” are generated and put into  $OG$ . “2” is put into s1pool.

s1pool = {“2”}, s2pool = {}

“1” is put into s2pool.

s1pool = {“2”}, s2pool = {“1”}

- (3) Marking “2” is gotten from s1pool and what arcs starting from (1, 1, 0, 0) in subnet 1 are checked.

- a. The arc  $((1, 1, 0, 0), t_{l-1}, (0, 2, 0, 0))$  is found. When  $t_{l-1}$  is not in the merging relation and node “3” is not in  $OG$ , arc  $(“2”, t_{l-1}, “3”)$  and node “3” are generated and put into  $OG$ . “3” is put into s1pool.

$$s1pool = \{“3”\}, s2pool = \{“1”\}$$

- b. The arc  $((1, 1, 0, 0), t_{l-2}, (1, 0, 1, 0))$  is found. When  $t_{l-2}$  is in the merging relation, “2” is marked with  $t_{l-2}$  (denoted as “2”<sub>m</sub>).

“2”<sub>m</sub> is put into s2pool.

$$s1pool = \{“3”\}, s2pool = \{“1”, “2”_m\}$$

- (4) Marking “3” is gotten from s1pool and what arcs starting from  $(0, 2, 0, 0)$  in subnet 1 are checked.

- a. The arc  $((0, 2, 0, 0), t_{l-2}, (0, 1, 1, 0))$  is found. When  $t_{l-2}$  is in the merging relation, “3” is marked with  $t_{l-2}$ .

“3”<sub>m</sub> is put into s2pool.

$$s1pool = \{\}, s2pool = \{“1”, “2”_m, “3”_m\}$$

- (5) Marking “1” is gotten from s2pool and what arcs starting from  $(1, 0, 0, 0)$  in subnet 2 are checked.

- a. The arc  $((1, 0, 0, 0), t_{2-l}, (0, 1, 0, 0))$  is found. When  $t_{2-l}$  is not in the merging relation and node “4” is not in  $OG$ , arc  $(“1”, t_{2-l}, “4”)$  and node “4” are generated and put into  $OG$ . “4” is put into s1pool.

$$s1pool = \{“4”\}, s2pool = \{“2”_m, “3”_m\}$$

- (6) Marking “2” is gotten from s2pool and what arcs starting from  $(1, 0, 0, 0)$  in subnet 2 are checked.

- a. The arc  $((1, 0, 0, 0), t_{2-l}, (0, 1, 0, 0))$  is found. When  $t_{2-l}$  is not in the merging relation and node “5” is not in  $OG$ , arc  $(“2”, t_{2-l}, “5”)$  and node “5” are generated and put into  $OG$ . “5” is put into s1pool.

$$s1pool = \{“4”, “5”\}, s2pool = \{“3”_m\}$$

(7) Marking “3” is gotten from s2pool and what arcs starting from (1, 0, 0, 0) in subnet 2 are checked.

- a. The arc  $((1, 0, 0, 0), t_{2-1}, (0, 1, 0, 0))$  is found. When  $t_{2-1}$  is not in the merging relation and node “6” is not in  $OG$ , arc (“3”,  $t_{2-1}$ , “6”) and node “6” are generated and put into  $OG$ . Puts “6” into s1pool.

$$s1pool = \{\text{“4”}, \text{“5”}, \text{“6”}\}, s2pool = \{\}$$

(8) Marking “4” is gotten from s1pool and what arcs starting from (2, 0, 0, 0) in subnet 1 are checked.

- a. The arc  $((2, 0, 0, 0), t_{1-1}, (1, 1, 0, 0))$  is found. When  $t_{1-1}$  is not in the merging relation and node “5” is in  $OG$ , arc (“4”,  $t_{1-1}$ , “5”) is generated and put into  $OG$ .

“4” is put into s2pool.

$$s1pool = \{\text{“5”}, \text{“6”}\}, s2pool = \{\text{“4”}\}$$

(9) Marking “5” is gotten from s1pool and what arcs starting from (1, 1, 0, 0) in subnet 1 are checked.

- a. The arc  $((1, 1, 0, 0), t_{1-1}, (0, 2, 0, 0))$  is found. When  $t_{1-1}$  is not in the merging relation and node “6” is in  $OG$ , arc (“5”,  $t_{1-1}$ , “6”) is generated and put into  $OG$ .

- b. The arc  $((1, 1, 0, 0), t_{1-2}, (1, 0, 1, 0))$  is found. When  $t_{1-2}$  is in the merging relation, “5” is marked with  $t_{1-2}$ .

“5”<sub>m</sub> is put into s2pool.

$$s1pool = \{\text{“6”}\}, s2pool = \{\text{“4”}, \text{“5”}_m\}$$

(10) Marking “6” is gotten from s1pool and what arcs starting from (0, 2, 0, 0) in subnet 1 are checked.

- a. The arc  $((0, 2, 0, 0), t_{1-2}, (0, 1, 1, 0))$  is found. When  $t_{1-2}$  is in the merging relation, “6” is marked with  $t_{1-2}$ .

“6”<sub>m</sub> is put into s2pool.

$$s1pool = \{\}, s2pool = \{\text{“4”}, \text{“5”}_m, \text{“6”}_m\}$$



(11) Marking “4” is gotten from s2pool and what arcs starting from (0, 1, 0, 0) in subnet 2 are checked.

- a. The arc  $((0, 1, 0, 0), t_{2-2}, (0, 0, 1, 0))$  is found. When  $t_{2-2}$  is in the merging relation but “4” is not marked with  $t_{1-2}$ , nothing should be done.

$$s1pool = \{\}, s2pool = \{\text{“5”}_m, \text{“6”}_m\}$$

(12) Marking “5” is gotten from s2pool and what arcs starting from (0, 1, 0, 0) in subnet 2 are checked.

- a. The arc  $((0, 1, 0, 0), t_{2-2}, (0, 0, 1, 0))$  is found. When  $t_{2-2}$  is in the merging relation, “5” is marked with  $t_{1-2}$ , and node “7” is not in  $OG$ , arc  $(\text{“5”}, t_m, \text{“7”})$  and node “7” are generated and put into  $OG$ . “7” is put into s1pool.

$$s1pool = \{\text{“7”}\}, s2pool = \{\text{“6”}_m\}$$

(13) Marking “6” is gotten from s2pool and what arcs starting from (0, 1, 0, 0) in subnet 2 are checked.

- a. The arc  $((0, 1, 0, 0), t_{2-2}, (0, 0, 1, 0))$  is found. When  $t_{2-2}$  is in the merging relation, “6” is marked with  $t_{1-2}$ , and node “8” is not in  $OG$ , arc  $(\text{“6”}, t_m, \text{“8”})$  and node “8” are generated and put into  $OG$ . “8” is put into s1pool.

$$s1pool = \{\text{“7”}, \text{“8”}\}, s2pool = \{\}$$

... (until s1pool and s2pool are both empty)

Finally,  $OG$  becomes the O-graph shown in Figure 5.3 (b).

## Chapter 6. Conclusion and Future Works

The major contribution of this thesis is to introduce a method that reduces the O-graph building time when merging some pairs of transitions in two Petri nets. The purpose is achieved by utilizing the O-graph of the two Petri nets directly to generate the new one. O-graph building time reduction contributes to the analyses to Petri net.

Besides merging pair of transitions, this thesis also introduces two extensions: The first one is merging pairs of places when their input and output transitions are merged. The algorithm of transition merge can be used in this case. Second, merging tree is introduced to model merging policy when a sequence of transition merges happens. What applications could use the method provided by us is also discussed here. Two examples about merging two Workflow nets and dividing a Petri net for incremental analysis are used to demonstrate the method.

The future works are listed as follows:

1. State explosion and infinite states are two critical problems of O-graph. There are many methods provided ([2], [3]) for solving these problems. Concerning these two factors to integrate existing methods could increase the ability of our method.
2. The policies of building the merging trees are different in diverse applications. When the structure of a merging tree has big influence to the analysis time, the researches about how to build merging trees for applications is needed.
3. There are some actions/conditions not concerned. For example, places merging is not concerned much. They might be worth while for further study.

4. There are many kinds of high-level Petri nets (For example, coloured Petri nets introduced in [1], [2], and [4] and timed Petri nets introduced in [4]). One of our future works is adapting our method to handle these high-level Petri nets.



## Reference

- [1] Kurt Jensen, “Coloured Petri Nets: Basic Concepts,” Springer, 1992.
- [2] Kurt Jensen, “Coloured Petri Nets: Analysis Methods,” Springer, 1995.
- [3] W. Reisig and G. Rozenberg, “Lectures on Petri Nets I: Basic Models,” Springer, 1998.
- [4] W. Reisig and G. Rozenberg, “Lectures on Petri Nets II: Applications,” Springer, 1998.
- [5] Jörg Desel and Gabriel Juhás, “What Is a Petri Net,” Unifying Petri Nets, Lecture Notes in Computer Science, pp. 1-25, Springer, 2001.
- [6] Piotr Chrzastowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O’Dell, and Adi Susanto, “A Top-Down Petri Net-Based Approach for Dynamic Workflow Modeling,” International Conference on Business Process Management, Springer-Verlag Berlin Heidelberg 2003.
- [7] C.A. Petri, “Kommunikation mit Automaten,” PhD thesis, University of Bonn, Bonn, Germany, 1962.
- [8] T. Murata, “Petri Nets: Properties, Analysis and Applications,” Proceedings of the IEEE, vol. 77, issue 4, pp. 541 – 580, April 1989.
- [9] W.M.P. van der Aalst, “Verification of Workflow Nets,” Application and Theory of Petri Nets, volume 1248 of Lecture Notes in Computer Science, pages 407–426, Springer-Verlag, Berlin, 1997.
- [10] W.M.P. van der Aalst, “The Application of Petri Nets to Workflow Management,” The Journal of Circuits, Systems and Computers, vol. 8, no. 1, pp. 21–66, 1998.
- [11] W.M.P. van der Aalst and A.H.M ter Hofstede, “Verification of workflow task structures: A petri-net-based approach,” Information System, Vol. 25, No. 1, pp. 43-69, 2000.
- [12] Julia Padberg, Maike Gajewsky, and Kathrin Hoffmann, “Incremental Development of

Safety Properties in Petri Net Transformations,” Lecture Notes in Computer Science, pp. 410-425, Springer-Verlag, 2000.

- [13] J. Padberg, M. Gajewsky, and C. Ermel, “Rule-based refinement of high-level nets preserving safety properties,” *Science of Computer Programming*, vol. 40, issue 1, pp. 97-118, May 2001.
- [14] Julia Padberg, “Categorical Approach to Horizontal Structuring and Refinement of High-Level Replacement Systems,” *Applied Categorical Structures*, vol. 7, number 4, Springer Netherlands, 1999.
- [15] J. Padberg, H. Ehrig, and L. Ribeiro, “Algebraic High-Level Net Transformation Systems,” *Mathematical Structures in Computer Science* 5, pp. 217–256, 1995.
- [16] P. Molinaro, D. Roux, and O. Delfieu, “Improving the calculus of the marking graph of Petri net with BDD like structure,” *IEEE International Conference on Systems, Man and Cybernetics*, vol. 1, pp. 43-48, Oct. 2002.
- [17] P. Molinaro and M. Magnin, “Markg”, Available at <http://markg.rts-software.org>, 2006.
- [18] Sheldon B. Akers, “Binary Decision Diagrams,” *IEEE Transactions on Computers*, vol. C-27, issue 6, pp. 509-516, Jun. 1978.
- [19] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, “Introduction to Algorithms, Second Edition, Chapter 22. Elementary Graph Algorithms,” The MIT Press, 2001.

## Appendix

The content of the box shown below contains the additional data structures about two pools used in the following construction-based algorithm of transition mergence.

```
Structure Subnet1Pool {
 Queue(Vertex) ums1; // a queue with the element type "Vertex"
}

Structure Subnet2Pool {
 Queue(UnsolvedMarkingS2) ums2; // a queue with the element type
 "UnsolvedMarkingS2"
}

Structure UnsolvedMarkingS2 {
 Vertex v;
 Arc[] marc; // the arcs that must be marked
}
```

Algorithm A is the detail of Algorithm 3.3, which is the construction-based algorithm for transition mergence.

**Algorithm A: PNTransitionMergenceV2(PetriNet  $PN1$ , PetriNet  $PN2$ , MergingRelation  $MR$ )**

```

1 // Creat a new empty Petri Net then add $PN1$ and $PN2$ in it.
2 PetriNet PN = new PetriNet;
3
4 // merge the Petri Net part
5 $PN.p$ = clone($PN1.p$ + $PN2.p$);
6 $PN.t$ = clone($PN1.t$ + $PN2.t$);
7 $PN.f$ = clone($PN1.f$ + $PN2.f$);
8 Transition[] mt = new Transition[$MR.tp.size$];
9 FOR i from 0 to ($MR.tp.size - 1$)
10 // make new transition in PN
11 $mt[i]$ = mergeTransition(PN , $MR.tp[i]$);
12 END
13 // merge the occurrence graph part
14 int newID = 0;
15 Subnet1Pool $s1pool$ = new Subnet1Pool;
16 Subnet2Pool $s2pool$ = new Subnet2Pool;
17 Vertex vx = $PN1.OG.pointerTable1D[0]$;
18 Vertex vy = $PN2.OG.pointerTable1D[0]$;
19 Vertex v = new Vertex;
20 $v.m$ = $vx.m$ + $vy.m$;
21 $v.tmpID1$ = $vx.ID$;
22 $v.tmpID2$ = $vy.ID$;
23 $v.ID$ = newID;
24 $PN.OG.pointerTableD1[v.ID]$ = v ;
25 newID++;
26 $PN.OG.v += v$;
27 $PN.OG.pointerTableD2[v.tmpID1][v.tmpID2]$ = v ;
28 add($s1pool.ums1$, v);
29

```

(to be continued)

```

30 WHILE (s1pool.ums1 ≠ ∅ or s2pool.ums2 ≠ ∅)
31 WHILE (s1pool ≠ ∅)
32 v = remove(s1pool.umd1);
33 UnsolvedMarkingS2 ums2 = new UnsolvedMarkingS2(v);
34 FOR ALL (originalArc ∈ PN1.OG.pointerTable[v.tmpID1].arcOut)
35 boolean isMT = false;
36 FOR i from 0 to MR.tp.size - 1
37 IF originalArc.t == TM.tp[i].t1
38 ums2.marc[i] = originalArc;
39 isMT = true;
40 END
41 END
42 IF isMT != true
43 Arc arc = new Arc;
44 arc.vStart = v;
45 v.arcOut += arc;
46 IF PN.OG.pointerTable[originalArc.vEnd.ID][v.tmpID2] == null
47 Vertex newV = new Vertex;
48 newV.m = originalArc.vEnd.m + PN2.OG.pointerTable[v.tmpID2].m;
49 newV.tmpID1 = originalArc.vEnd.ID;
50 newV.tmpID2 = v.tmpID2;
51 newV.ID = newID;
52 PN.OG.pointerTable1D[newV.ID] = newV;
53 newID++;
54 PN.OG.v += newV;
55 PN.OG.pointerTable2D[newV.tmpID1][newV.tmpID2] = newV;
56 add(s1pool.ums1, v);
57 ELSE
58 Vertex newV = PN.OG.pointerTable[originalArc.vEnd.ID][v.tmpID2]
59 END
60 arc.vEnd = newV;
61 newV.arcIn += arc;
62 arc.t = originalArc.t;
63 PN.OG.a += arc;
64 END
65 END
66 add(s2pool.umd2, ums2);
67 END
68

```

(to be continued)



```

69 WHILE (s2pool ≠ ∅)
70 ums2 = remove(s2pool.ums2);
71 v = ums2.v;
72 FOR ALL (originalArc ∈ PN.OG.pointerTable[v.tmpID2].arcOut)
73 boolean isMT = false;
74 FOR i from 0 to MR.tp.size - 1
75 IF originalArc.t == MR.tp[i].t2;
76 isMT = true;
77 IF ums2.marc[i] != null
78 Arc arc = new Arc;
79 arc.vStart = v;
80 v.arcOut += arc;
81 IF PN.OG.pointerTable[ums2.marc[i].vEnd.ID][originalArc.vEnd.ID] == null
82 Vertex newV = new Vertex;
83 newV.m = ums2.marc[i].vEnd.m + originalArc.vEnd.m;
84 newV.tmpID1 = ums2.marc[i].vEnd.ID;
85 newV.tmpID2 = originalArc.vEnd.ID;
86 newV.ID = newID;
87 PN.OG.pointerTable1D[newV.ID] = newV;
88 newID++;
89 PN.OG.v += newV;
90 PN.OG.pointerTable2D[newV.tmpID1][newV.tmpID2] = newV;
91 add(s1pool.ums1, v);
92 ELSE
93 Vertex newV =
94 PN.OG.pointerTable[ums2.marc[i].vEnd.ID][originalArc.vEnd.ID];
95 END
96 Arc.vEnd = newV;
97 newV.arcIn += arc;
98 arc.t = mt[i];
99 PN.OG.a += arc;
100 END
101 END
102 END

```

(to be continued)

```

103 IF isMT != true;
104 Arc arc = new Arc;
105 arc.vStart = v;
106 v.arcOut += arc;
107 IF PN.OG.pointerTable[v.tmpID1][originalArc.vEnd.ID] == null
108 Vertex newV = new Vertex;
109 newV.m = PN.OG.pointerTable[v.tmpID1].m + originalArc.vEnd.m;
110 newV.tmpID1 = v.tmpID1;
111 newV.tmpID2 = originalArc.vEnd.ID;
112 newV.ID = newID;
113 PN.OG.pointerTable1D[newV.ID] = newV;
114 newID++;
115 PN.OG.v += newV;
116 PN.OG.pointerTable2D[newV.tmpID1][newV.tmpID2] = newV;
117 add(s1pool.ums1, v);
118 ELSE
119 Vertex newV = PN.OG.pointerTable[v.tmpID1][originalArc.vEnd.ID]
120 END
121 arc.vEnd = newV;
122 newV.arcIn += arc;
123 arc.t = originalArc.t;
124 PN.OG.a += arc;
125 END
126 END
127 END
128 END
129
130 RETURN PN;

```