國立交通大學

資訊科學與工程研究所

碩 士 論 文

當編輯派翠網時有效率維護發生圖之方法

An Efficient Method to Maintain the O-graph when Editing a Petri Net

研 究 生：韓兆庭

指導教授：王豐堅　教授

中 華 民 國　九 十 七　年 八 月

當編輯派翠網時有效率維護發生圖之方法

An Efficient Method to Maintain the O-graph when Editing a Petri Net

研 究 生：韓兆庭　　Student：Zhao-Ting Han

指導教授：王豐堅　　Advisor：Feng-Jian Wang

國 立 交 通 大 學

資訊科學與工程研究所

碩 士 論 文

A Thesis

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

August 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年八月

當編輯派翠網時有效率維護發生圖之方法

研究生: 韓兆庭　　　　指導教授: 王豐堅 博士

國 立 交 通 大 學
資訊科學與工程研究所
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：當編輯派翠網時利用維護原本的發生圖來縮短建立新發生圖的時間。根據派翠網的語意，本文討論到的編輯動作包含：新增、刪除、合併各項派翠網元件。為了保持派翠網語意上的正確性，每個編輯的動作可能是一或一組點或線的自動編輯動作。相對的，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。

關鍵字： 派翠網、發生圖、狀態圖、遞增分析、工作流程

Maintaining the O-graph when Editing a Petri Net

Student: Zhao-Ting Han Advisor: Dr. Feng-Jian Wang

Institute of Computer Science and Engineering

National Chiao Tung University

1001 Ta Hsueh Road, Hsinchu, Taiwan, ROC

Abstract

Petri Net can model many kinds of systems. An occurrence graph is the state diagram of a Petri net. It can show useful properties such as reachability, boundness, home, liveness, and fairness. This thesis presents a technique that reduces the occurrence graph building time by maintaining current, instead of constructing new O-graph of a Petri net when an edit action is done. The maintenance is based on three groups of edit actions: (1) addition of a(n) node, arc, or token, (2) deletion of a(n) node, arc, or token, (3) mergence of places or transitions where each of them represents a or a sequence of addition/deletion of place, transition, and/or arc. The maintenance algorithm, proof, and time complexity for each edit action are discussed respectively.

Keywords: Petri net, 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、留忠賢博士以及陳俊穎博士，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，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

List of Figures

Chapter 1. Introduction

Petri Net can model many kinds of systems ([12], [13], [14]) and can be represented by graphical and mathematical ways. Obviously, the graphical representations can help designers edit Petri nets. Since the mathematical representations define a Petri net clearly, there are many analysis methods developed for Petri Net ([11], [13], [14]).

An occurrence graph (O-graph) is the state space diagram of a Petri net ([14]). 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. Many analysis tools for Petri nets are based on their behaviors contained in O-graphs ([2], [11]).

Nowadays, the support of a Petri net design is an environment, not a single editor only. In the environment, some tools associated with the editor might work per edit action. These tools are called 'incremental analysis', since they work based on environment, edited datum, and graphs, but not from the batch. Incremental analysis could give designers some warnings which assist designers right after an edit action. Such a tool might keep its corresponding information for edited datum and analysis time might be reduced when designers continue to edit Petri nets.

Because editing a Petri net includes the nodes and arcs edit actions, this thesis conclude three groups of edit actions: (1) addition of a(n) node, arc, or token, (2) deletion of a(n) node, arc, or token, (3) mergence of places or transitions. To ensure the semantic correctness of a Petri net, every transition has both input and output places and each arc connects a place and a transition in the net. Some edit actions have a batch of simple actions which are done by our editor automatically.

This thesis presents a technique to reduce the O-graph construction time incrementally. Thus, the technique proposed maintains the O-graph instead of creating a new O-graph. In other word, it modifies current O-graph according to each edit action. Most of the differences between current Petri net and the one after an edit action are small and can be found by rules. When a designer edits a Petri net, its O-graph can show useful properties such as reachability, boundness, home, liveness, and fairness. These analysis results can assist designers in speed and correctly when editing Petri nets.

The remainder of this thesis is organized as follows. Chapter 2 presents the motivation and introduces Petri nets, O-graphs, and related works. Three group of basic edit actions are defined in Chapter 3. Chapter 4 introduces the definitions and algorithms of maintaining the O-graph for item insertion. Chapter 5 introduces the definitions and algorithms of maintaining the O-graph for item deletion and mergence. A comparison between our approach and the original algorithm of building O-graph from a Petri net is given in Chapter 6. Chapter 7 concludes the thesis and indicates some future works.

Chapter 2. Background

2.1 Petri Net

Petri Net model is originated from the early work of Carl Adam Petri ([15]). Most readers refer to [14] when applying Petri nets. Petri net can be used to model many kinds of system and analyze with associated techniques.

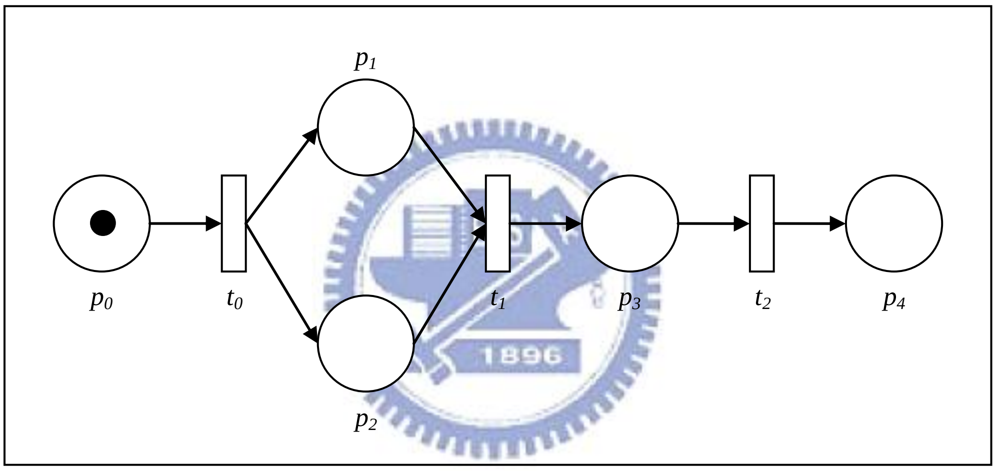


Figure 2.1 An example of a Petri net

A Petri net (also called "net" in this thesis for short) is a directed graph with two kinds of nodes, named place and transition. There are no arcs connecting two places or two transitions. A Petri net is also equipped with an initial marking, i.e. initially putting tokens in some places. A marking is a current state of the net, as will be shown in Definition 2.2. Figure 2.1 is a Petri net example. A circle means a place, a rectangle means a transition, and a dot means a token. If the initial marking of the Petri net puts a token in place p_0, Figure 2.1 shows the initial state of the Petri net.

Definition 2.1 (Petri net)

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

Definition 2.1 is the algebraic definition of a Petri net. While this thesis presents a technique about Petri Net, some precise algebraic definitions about Petri Net must be defined.

Definition 2.2 (marking)

- A *marking* of a set of place P is a mapping $\boldsymbol{m}: P \rightarrow \mathbb{N}$ where $\mathbb{N} = \{0, 1, 2, \ldots\}$.
- 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 (pre- /post- set)

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

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

Definition 2.3 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.

A transition t is said to be enabled at m_0 if for all $p \in {}^{\bullet}t$, $m_0(p) \geqq 1$. A transition may fire if it is enabled. The new marking *m'* is obtained by removing one token from each of its input places and by putting one token to each of its output places. This process is denoted by $m\ [t > m'$. Extension to **firing sequences** is denoted by $m\ [\sigma > m'$ where $\boldsymbol{\sigma}$ is a sequence of transitions that brings *m* to *m'*.

Definition 2.4 (incidence matrix)
The *incidence matrix* $\boldsymbol{C} = [c_{ij}]$ is such that

$$c_{ij} = \begin{cases} 1 & \text{if } t_j \in {}^{\bullet}p_i \setminus p_i^{\bullet} \\ -1 & \text{if } t_j \in p_i^{\bullet} \setminus {}^{\bullet}p_i \\ 0 & \text{otherwise} \end{cases}$$

Definition 2.5 (P-characteristic vector)
The *P-characteristic vector* is a vector of dimension |P|, where |P| is the number of places. In a *P-characteristic vector PCV.*p_i, $\forall\, p_i \in P$: the dimension for p_i is set to be 1, and the rest are 0.

For example, $PCV.p_1$=(0,1,0,0,0) is a P-characteristic vector of the Petri net in Figure 2.1, where each dimension represents a place and the number in the dimension for p_1 is 1.

Definition 2.6 (T-characteristic vector)
The *T-characteristic vector* is a vector of dimension |T|, where |T| is the number of places. In a *T-characteristic vector TCV.*t_i, $\forall\, t_i \in T$: the dimension for t_i is set to be 1, and the rest are 0.

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'*. The sentence "*m* marks all places in ${}^{\bullet}t$" indicates that the marking *m* puts at least one token on each place in ${}^{\bullet}t$. $m' = m + C * e_t$

shows the behavior of Petri nets while an occurrence of a transition (t) can transform from one marking (m) to another (m') 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.

For any m such that $m_0 [\sigma > m$, $m = m_0 + C*\vec{\sigma}$ where $\vec{\sigma}$, called the **firing count vector**, is a vector whose *i-th* entry denotes the number of occurrences of t_i in σ. If there is a marking m' that $m [t > m'$, the firing count vector at m' is $\vec{\sigma}' = \vec{\sigma} + e_t$.

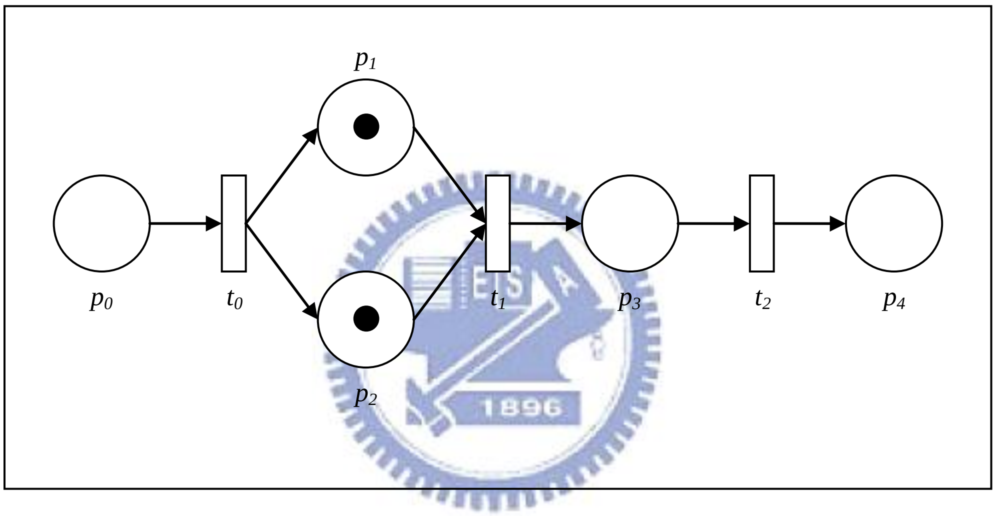


Figure2.2 The marking after firing t_1

For example, in the net showed in Figure 2.1, if t_0 is fired, the marking of this net becomes Figure 2.2 – a token is removed from p_0 and two tokens is added on p_1 and p_2 separately. The marking of the net showed in Figure2.2 can be represented as $m' = (0, 1, 1, 0, 0)$, while m is the marking of Figure 2.1.

$$\begin{array}{ccccccc} m & & C & & TCV.t_0 & & m' \\ \begin{bmatrix}1\\0\\0\\0\\0\end{bmatrix} & + & \begin{bmatrix}-1 & 0 & 0\\1 & -1 & 0\\1 & -1 & 0\\0 & 1 & -1\\0 & 0 & 1\end{bmatrix} & * & \begin{bmatrix}1\\0\\0\end{bmatrix} & = & \begin{bmatrix}0\\1\\1\\0\\0\end{bmatrix} \end{array}$$

2.2 Editors for Petri Net

There are various Petri Net editors. In this thesis, a typical editor, called “Platform Independent Petri Net Editor” PIPE2 [16], is a graphical tool of open source with Java. PIPE2 contains the following analysis modules: Invariant, Simulation, and State Space, etc. Invariant Analysis Module calculates T- and P-invariants and the P-invariant equations. Simulation Analysis shows the average number of tokens in each place. In State Space Analysis Module, the result of the analysis is the determination of the boundness and the safeness of the current Petri net.

2.3 The Incremental Analysis Based on Occurrence Graph

2.3.1 Occurrence Graph

An occurrence graph (O-graph) can also be called a reachability graph. 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 2.7 (arc-labeled directed graph)

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 2.8 (occurrence graph)

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;
- $A = \{(m_i, t, m_j) \in M \times T \times M \mid m_j$ is the marking after firing t on $m_i\}$.

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.

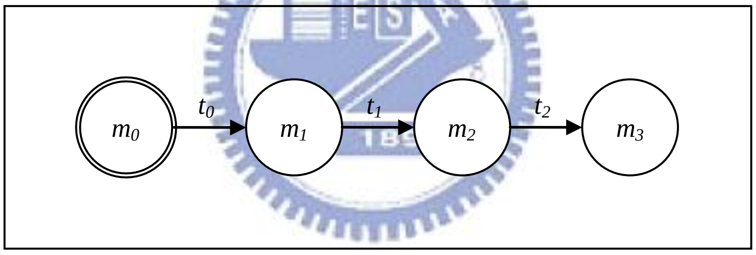


Figure 2.3 The O-graph of the Petri net in Figure 2.1

Figure 2.3 shows the O-graph of the net showed in Figure 2.1. Each circle denoted by a identification 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: $m_0 \rightarrow (1, 0, 0, 0, 0)$, $m_1 \rightarrow (0, 1, 1, 0, 0)$, $m_2 \rightarrow (0, 0, 0, 1, 0)$, $m_3 \rightarrow (0, 0, 0, 0, 1)$.

Some of the properties that can be studied by using the O-graph are the following: reachability, boundness, home, liveness, and fairness. These properties could give designers some warnings. State space analysis is a popular formal reasoning technique. State-based formal reasoning techniques [11] commonly involve examining every possible state of a system. Such techniques are automatic, can be applied by less trained personnel, and can be used for analysis and error detection as well as verification.

2.3.2 Related Works

Piotr Chrząstowski-Wachtel [6] proposed five basic refinement rules: sequential place split, sequential transition split, OR-split, AND-split, and Loop, to refine a kind of workflow nets, named WF-net [10] but a Petri net, using a top-down manner. If the design of a WF-net starts from a single place with no transitions and above five refinement rules are applied only, the resulting WF-net is sound [6] [10]. However, the refinement rules are not enough to construct all kinds of WF-nets and he presented two non-refinement rules, “communication” and “synchronization”, in addition to modifying WF-nets without losing soundness property.

Glenn Lewis and Charles Lakos [4] had certain points of contact with the approach of Christensen and Petrucci (Modular Analysis of Petri Nets, defined in [3]), in attempting to alleviate state space explosion by utilizing the incremental structure based on Coloured Petri Net (CPN) models. Christensen and Petrucci illustrated their techniques by means of modular Place/Transitions nets, and the results are called modular PT-nets. For state spaces, they showed that it is possible to decide behavioural properties of the modular PT-net from state spaces of the individual modules plus a *synchronization graph*, without unfolding to the ordinary state space. Designers commonly start with an abstract model of the system (and possibly verify certain properties of the abstraction) and then progressively refine that model till sufficient details are included. There are three forms of refinement which is called system morphism. Type refinement involves incorporating additional information in the tokens and firing modes. Subnet refinement involves augmenting a subnet with additional places, transitions, and arcs. Node refinement replaces a place (transition) by a place (transition) bordered subnet. Nevertheless, the subnet and bordered subnet to be added have many constraints [4].

2.3.3 Previous Research

Yo-Han Lin [8] presented a technique to reduce the construction time of the O-graph which belongs to a refined net modified by an editing action, called transition mergence. By merging parts of transitions in two abstract nets we can construct a refined net. Incremental state space construction in this article is implemented by reusing the O-graphs of two abstract nets instead of generating from the refined net. The transition mergence only allows two Petri nets. Additionally, he introduced the policy of merging more than two nets. The deletion-based transition mergence algorithm combines two O-graphs of the original nets and then deletes the useless markings and arcs. The construction-based mergence algorithm examines two O-graphs of the original nets and then constructs the useful markings and arcs.

2.4 Motivation

In general, Petri Net can be applied to model workflows. As workflows are more and more complex, the capability to incrementally analyze Petri nets that designers are editing is an essential requirement for most Petri Net editor software. When designers use the editor software which has the ability of incremental analysis, the errors may be found and corrected after each modification actions such as adding arcs, places, and transitions.

Current analysis techniques including the approaches in section 2.3.2 pay little attention to incremental analysis for various Petri net modifications. However, such analysis based on

O-graphs is very important since Petri Net designers commonly start with an abstract model of the system. Each of above modification rules is restricted for some specific properties. The properties after the integration of these actions are more complicated in general. Still, some properties can not be preserved after several modifications [4] [6].

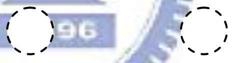
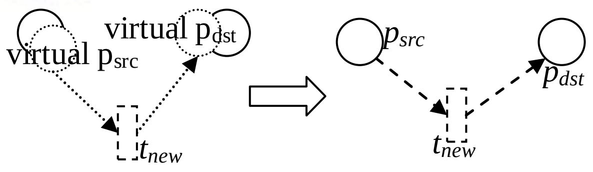
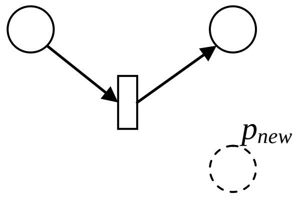
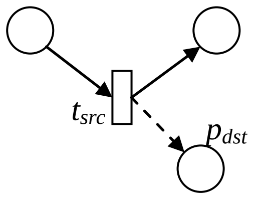
In PIPE2 [16], there are eight modes: Place, Transition, Timed Transition, Arc, Select, Add Token, and Delete Token Mode, for Petri net edition. PIPE2 has editing Petri Net tools for designers and presents several modules for detecting the properties of Petri nets. Nevertheless, there is no incremental analysis presented.

As in section 2.3.3, a transition mergence to refine Petri nets was proposed and the corresponding incremental analysis method is provided. However, an editor which only provides transition mergence editing action is not complete and intuitive for designers.

Chapter 3. The Changes of Occurrence Graph for a Petri Net Edition

3.1 The Definition of Basic Actions

There exist many editors for Petri nets nowadays. For the discussion of incremental analysis based on occurrence graph, here we define a general editor containing three groups of edit actions for designers to draw Petri nets. The first group is used for the addition of a(n) node, arc, or token. The second group is used for the deletion of a(n) node, arc, or token. The third group is a set of specific actions for the mergence of places or transitions. When a Petri net is created, there are two places constructed by default.

initial graph	
$E = addTrans(PN, t_{new}, p_{src}, p_{dst})$	(a)  virtual p_{dst} virtual p_{src} t_{new} p_{src} p_{dst} t_{new}
$E = addPlace(PN, p_{new})$	(b) 
$E = addTtoParc(PN, t_{src}, p_{dst})$	(c) 

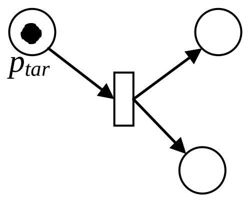
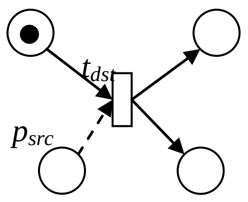
$E = addToken(PN, p_{tar})$	(d) 
$E = addPtoTarc(PN, p_{src}, t_{dst})$	(e) 

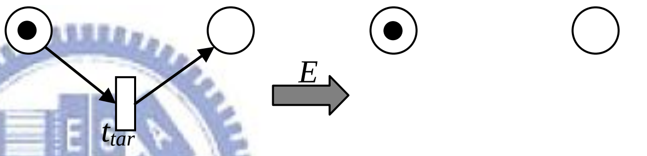
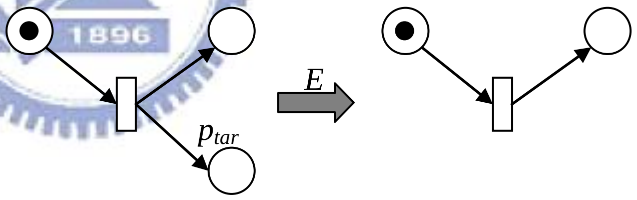
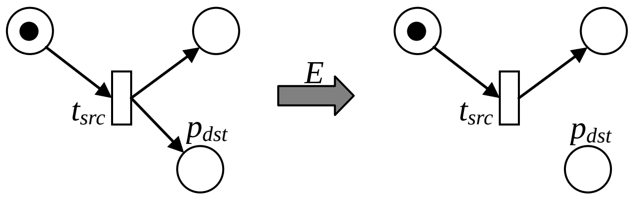
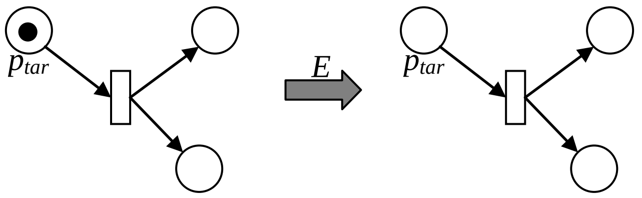
Figure 3.1 Examples of the basic edit actions in the first group

This thesis is mainly concerned with the actions of the first group. Figure 3.1 shows a set of editing actions, where (b), (c), (d), and (e) show the addition (insertion) of a place, TtoP arc, token, and PtoT arc respectively. In order to ensure that every transition has at least one input place and one output place, our editor provides a batch of actions for adding a transition. When a designer adds one transition in the graph, "virtual p_{src}" and "virtual p_{dst}" will appear concurrently. The designer needs to drag them individually to a source place and a destination place and then the batch of actions are accomplished.

Basic editing action	Pre-condition
$E = addTransition(PN, t_{new})$	$t_{new} \notin T$
$E = addPtoTarc(PN, p_{src}, t_{dst})$	$p_{src} \in P$ $t_{dst} \in T$
$E = addTtoParc(PN, t_{src}, p_{dst})$	$t_{src} \in T$ $p_{dst} \in P$
$E = addTrans(PN, p_{src}, t_{new}, p_{dst})\{$ $addTransition(PN, t_{new})$ $addPtoTarc(PN, p_{src}, t_{new})$ $addTtoParc(PN, t_{new}, p_{dst})$ $\}$	$p_{src}, p_{dst} \in P$ $t_{new} \notin T$
$E = addPlace(PN, p_{new})$	$p_{new} \notin P$
$E = addToken(PN, p_{tar})$	$p_{tar} \in P$

Table 3.1 Basic edit actions of the first group

Table 3.1 shows the pre-condition(s) according to each addition action above. The parameters and pre-condition of basic edit action E in Table 3.1 are explained, using E = $addPtoTarc(PN, p_{src}, t_{dst})$ as an example. $PN = (P, T, F, m_0)$ is the original net before applying E and the rest parameters p_{src} and t_{dst} contains the information of the newly added arc. The pre-conditions ensure that newly added flow relations can obtain the correct source place or destination transition in PN. The difference between F and F' is the newly added arc (p_{src}, t_{dst}). In addition, *addTrans* is the specific action for our editor and contains a batch of actions corresponding to the newly added transition, "virtual p_{src}", and "virtual p_{dst}".

$E = delTransition(PN, t_{tar})$	(a) 
$E = delPlace(PN, p_{tar})$	(b) 
$E = delTtoParc(PN, t_{src}, p_{dst})$	(c) 
$E = delToken(PN, p_{tar})$	(d) 

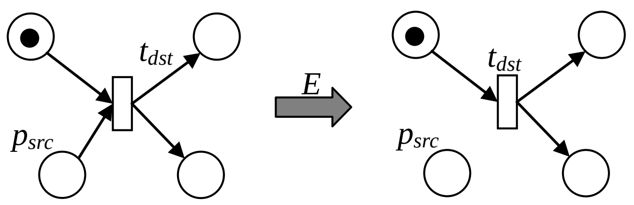
$E = delPtoTarc(PN, p_{src}, t_{dst})$	(e) 

Figure 3.2 Examples of the basic edit actions in the second group

Figure 3.2 shows a set of deletion actions, where (c), (d), and (e) show the deletion of a TtoP arc, token, and PtoT arc respectively. In order to ensure that every transition has both input and output places and each arc connects a place and a transition in the net, there are two constraints specified in our editor in addition: (1) When a place or transition is deleted, the connected arcs are killed automatically. (2) A deletion is not allowed to make a transition has no input or output arc.

Basic editing action	Pre-condition
$E = delTransition(PN, t_{tar})$	$t_{tar} \in T$
$E = delTtoParc(PN, t_{src}, p_{dst})$	$(t_{src}, p_{dst}) \in F$ $\exists(t_{src}, p_j) \in F: p_j \neq p_{dst}$
$E = delPtoTarc(PN, p_{src}, t_{dst})$	$(p_{src}, t_{dst}) \in F$ $\exists(p_i, t_{dst}) \in F: p_i \neq p_{src}$
$E = delToken(PN, p_{tar})$	$p_{tar} \in P$ $m_0(p_{tar}) > 0$
$E = delPlace(PN, p_{tar})$	$p_{tar} \in P$ $\forall(p_{tar}, t_j) \in F, \exists(p_i, t_j) \in F: p_i \neq p_{tar}$ $\forall(t_i, p_{tar}) \in F, \exists(t_i, p_j) \in F: p_j \neq p_{tar}$

Table 3.2 Basic edit actions of the second group

Table 3.2 shows the pre-condition(s) according to each deletion action above. In *delPlace*(PN, p_{tar}), p_{tar} and the arcs connected with p_{tar} are deleted. The pre-conditions ensure that the deletions of these arcs do not violate constraint (2).

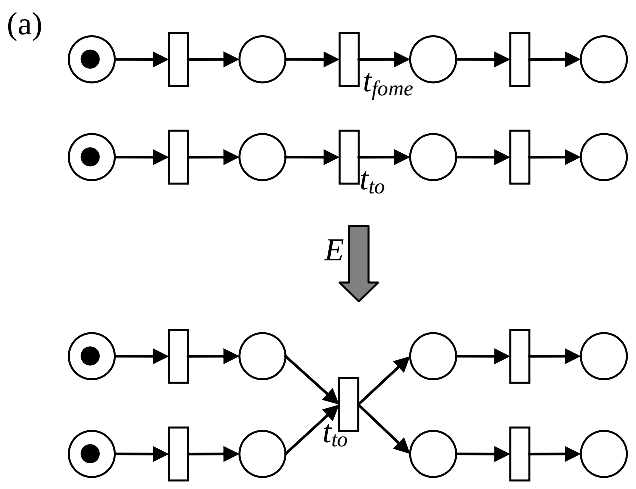
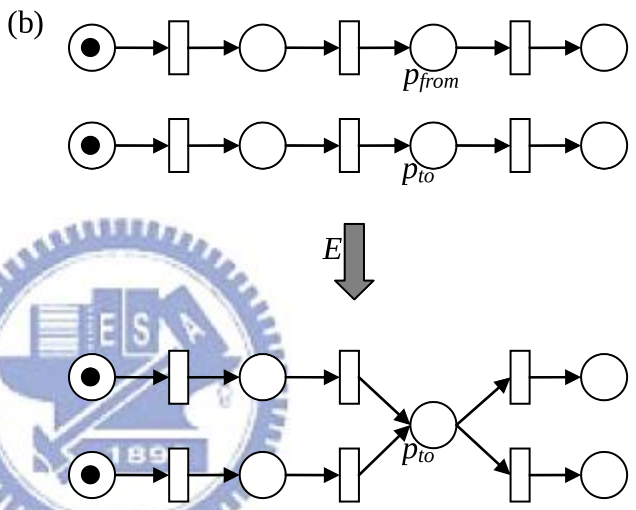
$E = mergeTrans(PN, t_{from}, t_{to})$	
$E = mergePlace(PN, p_{from}, p_{to})$	

Figure 3.3 Examples of the basic edit actions in the third group

Figure 3.3 shows a set of edit actions in the third group, where Figure 3.3(a) and (b) show the mergence of two transitions or places respectively. When a designer drags transition t_{from} to transition t_{to}, t_{from} and t_{to} are merged by redirecting the arcs connected with t_{from}, and then deleting t_{from}. The redirections are done as following: (1) For the arc whose head is t_{from}, its head is changed to t_{to}. (2) For the arc whose tail is t_{from}, its tail is changed to t_{to}. The mergence of two places are similar, shown in Figure 3.3(b), except that the tokens in p_{from} and p_{to} of the original net are put in p_{to} of the refined net.

Basic editing action	Pre-condition
$E = mergeTrans(PN, t_{from}, t_{to})$	$t_{from}, t_{to} \in T$
$E = mergePlace(PN, p_{from}, p_{to})$	$p_{from}, p_{to} \in P$

Table 3.3 Basic edit actions of the third group

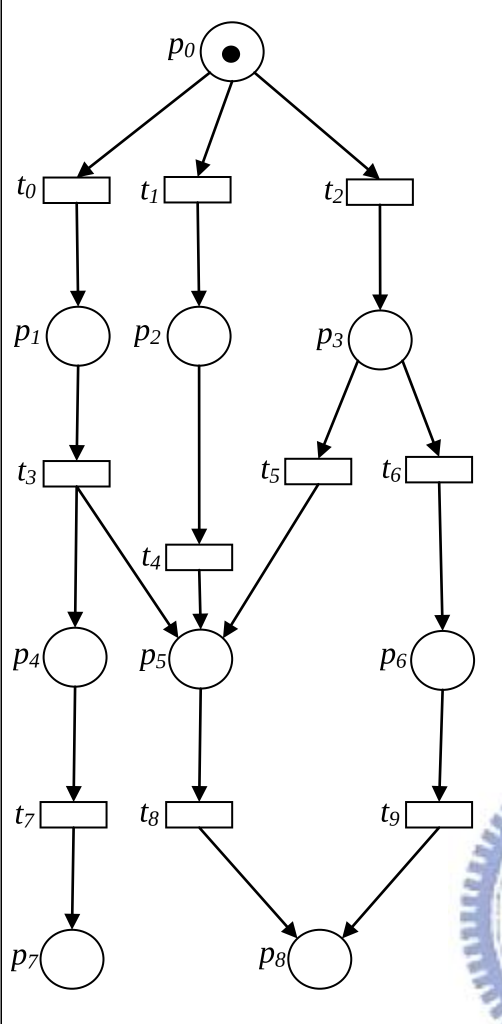
In $mergeTrans(PN, t_{from}, t_{to})$, t_{from} is merged into t_{to} in *PN*. In $mergePlace(PN, p_{from}, p_{to})$, p_{from} is merged into p_{to} in *PN*.

3.2 The Examples of O-graph Change for Item Insertion

This section presents two kinds of O-graph changes, called modification and extension, due to Petri Net edit actions. A modification is done by modifying the nodes in the O-graph. An extension from one O-graph to the other can be represented by the difference of them. They will be discussed accordingly with addTtoParc, addToken, addTrans, and addPtoTarc examples.

1. addTtoParc example

Figure 3.4 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph, *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.5.



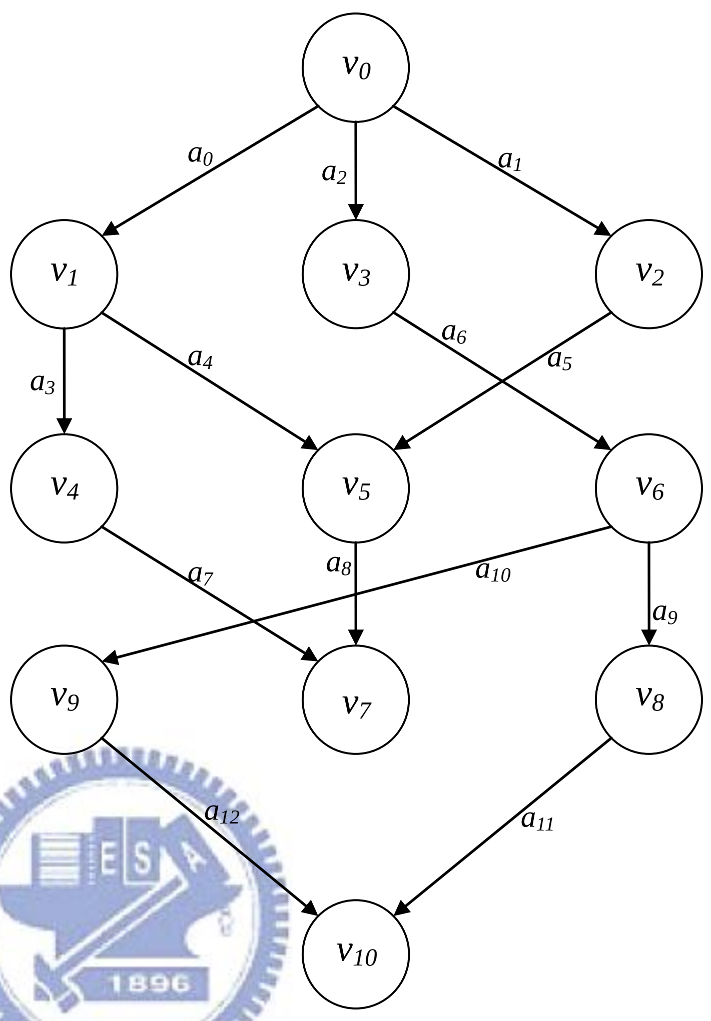


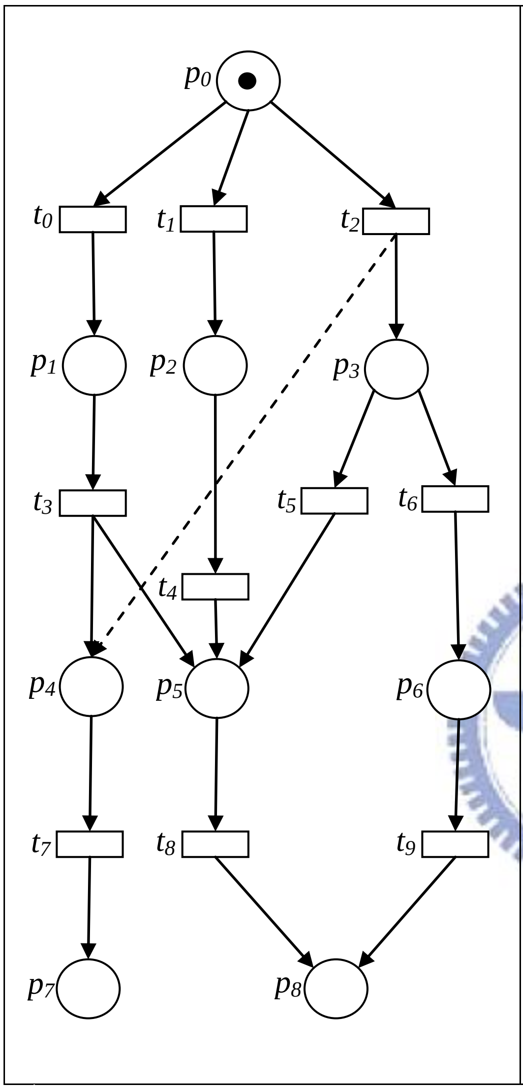
Figure 3.4 Figure 3.5

Figure 3.4 The original net *PN* of an addTtoParc example.

Figure 3.5 The O-graph *OG* of Figure 3.4.

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.1(a) and Table A3.1(b) respectively. Table A3.1(a) contains the marking and the canonical firing count vector of each node in Figure 3.5. Table A3.1(b) contains the source marking, transition, and destination marking corresponding to each arc. In order to identify ignored firing count vector, each arc also has a flag called *atf*. There are two firing count vectors, (0,0,1,0,0,1,0,0,0,0) and (0,1,0,0,1,0,0,0,0,0), corresponding to node v_5 in Figure 3.5. Assume m_5 is constructed firstly after t_5 is fired at m_1,and then m_5 has canonical firing count vector

$\vec{\sigma}_5^*$=(0,0,1,0,0,1,0,0,0,0), shown in Table A3.1(a). When t_4 is fired at m_2, arc a_5 = (m_2,t_4,m_5) is created in *A*. Since $\vec{\sigma}_5$=(0,1,0,0,1,0,0,0,0,0) generated by a_5 is ignored in v_5, atf_5 is set true.



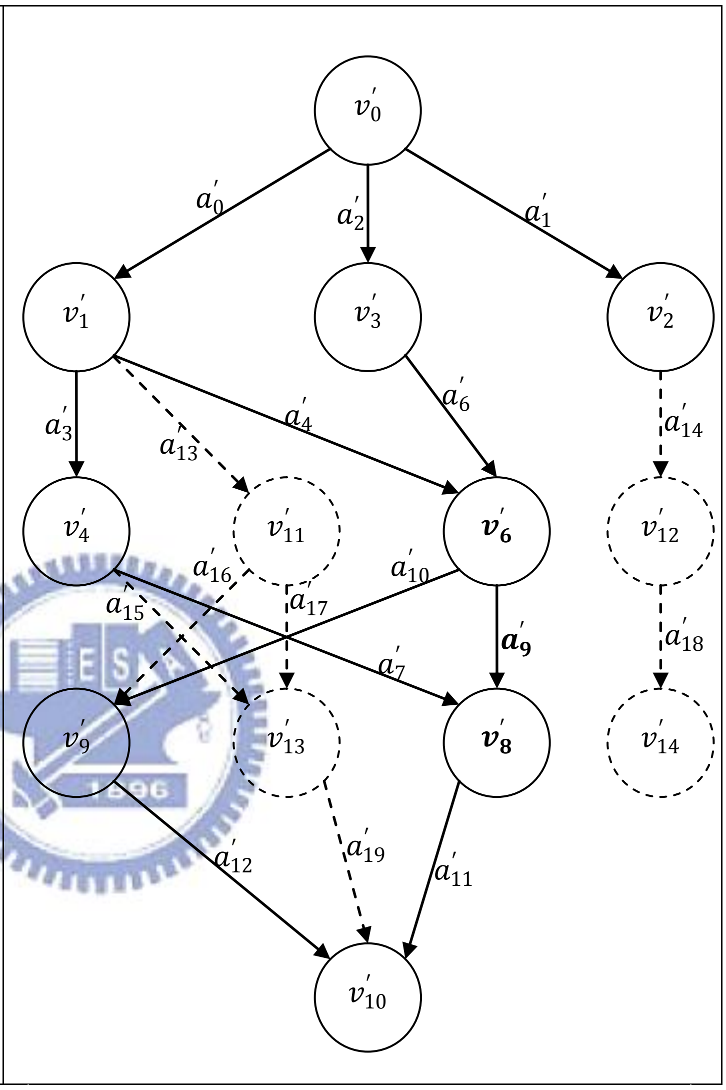


Figure 3.6 Figure 3.7

Figure 3.6 The refined net *PN'* of an addTtoParc example.

Figure 3.7 The O-graph *OG'* of Figure 3.6.

Let a designer add a new arc from transition t_2 to place p_4 in Figure 3.4, the Petri net *PN* would become the refined net *PN'* in Figure 3.6. Before addition of arc (t_2, p_4), the tuple of transition t_2 in {$\vec{\sigma}_1^*$, $\vec{\sigma}_4^*$, $\vec{\sigma}_5^*$, $\vec{\sigma}_7^*$} is one, shown in Table A3.1(a). The marking change is from m_4=(0,0,0,0,**0**,0,1,0,0) to m_4'=(0,0,0,0,**1**,0,1,0,0) in Table A3.2(a). m_4' has a token in place p_4

due to newly added arc and $\vec{\sigma}_4^*[t_2]$. Each marking in *OG* can be modified as above. However, modified m_5 is the same as modified m_6. Thus, $\{v_5, v_6\}$ and $\{a_4, a_5, a_6, a_8, a_9, a_{10}\}$ becomes $\{v_6'\}$ and $\{a_4', a_6', a_9', a_{10}'\}$. Since v_5 ignores the firing count vectors (0,1,0,0,1,0,0,0,0,0), a_5 = (m_2,t_4,m_5) corresponding to ignored $\vec{\sigma}_5$ is deleted and re-generated arc of a_5 is a_{14}' = (m_2',t_4,m_{12}').

In Figure 3.7, $\{v_{12}', v_{14}'\}$ and $\{a_{18}'\}$ are generated from the re-generated arc a_{14}'. $\{m_1', m_4'\}$ have tokens in place p_4 and marking $\{m_1, m_4\}$ do not have any tokens in place p_4. Thus, $\{a_{13}', a_{15}', a_{16}', a_{17}', a_{19}'\}$ and $\{v_{11}', v_{13}'\}$ are generated from m_1' and m_4'. The extension of the O-graph is the node set $\{v_{11}', v_{12}', v_{13}', v_{14}'\}$ and the arc set $\{a_{13}', a_{14}', a_{15}', a_{16}', a_{17}', a_{18}', a_{19}'\}$, shown in Figure 3.7. Because $\{m_6', m_8', m_{10}'\}$ have two firing count vectors and $\{a_6', a_9', a_{11}'\}$ are modified from *A*, $\{atf_4', atf_7', atf_{16}', atf_{19}'\}$ are set true.

2. addPtoTarc example

Figure 3.8 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.9.

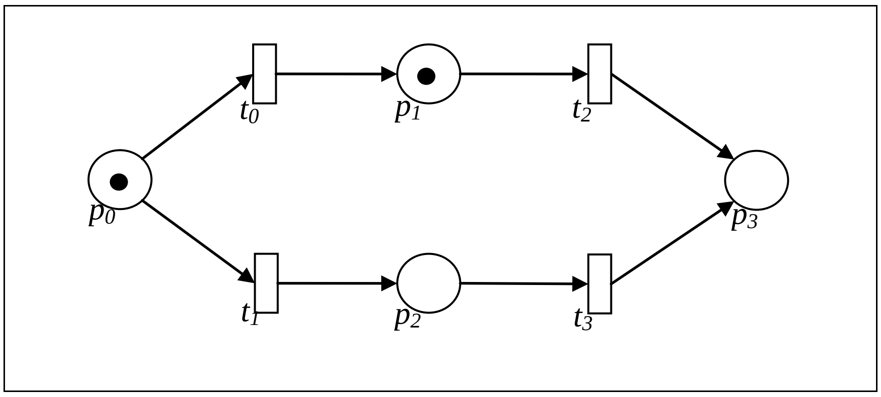


Figure 3.8 The original net *PN* of an addPtoTarc example

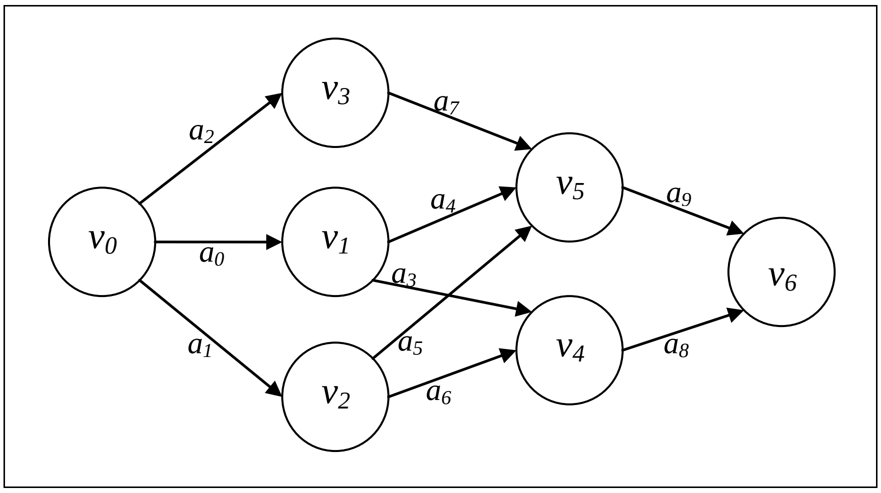


Figure 3.9 The O-graph *OG* of Figure 3.8

After a designer adds a new arc from place p_1 to transition t_3 in Figure 3.8, the original net *PN* transfers to the refined net *PN'* in Figure 3.10. The tuple of transition t_3 in $\vec{\sigma}_6^*$ is one, shown in Table A3.3(a). m_6 is deleted the token at p_1 due to the newly added arc (p_1, t_3). Because $m_6[p_1]=0$, m_6 is deleted in Figure 3.11. Since atf_5 and atf_9 are true, a'_{10} and a'_{11} are generated.

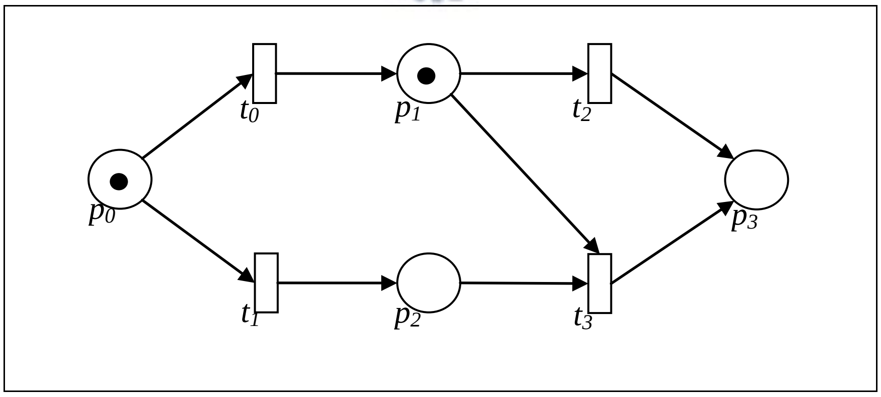


Figure 3.10 The refined net *PN'* of an addPtoTarc example

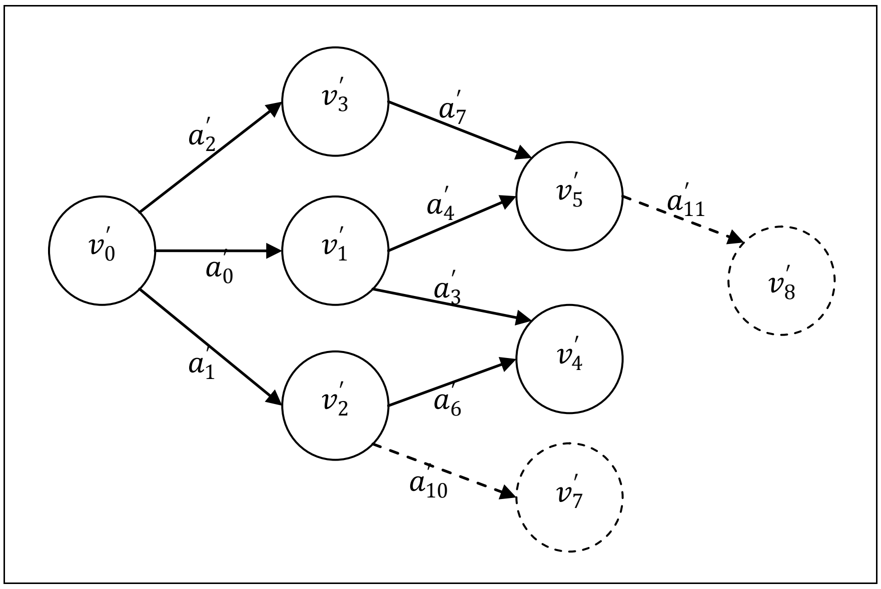


Figure 3.11 The O-graph *OG'* of Figure 3.10

The extension of the O-graph is the node set $\{v_7', v_8'\}$ and the arc set $\{a_{10}', a_{11}'\}$.

3. addToken example

Figure 3.12 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.13.

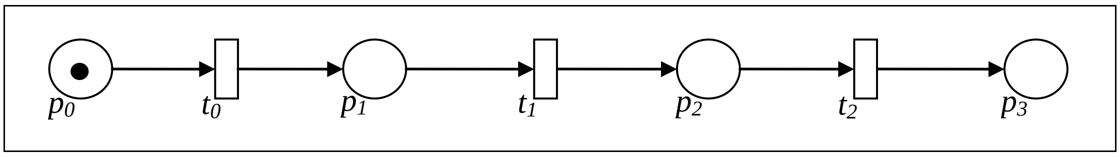


Figure 3.12 The original net *PN* of an addToken example

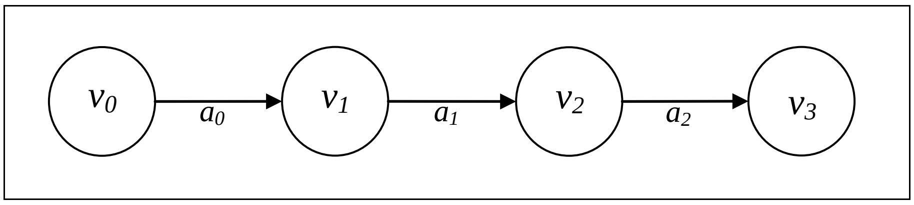


Figure 3.13 The O-graph *OG* of Figure 3.12

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.5(a) and Table A3.5(b) respectively. Table A3.5(a) contains the marking and the canonical firing count vector of each node in Figure 3.13. Table A3.5(b) contains the source marking, transition, and destination marking corresponding to each arc.

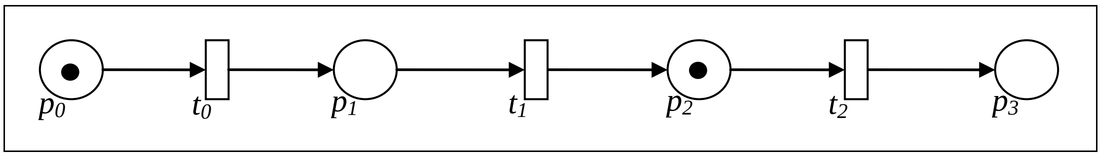


Figure 3.14 The refined net *PN'* of an addToken example

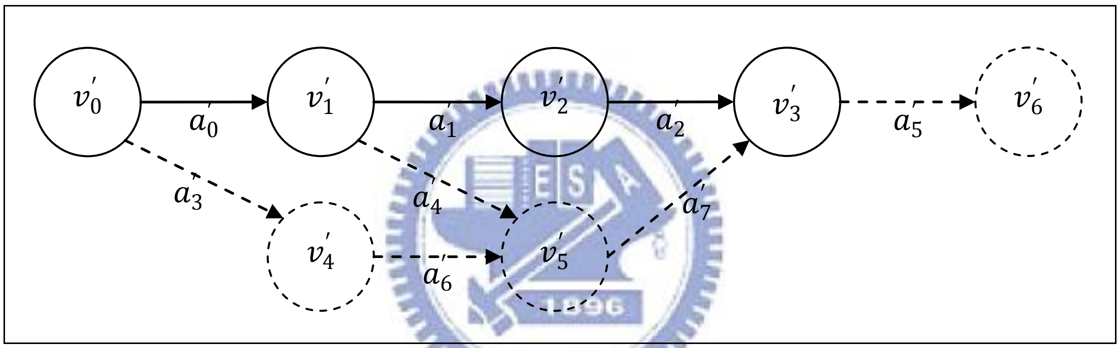


Figure 3.15 The O-graph *OG'* of Figure 3.14

Let a designer add a new token to place p_2 in Figure 3.12, the Petri net *PN* would become the refined net *PN'* in Figure 3.14. The marking change is from m_2=(0,0,1,0) to m_2'=(0,0,2,0), shown in Table A3.6(a). m_4' has one more token in place p_2 due to newly added token in p_2. Each marking in *OG* can be modified as above.

The extension of the O-graph is the node set $\{v_4', v_5', v_6'\}$ and the arc set $\{a_3', a_4', a_5', a_6', a_7'\}$, shown in Figure 3.15.

4. addTrans example

Let the original net *PN* in Figure 3.16 be edited at present. Figure 3.17 shows the O-graph of Figure 3.16.

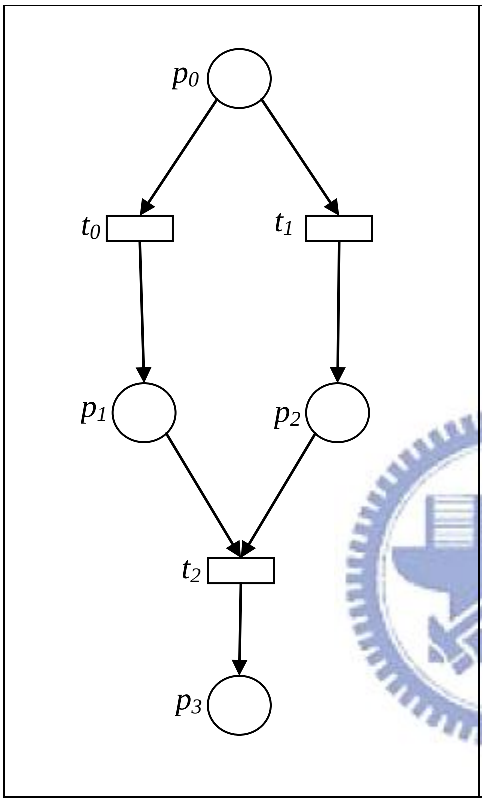


Figure 3.16

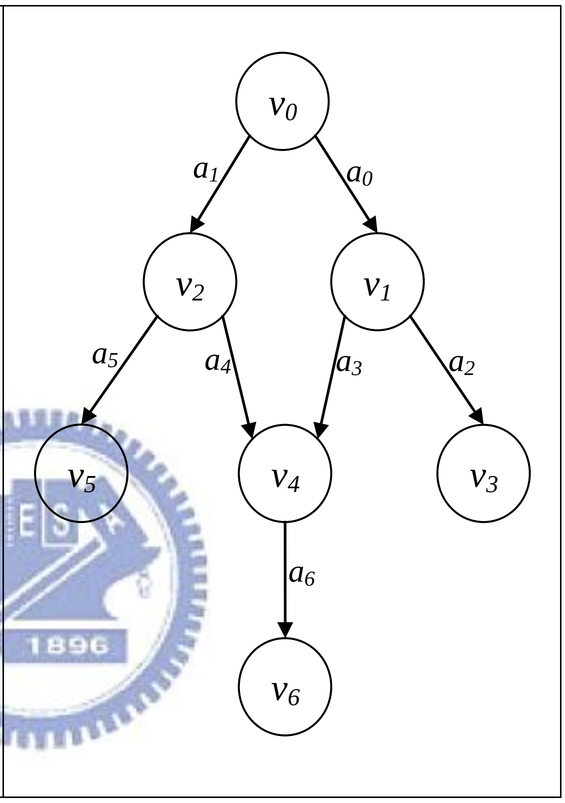


Figure 3.17

Figure 3.16 The original net *PN* of an addTrans example.

Figure 3.17 The O-graph *OG* of Figure 3.16.

Although m_0 [t_0 > m_2 [t_1 > m_4 and m_0 [t_1 > m_1 [t_0 > m_4, the firing count vectors for these firing sequences from m_0 to m_4 are both $\vec{\sigma}_4^*$=(1,1,0), shown in Table A3.7(a).

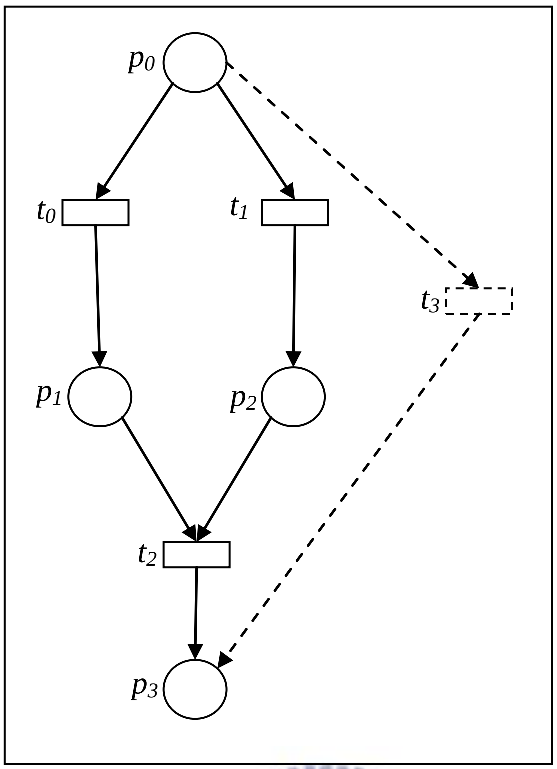


Figure 3.18 The refined net *PN'* of an addTrans example

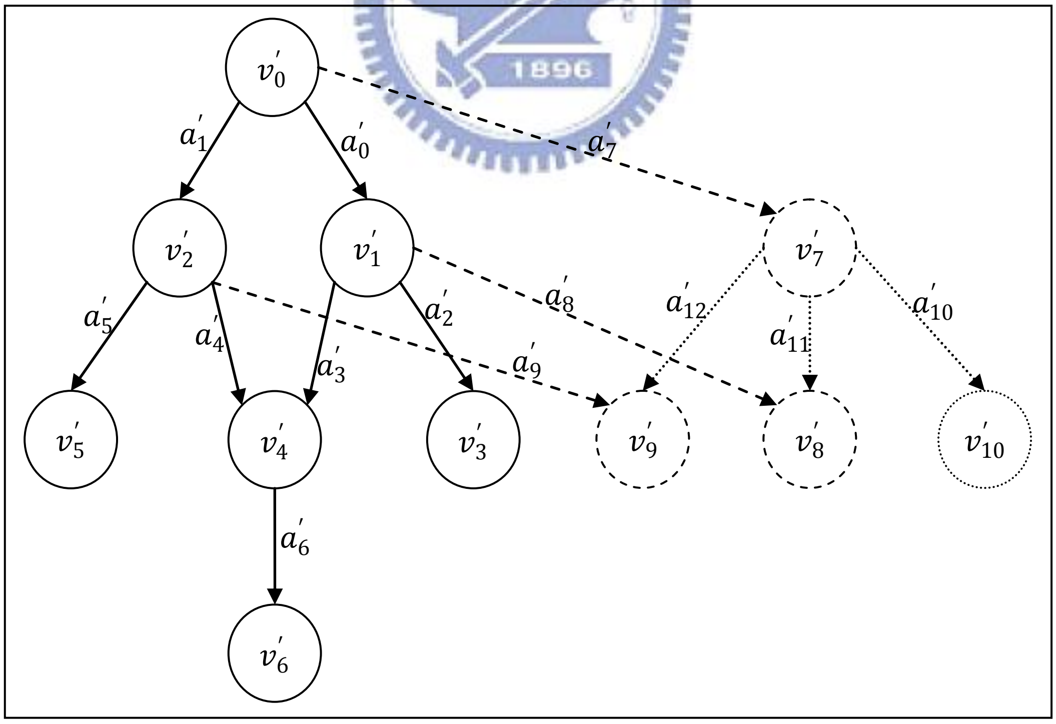


Figure 3.19 The O-graph *OG'* of Figure 3.18

After a designer adds a new transition t_3 from place p_0 to place p_3 in Figure 3.16, the original net *PN* transfers to the refined net *PN'* in Figure 3.18. On account of the newly added transition t_3, the canonical firing count vector of each vertex in Figure 3.17 would need an additional tuple for t_3 as $\{\vec{\sigma}_0^{*'}, \vec{\sigma}_1^{*'}, \vec{\sigma}_2^{*'}, \vec{\sigma}_3^{*'}, \vec{\sigma}_4^{*'}, \vec{\sigma}_5^{*'}, \vec{\sigma}_6^{*'}\}$ in Table 3.8(a).

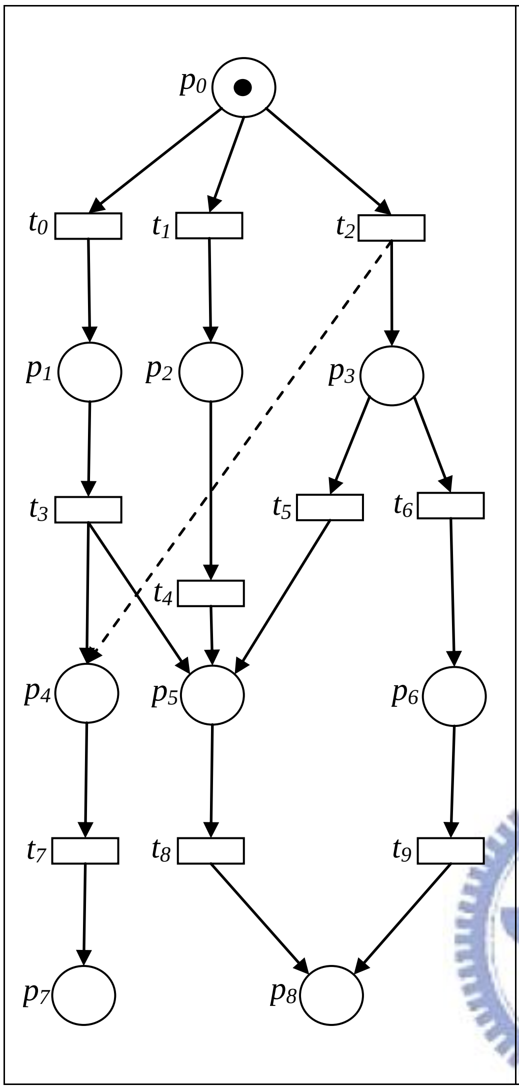
The extension of the O-graph is the node set $\{v_7', v_8', v_9', v_{10}'\}$ and the arc set $\{a_7', a_8', a_9', a_{10}', a_{11}', a_{12}'\}$. Since $\{m_0', m_1', m_2'\}$ have tokens in place p_0, these markings enable newly added transition t_3. Thus, $\{a_7', a_8', a_9'\}$ and $\{v_7', v_8', v_9'\}$ are generated after t_3 is fired at $\{m_0', m_1', m_2'\}$.

3.3 The Examples of O-graph Change for Item Deletion and Mergence

Modifications and extensions will be discussed accordingly with delTtoParc, delPtoTarc, delToken, delTransition, delPlace, and mergePlace examples.

5. delTtoParc example

Figure 3.20 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph, *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.21.



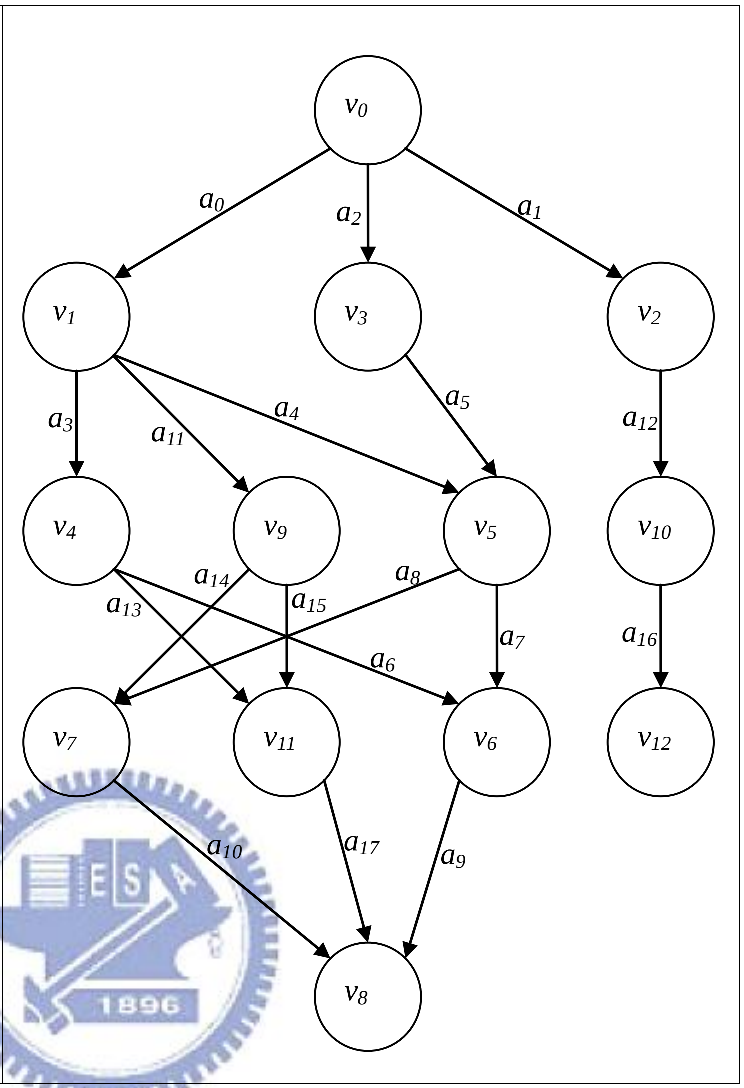
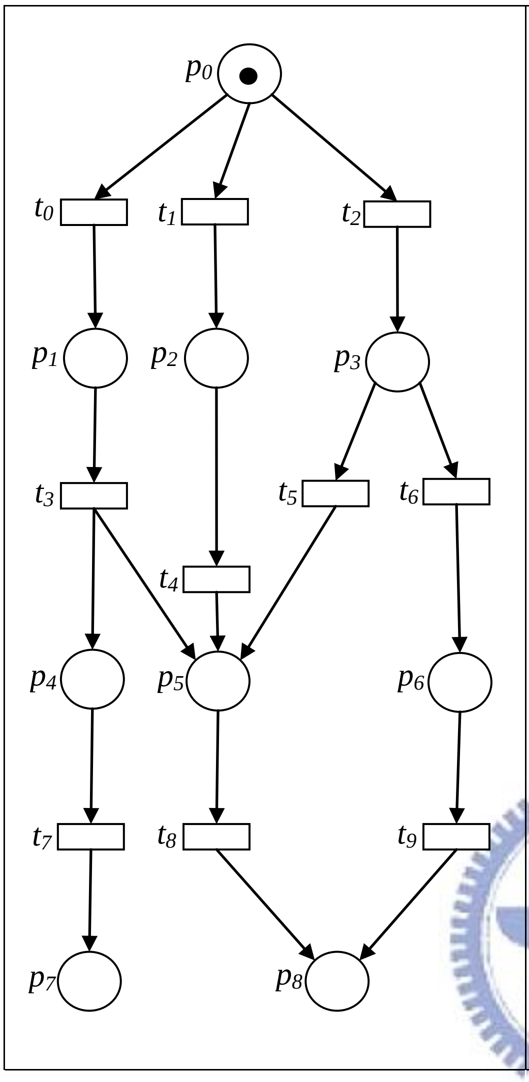


Figure 3.20 Figure 3.21

Figure 3.20 The original net *PN* of an delTtoParc example.

Figure 3.21 The O-graph *OG* of Figure 3.20.

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.9(a) and Table A3.9(b) respectively. Table A3.9(a) contains the marking and the canonical firing count vector of each node in Figure 3.21. Table A3.9(b) contains the source marking, transition, and destination marking corresponding to each arc.



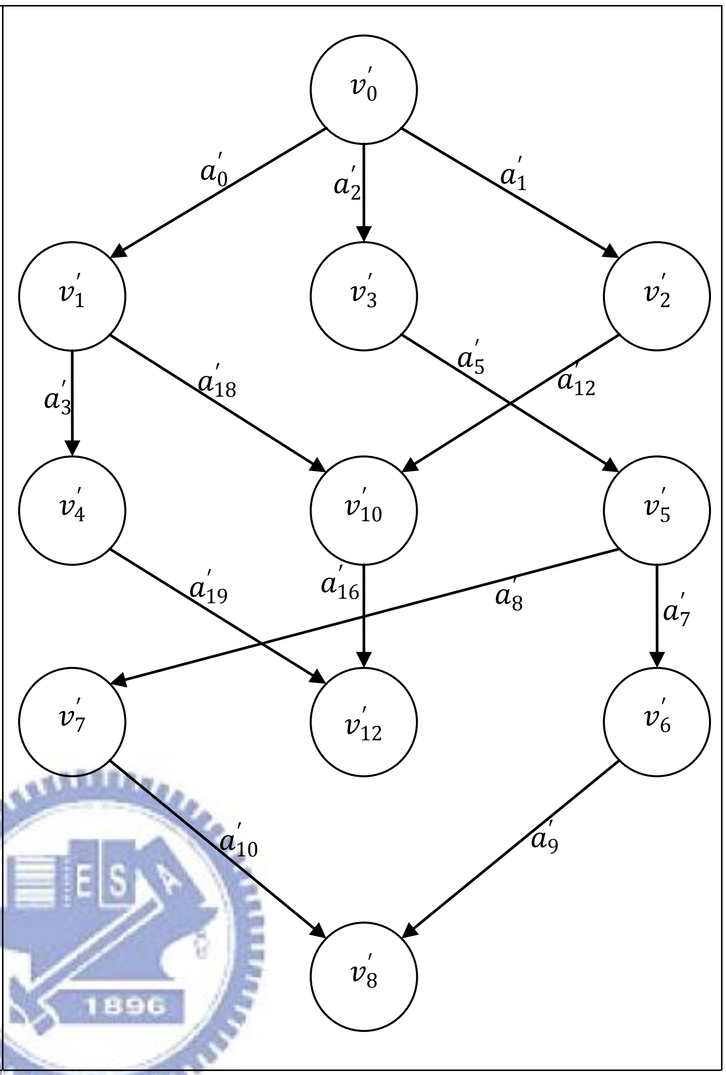


Figure 3.22 Figure 3.23

Figure 3.22 The refined net *PN'* of an delTtoParc example.
Figure 3.23 The O-graph *OG'* of Figure 3.22.

Let a designer delete the arc from transition t_2 to place p_4 from Figure 3.20, the Petri net *PN* would become the refined net *PN'* in Figure 3.22. Before deletion of arc (t_2, p_4), the tuple of transition t_2 in $\{\vec{\sigma}_1^*, \vec{\sigma}_4^*, \vec{\sigma}_9^*, \vec{\sigma}_{11}^*\}$ is one, shown in Table A3.9(a). The marking change is from m_4=(0,0,0,0,**1**,0,1,0,0) in *OG* to m_4'=(0,0,0,0,**0**,0,1,0,0) in Table A3.10(a). $m_4[p_4]$ is one due to arc (t_2, p_4) and $\vec{\sigma}_4^*[t_2]$. Each marking in *OG* can be modified as above. However, $m_9[p_4] < \vec{\sigma}_9^*[t_2]$ causes that the tuple of p_4 in modified m_9 becomes negative. Thus, node v_9 and the arc set $\{a_{11}, a_{14}, a_{15}\}$ are deleted. In Figure 3.23, $\{a_{18}', a_{19}'\}$ are generated since atf_4 and atf_6 are true.

6. delPtoTarc example

Figure 3.24 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.25.

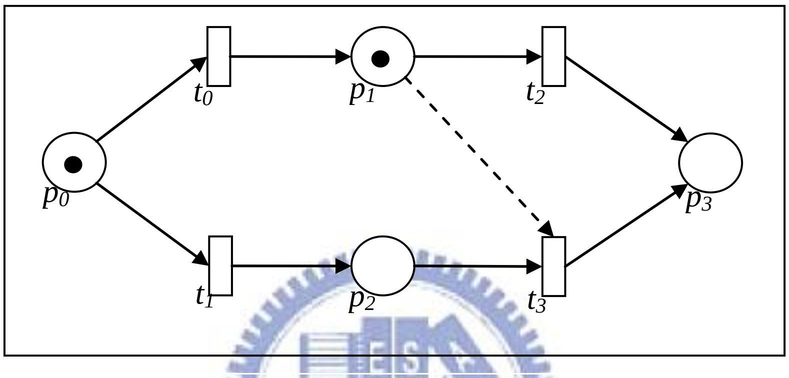


Figure 3.24 The original net *PN* of an delPtoTarc example

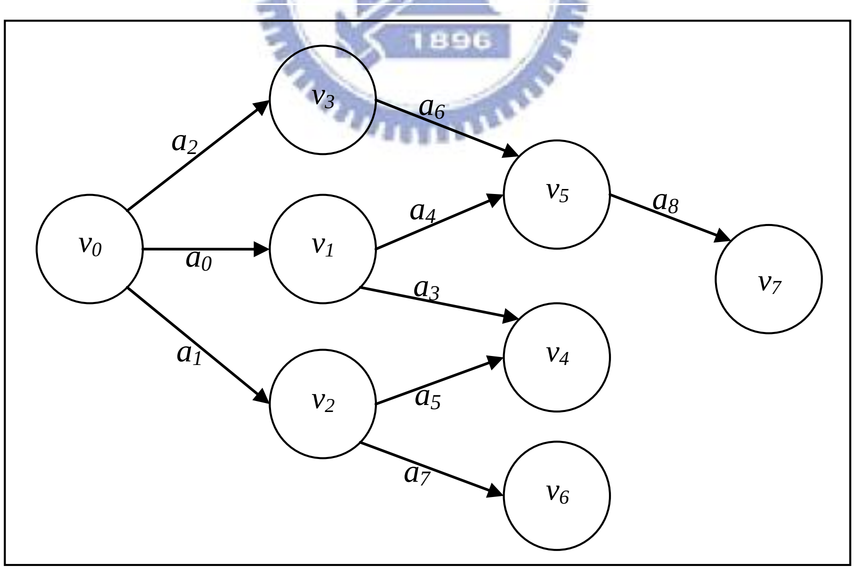


Figure 3.25 The O-graph *OG* of Figure 3.24

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.11(a) and Table A3.11(b) respectively. Table A3.11(a) contains the marking and the canonical firing count vector of each node in Figure 3.25. Table A3.11(b) contains the source marking, transition, and destination marking corresponding to each arc.

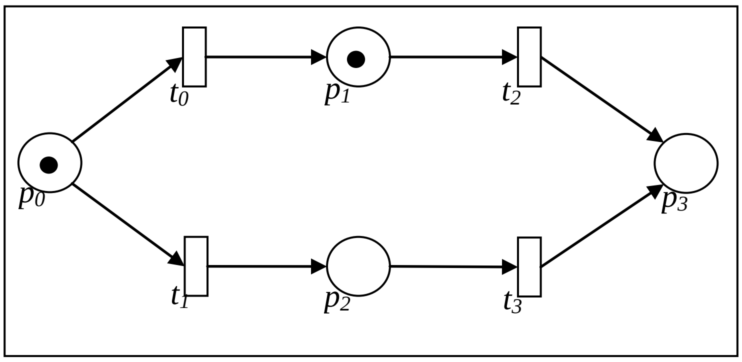


Figure 3.26 The refined net *PN'* of an delPtoTarc example

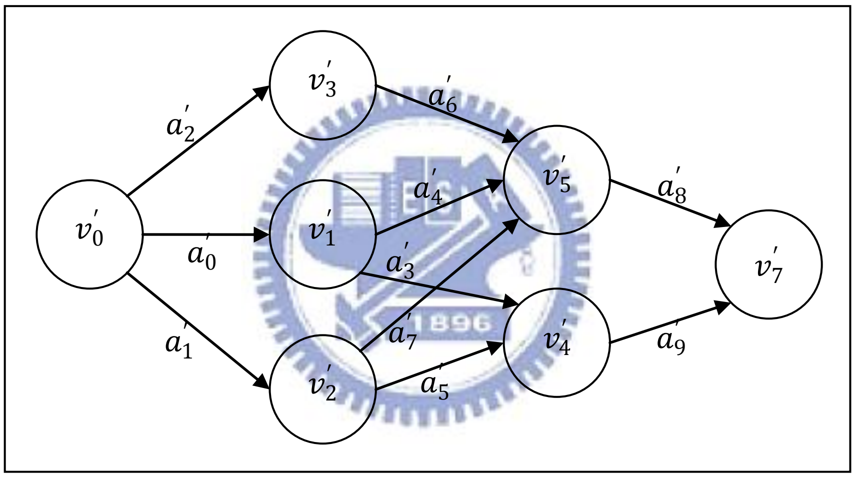


Figure 3.27 The O-graph *OG'* of Figure 3.26

Let a designer delete the arc from place p_1 to transition t_3 in Figure 3.8, the Petri net *PN* would become the refined net *PN'* in Figure 3.26.

The extension of the O-graph is arc a_9', shown in Figure 3.27.

7. delToken example

Figure 3.28 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph, *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in

Figure 3.29.

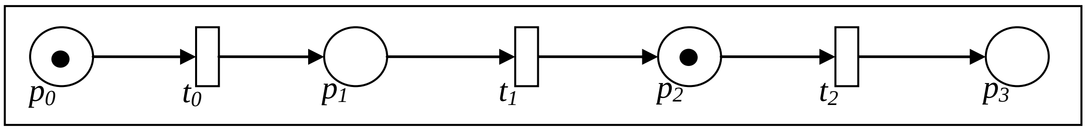


Figure 3.28 The original net *PN* of an delToken example

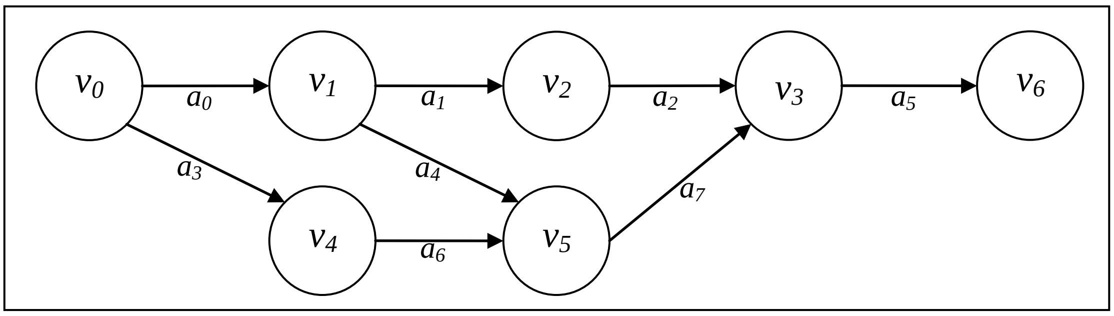


Figure 3.29 The O-graph *OG* of Figure 3.28

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.13(a) and Table A3.13(b) respectively. Table A3.13(a) contains the marking and the canonical firing count vector of each node in Figure 3.29. Table A3.13(b) contains the source marking, transition, and destination marking corresponding to each arc.

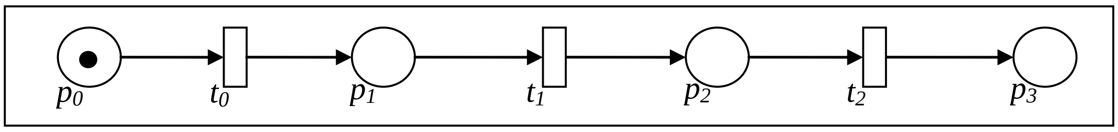


Figure 3.30 The refined net *PN'* of an delToken example

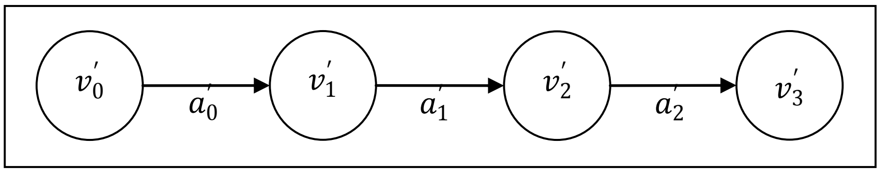


Figure 3.31 The O-graph *OG'* of Figure 3.30

Let a designer delete a token from p_2 in Figure 3.28, the Petri net *PN* would become the refined net *PN’* in Figure 3.30. The marking change is from m_0=(1,0,1,0) in *OG* to m'_0=(1,0,0,0) in Table A3.14(a). $m_0[p_2]$ is zero due to the deleted token. Each marking in *OG* can be modified as above. However, $m_4[p_2]$=0 causes that the tuple of p_2 in modified m_4 becomes negative. Thus, node v_4 and the arc set {a_3, a_6} are deleted.

8. delTransition example

Figure 3.32 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph, *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.33.

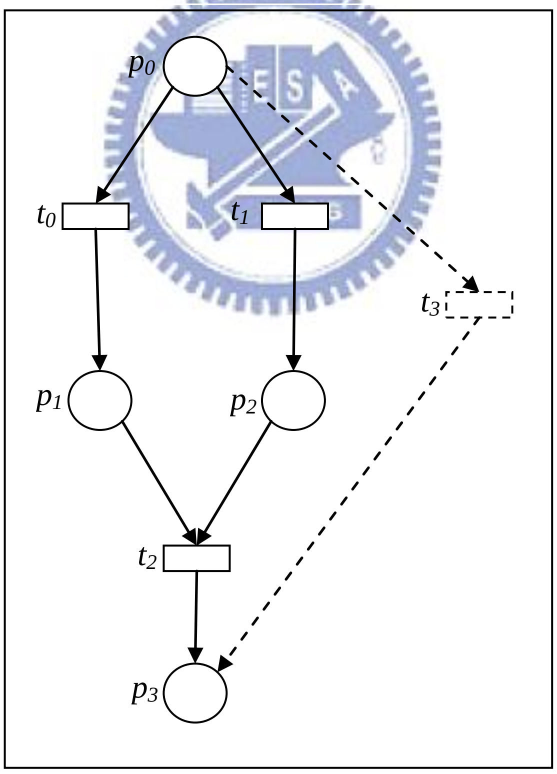


Figure 3.32 The original net *PN* of an delTrans example

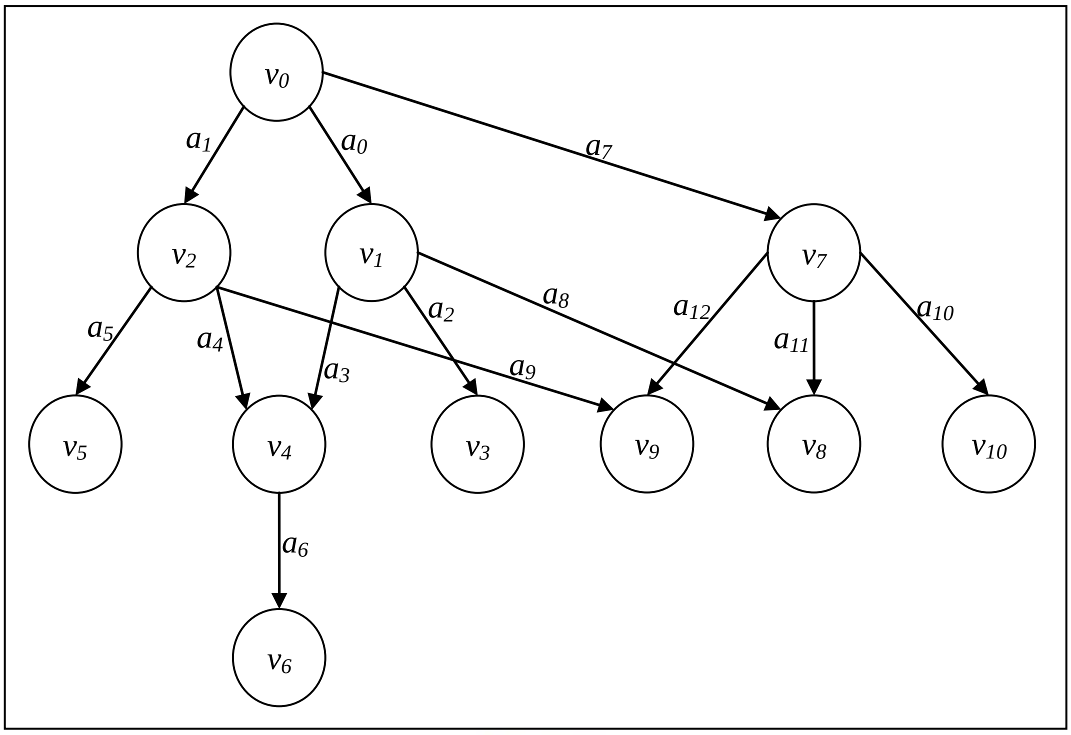


Figure 3.33 The O-graph *OG* of Figure 3.32

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.15(a) and Table A3.15(b) respectively. Table A3.15(a) contains the marking and the canonical firing count vector of each node in Figure 3.33. Table A3.15(b) contains the source marking, transition, and destination marking corresponding to each arc.

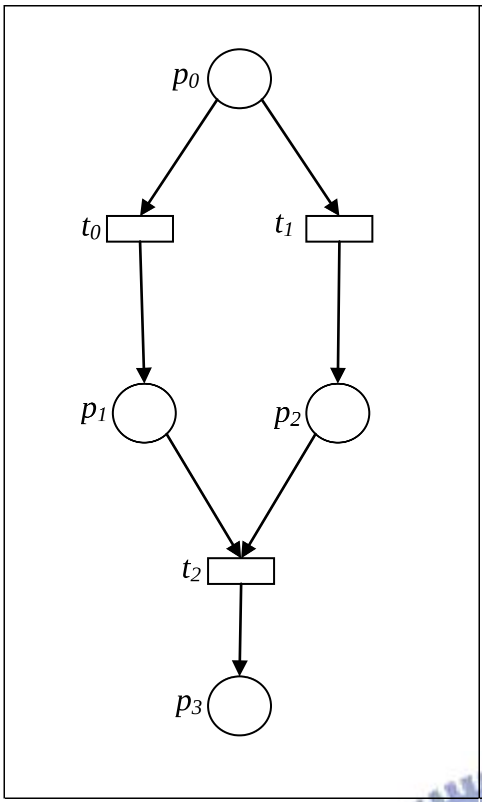


Figure 3.34

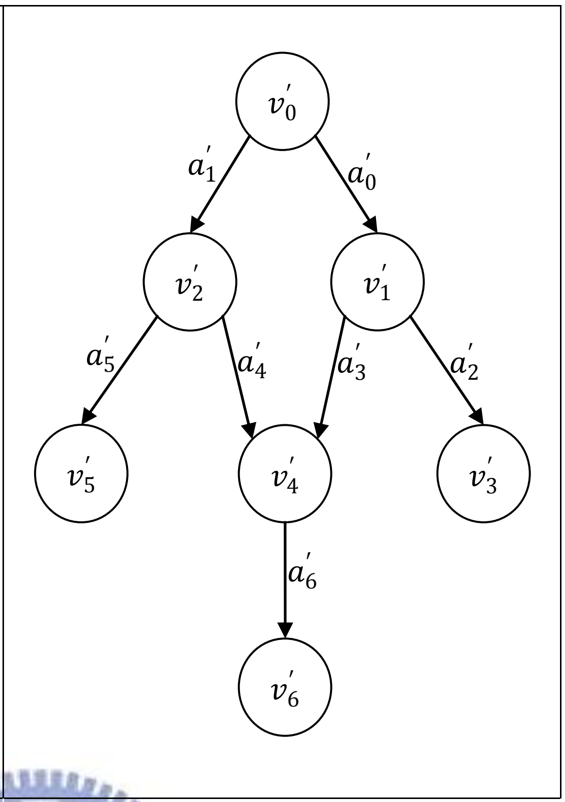


Figure 3.35

Figure 3.34 The refined net *PN'* of an delTrans example.

Figure 3.35 The O-graph *OG'* of Figure 3.34.

Let a designer delete the transition t_3 in Figure 3.32, the Petri net *PN* would become the refined net *PN'* in Figure 3.34. Since $\vec{\sigma}_7^*[t_3]>0$, node v_7 and the arc set $\{a_7, a_{10}, a_{11}, a_{12}\}$ are deleted.

9. delPlace example

Figure 3.36 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph, *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.37.

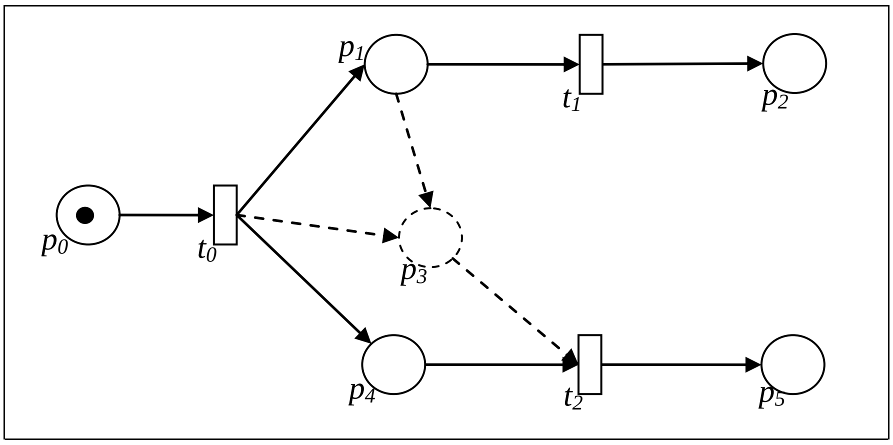


Figure 3.36 The original net *PN* of an delPlace example

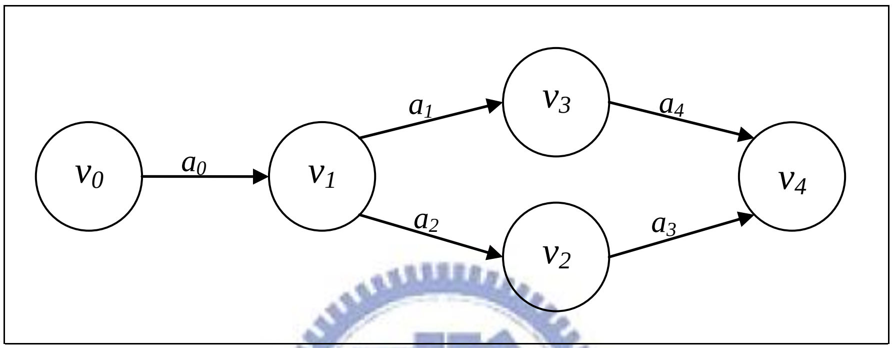


Figure 3.37 The O-graph *OG* of Figure 3.36

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.17(a) and Table A3.17(b) respectively. Table A3.17(a) contains the marking and the canonical firing count vector of each node in Figure 3.37. Table A3.17(b) contains the source marking, transition, and destination marking corresponding to each arc.

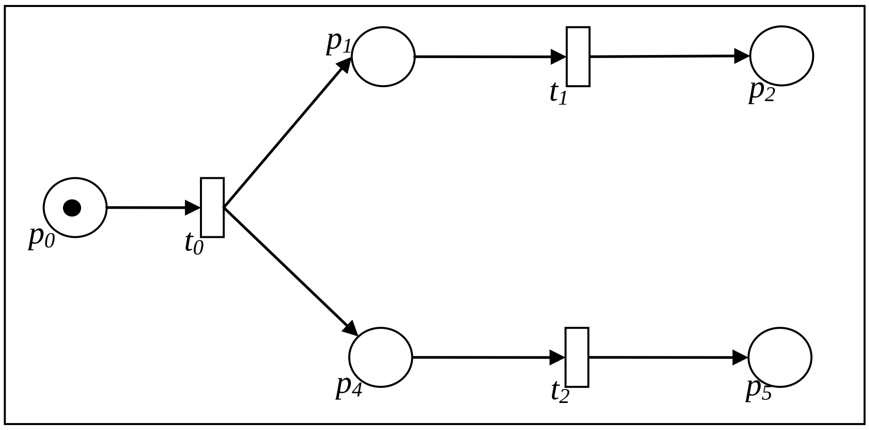


Figure 3.38 The refined net *PN'* of an delPlace example

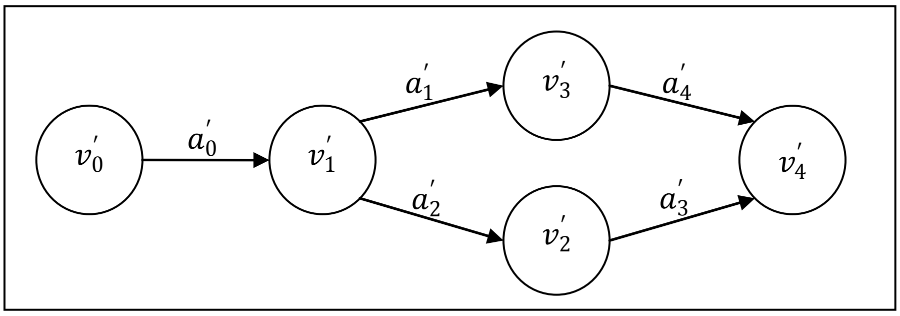
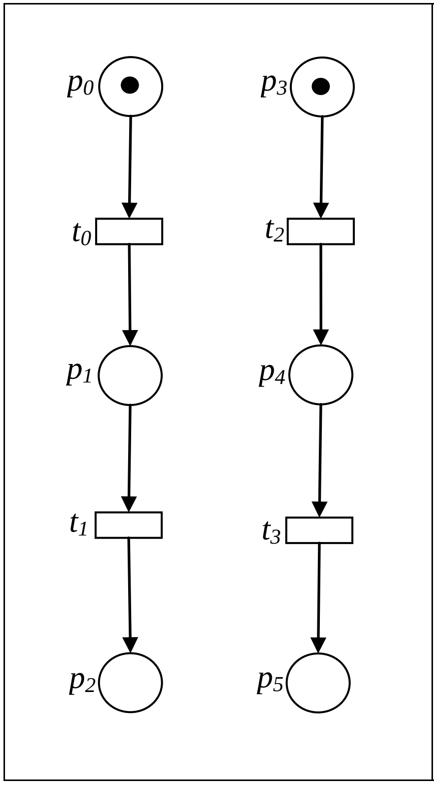


Figure 3.39 The O-graph *OG'* of Figure 3.38

Let a designer delete place p_3 in Figure 3.36, the Petri net *PN* would become the refined net *PN'* in Figure 3.38. The marking change is from m_4=(0,0,1,1,0,1) in *OG* to m_4'=(0,0,1,0,1) in Table A3.18(a). The tuple for p_3 in modified m_4 is deleted. Each marking in *OG* can be modified as above.

10.mergePlace example

Figure 3.40 is the original net *PN* needed for demonstration. Based on the definition of occurrence graph, *PN* has an O-graph named *OG* which contains two parts, *V* and *A*, shown in Figure 3.41.



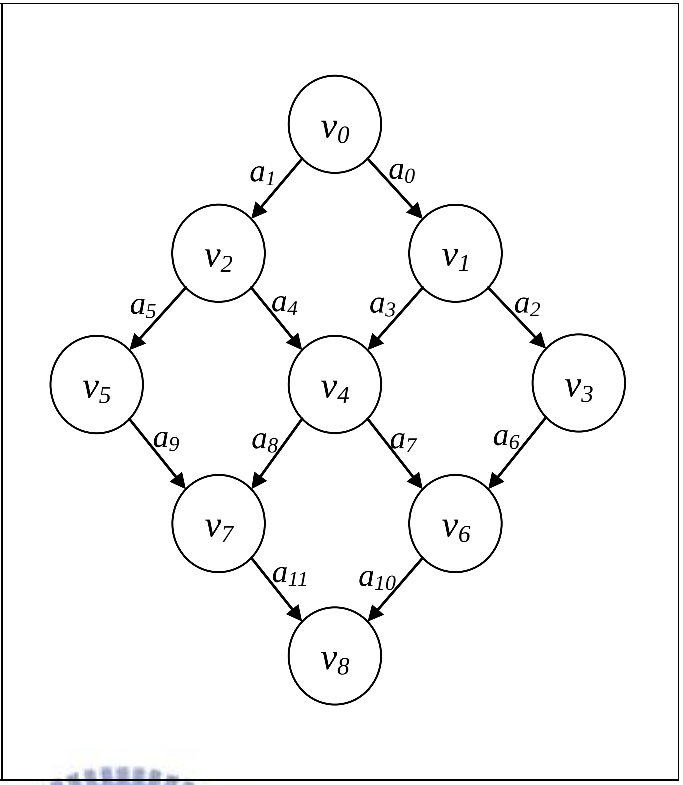
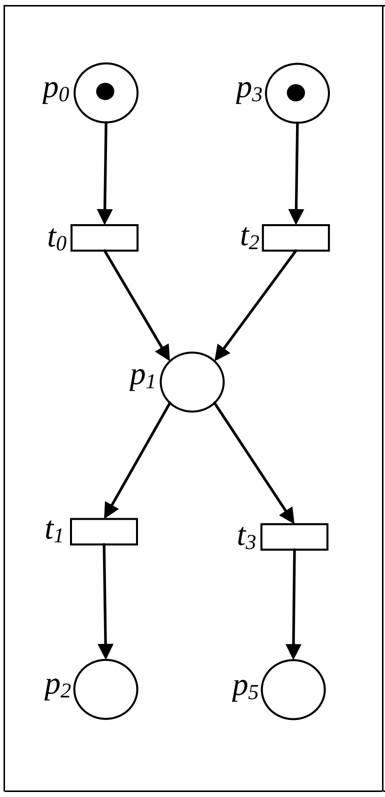


Figure 3.40 Figure 3.41

Figure 3.40 The original net *PN* of an mergePlace example.

Figure 3.41 The O-graph *OG* of Figure 3.40

The contents for nodes and arcs of an occurrence graph can be expressed by Table A3.19(a) and Table A3.19(b) respectively. Table A3.19(a) contains the marking and the canonical firing count vector of each node in Figure 3.41. Table A3.19(b) contains the source marking, transition, and destination marking corresponding to each arc.



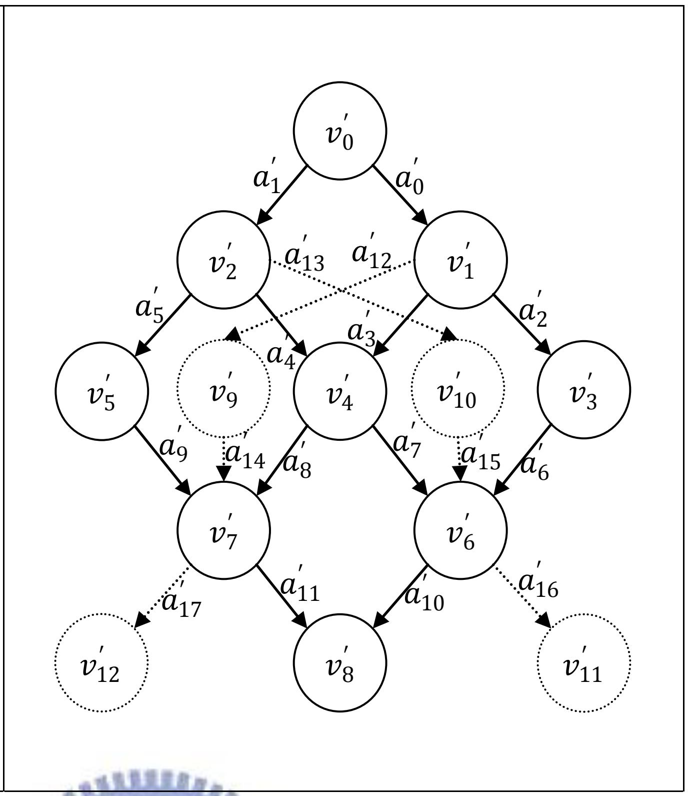


Figure 3.42 Figure 3.43

Figure 3.42 The refined net *PN'* of an mergePlace example.

Figure 3.43 The O-graph *OG'* of Figure 3.42.

Let a designer merge two places from place p_4 to place p_1 in Figure 3.42, the Petri net *PN* would become the refined net *PN'* in Figure 3.42. The marking change is from m_4=(0,1,0,0,1,0) to m'_4=(0,2,0,0,0) in Table A3.20(a). m'_4 has two tokens in place p_1 due to $m_4[p_1]$ and $m_4[p_4]$. Each marking in *OG* can be modified as above.

In Figure 3.43, $\{m'_1, m'_7\}$ can enable t_1 and $\{m_1, m_7\}$ can not enable t_1. Thus, $\{a'_{12}, a'_{17}\}$ and $\{v'_9, v'_{12}\}$ are generated from m'_1 and m'_7. The extension of the O-graph is the node set $\{v'_9, v'_{10}, v'_{11}, v'_{12}\}$ and the arc set $\{a'_{12}, a'_{13}, a'_{14}, a'_{15}, a'_{16}, a'_{17}\}$, shown in Figure 3.43.

Chapter 4. Maintaining the O-graph for Item Insertion

When editing a Petri net, every time users can insert one of the following items: (1) an arc, (2) a token, (3) a transition, or (4) a place. Correspondingly, the O-graph of the Petri net has to be modified and extended. The modifications and extensions are discussed in the following subsections respectively.

4.1 Arc Insertion

There are two types of arc insertions: (1) inserting an arc from a transition to a place and (2) inserting an arc from a place to a transition. Section 4.1.1 discusses the maintenance of the O-graph for the former and section 4.1.2 for the letter.

4.1.1 TtoP arc insertion

4.1.1.1 Scenario of TtoP arc insertions

Let a relation which belongs to $T \times P$ be named as a *TPR* relation. The addTtoParc example in section 3.2 can be translated into: *PN* is transformed into *PN'* by applying *addTtoParc* with a *TPR* relation (t_2, p_4). A formal definition of a *TPR* addition is described in Definition 4.1.

Definition 4.1 (*TPR* addition)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by adding a *TPR* relation *tpr* into *PN* has the following properties:
(1) $P' = P$;

(2) $T' = T$;
(3) $F' = F \cup \{ tpr \}$.

In a *TPR* addition, both the original net *PN* and the refined net *PN'* have the same transitions, places, and tokens in the places, but PN' has one more arc representing the *TPR* relation added. Therefore, the initial markings of *PN* and *PN'* are equivalent, i.e. $m_0 = m_0'$.

Definition 4.2 (Canonical Transition Firing Count Vector $\vec{\sigma}_j^*$)
Let a Petri net be $PN = (P, T, F, m_0)$ and the O-graphs $OG = (V, A)$ belong to *PN*.
The *canonical transition firing count vector* $\boldsymbol{\vec{\sigma}_j^*}$ at vertex v_j is defined as $\forall\, v_j \in V$:

$$\begin{cases} \vec{\sigma}_j^* = \vec{0} & \text{, if } j = 0; \\ \exists\, (m_i, t_n, m_j) \in A: \vec{\sigma}_j^* = \vec{\sigma}_i^* + TCV.t_n & \text{, otherwise} \end{cases}$$

, where $\vec{0}$ denotes a vector of appropriate size |T| and all entries are zero.

There might be many distinct *firing count vector* $\vec{\sigma}_i$ at the vertex v_i. Ensuring that each vertex v_i has identical structure, we choose only one arc in the O-graph to construct the *canonical transition firing count vector* $\vec{\sigma}_i^*$ at v_i. Thus, the *canonical transition firing count vector* $\vec{\sigma}_i^*$ at v_i is unique.

Definition 4.3 (another firing arc set *ATF*)
Let a Petri net $PN = (P, T, F, m_0)$ have the O-graph $OG = (V, A)$. *OG* might have an *another firing arc set* $\boldsymbol{ATF} \subseteq A$,

$$\boldsymbol{ATF} = \{ a_k = (m_i, t_n, m_j) \in A \mid \vec{\sigma}_j^* \neq \vec{\sigma}_i^* + TCV.t_n \}.$$

There are two firing count vectors $\vec{\sigma}_5$ at v_5:
(0,0,1,0,0,1,0,0,0,0) = $\vec{\sigma}_1^*$ + $TCV.t_5$ = (0,0,1,0,0,0,0,0,0,0) + (0,0,0,0,0,1,0,0,0,0) and
(0,1,0,0,1,0,0,0,0,0) = $\vec{\sigma}_2^*$ + $TCV.t_4$ = (0,1,0,0,0,0,0,0,0,0) + (0,0,0,0,1,0,0,0,0,0).
However, Table A3.1(a) shows $a_4 = (m_1,t_5,m_5)$ and $\vec{\sigma}_5^* = \vec{\sigma}_1^* + TCV.t_5$. Then, $a_5 = (m_2,t_4,m_5)$ is put in *ATF* and atf_5 is true.

There are two firing count vectors $\vec{\sigma}_7$ at v_7:

(0,0,1,0,0,1,0,0,1,0) = $\vec{\sigma}_5^*$ + $TCV.t_8$ = (0,0,1,0,0,1,0,0,0,0) + (0,0,0,0,0,0,0,0,1,0) and

(0,0,1,0,0,0,1,0,0,1) = $\vec{\sigma}_4^*$ + $TCV.t_9$ = (0,0,1,0,0,0,1,0,0,0) + (0,0,0,0,0,0,0,0,0,1).

However, Table A3.1(a) shows $a_8 = (m_5,t_8,m_7)$ and $\vec{\sigma}_7^* = \vec{\sigma}_5^* + TCV.t_8$. Then, $a_7 = (m_4,t_9,m_7)$ is put in *ATF* and atf_7 is true.

Obviously, the set *ATF* of Figure 3.5 is $\{a_5, a_7\}$.

Definition 4.4 (state space mapping function for addTtoParc SSM_E)
Let a Petri net $PN = (P, T, F, m_0)$ have the O-graph $OG = (V, A)$, and M be the marking set. Assume that *PN* is changed into another Petri net *PN'* by applying the basic editing action $\boldsymbol{E}$ = *addTtoParc* as described in section 3.1 according to a *TPR* relation *tpr*. *M*' is the set of all markings of *PN'*. The *state space mapping function* $\boldsymbol{SSM_E}$ is a function defined from M into M', and $tpr = (t_{src}, p_{dst})$:

$\forall m_i \in M$: $\boldsymbol{SSM_E}(m_i) = m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst}$

SSM_E is a function which maps the reachable markings of *PN* to the reachable markings of *PN'*. For example, while using marking m_1=(0,0,0,1,0,0,0,0,0) in Table A3.1(a) as input, SSM_E outputs marking m_1' = (0,0,0,1,**1**,0,0,0,0) = $m_1 + \vec{\sigma}_1^*[t_2] * PCV.p_4$ = (0,0,0,1,0,0,0,0,0) + 1 * (0,0,0,0,**1**,0,0,0,0), as shown in Table A3.2(a).

Definition 4.5 (arc mapping function for addTtoParc R_E)
Let E = *addTtoParc* be defined as in Definition 4.4, and *M*' be the set of all markings of *PN'*. The *arc mapping function* $\boldsymbol{R_E}$ is a function defined from A into $M' \times T' \times M'$,
and $tpr = (t_{src}, p_{dst})$:

$\forall a_k = (m_i, t_n, m_j) \in A$: $\boldsymbol{R_E}(a_k) = (SSM_E(m_i), t_n, SSM_E(m_j))$;

For example, both marking m_6 and m_8 can enable the firing of transition t_7, i.e. $a_{10} = (m_6, t_7, m_9)$ and $a_{11} = (m_8, t_7, m_{10})$ in Table A3.1(b). After *arc mapping function* R_E is applied with inputs a_{10} and a_{11}, $R_E(a_{10}) = (m_6',t_7,m_9')$ and $R_E(a_{11}) = (m_8',t_7,m_{10}')$, as shown in Table A3.2(b).

Definition 4.6 (temporal occurrence graph for addTtoParc $\underline{OG}_E$)
Let E = *addTtoParc* be defined as in Definition 4.4, and $\underline{M}$ be the set of all markings generated from SSM_E. The *temporal occurrence graph* $\underline{OG}_E$ is the directed graph $\underline{OG}_E = (\underline{V}, \underline{A})$ with $tpr = (t_{src}, p_{dst})$, where
(1) $\forall\ v_i \in V, \underline{v}_i \in \underline{V}: \underline{m}_i = SSM_E(m_i)$
(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M}\times T\times\underline{M} \mid \underline{m}_i [\ t_n > \underline{m}_j \}$
(3) $\forall a_k \in A\backslash ATF, \underline{a}_k \in \underline{A}: \underline{a}_k = R_E(a_k)$

For example, Figure 4.1 is the $\underline{OG}_E$ after SSM_E and R_E are applied with input OG shown in Figure 3.5.

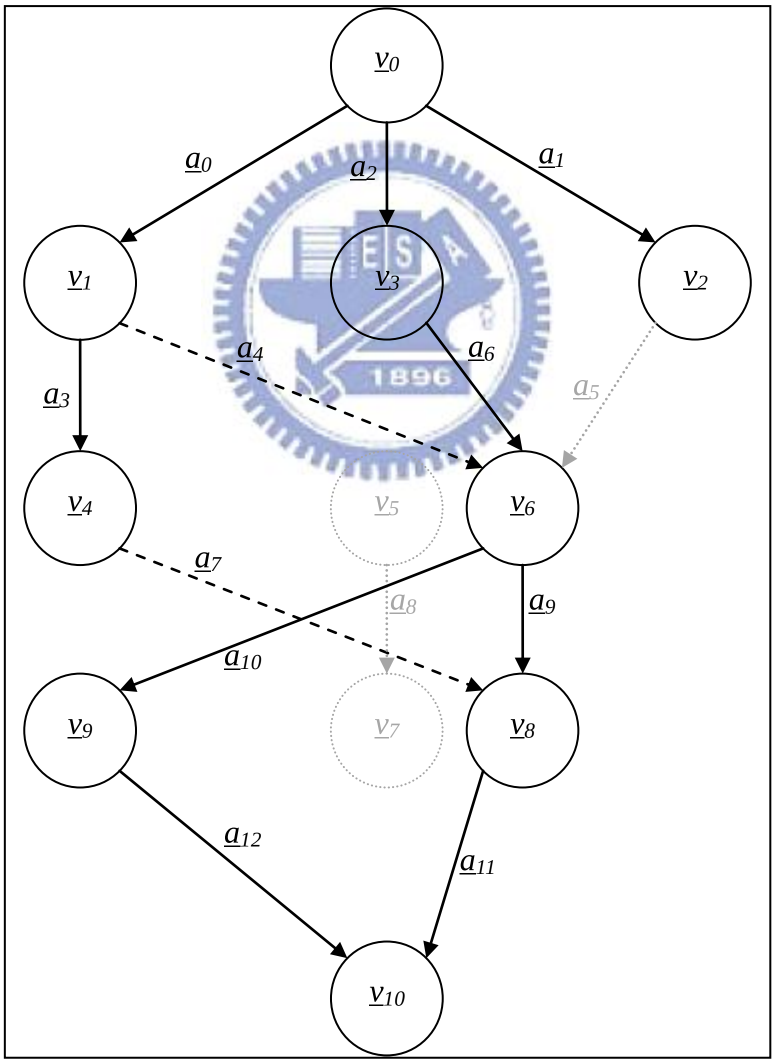


Figure 4.1 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.5.

According to Definition 4.6, the following properties ensure that OG' can be constructed by extending $\underline{OG}_E$.

Proposition 4.1

Let E = *addTtoParc* be defined as in Definition 4.4, C be the incidence matrix of PN, C' be the incidence matrix of PN', and a *temporal occurrence graph* $\underline{OG}_E$ be defined as in Definition 4.6. Then,

(1) $v_0 = v_0'$,

(2) $\underline{V} \subseteq V'$, and

(3) $\underline{A} \subseteq A'$.

Proof:

(1) Since the initial markings of PN and PN' are the same, $v_0 = v_0'$.

(2) For each node $\underline{v}_i$ in $\underline{V}$,

$$\underline{m}_i = m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst}.$$

Since the firing sequence of $\vec{\sigma}_i^*$ can be fired in PN', there is a marking

$$m_i' = m_0' + C' * \vec{\sigma}_i^*.$$

Because there might be some token(s) generated from the arc (t_{src}, p_{dst}), the difference between $m_i[p_{dst}]$ and $m_i'[p_{dst}]$ is $\vec{\sigma}_i^*[t_{src}]$. Besides, the rest of m_i are equivalent to those of m_i', i.e.

$$m_i' = m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst}.$$

According to Definition 4.4,

$$\underline{m}_i = SSM(m_i) \text{ is } m_i'.$$

From Definition 4.6 and $\underline{\vec{\sigma}}_i^* = \vec{\sigma}_i^{*'}$, $v_i = v_i'$.

Thus, and $\underline{V}$ is the subset of V'.

(3) For each arc $\underline{a}_k = (\underline{m}_i, t_n, \underline{m}_j)$ in $\underline{A}$, there is a node $v_i' \in V'$ and a marking $m_i' = \underline{m}_i$ based on the discussions in (2). Since $m_i'[p_{dst}] \geqq m_i[p_{dst}]$ and the token numbers in the rest places do not change, transition t_n enabled at m_i can also be fired at m_i'. From Definition

4.6(1), another marking $m_j \in M$ can be modified as

$$\underline{m}_j = m_j + \vec{\sigma}_j^*[t_{src}] * PCV.p_{dst}.$$

From Definition 2.4,

$$m_j = m_i + C * TCV.t_n.$$

From Definition 4.2,

$$\vec{\sigma}_j^* = \vec{\sigma}_i^* + TCV.t_n.$$

Therefore, $\underline{m}_j = m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst} + C * TCV.t_n + TCV.t_n[t_{src}] * PCV.p_{dst}$. Because $TCV.t_n[t_{src}] * PCV.p_{dst}$ is the number of new tokens in the place p_{dst},

$C' * TCV.t_n = C * TCV.t_n + TCV.t_n[t_{src}] * PCV.p_{dst}$.

Thus, there exists an arc $a_k' = (m_i', t_n, m_j') \in A'$ and $\underline{m}_j = m_i' + C' * TCV.t_n = m_j'$.

Hence, $\underline{a}_k = a_k'$ and $\underline{A}$ is the subset of A'.

Proposition 4.2

Let $E = addTtoParc$ be defined as in Proposition 4.1. Then,

$\forall\ a_k = (m_i, t_n, m_j) \in A$:

(1) if $a_k \in ATF$ and $(\vec{\sigma}_i^* + TCV.t_n)[t_{src}] = \vec{\sigma}_j^*[t_{src}]$,

then $R_E(a_k) \in A'$.

(2) if $a_k \in ATF$ and $(\vec{\sigma}_i^* + TCV.t_n)[t_{src}] \neq \vec{\sigma}_j^*[t_{src}]$,

then $R_E(a_k) \notin A'$ and

$(SSM_E(m_i), t_n, m_j + (\vec{\sigma}_i^* + TCV.t_n)[t_{src}] * PCV.p_{dst}) \in A'$.

(3) if $a_k \notin ATF$, then $R_E(a_k) \in A'$.

Proof:

(1) From Definition 4.6(1), $\underline{m}_j = m_j + \vec{\sigma}_j^*[t_{src}] * PCV.p_{dst}$.

From Definition 2.4 and $(\vec{\sigma}_i^* + TCV.t_n)[t_{src}] = \vec{\sigma}_j^*[t_{src}]$, $m_j + \vec{\sigma}_j^*[t_{src}] * PCV.p_{dst} = m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst} + C * TCV.t_n + TCV.t_n\ [t_{src}] * PCV.p_{dst}$.

From Proposition 4.1(2), there is a marking $m_i' = SSM_E(m_i)$.

Thus, $m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst} = m_i'$. $TCV.t_n\ [t_{src}] * PCV.p_{dst}$ represents the tokens in place

p_{dst} generated from arc (t_{src}, p_{dst}) after firing t_n. The difference between $C * TCV.t_n$ and $C' * TCV.t_n$ is the tokens generated from arc (t_{src}, p_{dst}).

Hence, $m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst} + C * TCV.t_n + TCV.t_n\,[t_{src}] * PCV.p_{dst} = m_i' + C' * TCV.t_n$. Because $m_i' + C' * TCV.t_n = m_j'$, $R_E(a_k) = (\ SSM_E(m_i)\ ,\ t_n\ ,\ SSM_E(m_j)\ ) = (\underline{m}_i, t_n, \underline{m}_j) = (m_i', t_n, m_j') \in A'$.

(2) From Proposition 4.1(2), there is a marking $m_i' = SSM_E(m_i)$. Since m_i enable t_n, m_i' enable t_n. $m_j + (\vec{\sigma}_i^* + TCV.t_n)[t_{src}] * PCV.p_{dst} = m_i + C * TCV.t_n + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst} + TCV.t_n[t_{src}] * PCV.p_{dst} = m_i' + C' * TCV.t_n = m_j'$.

Because $SSM_E(m_j) = m_j + \vec{\sigma}_j^*[t_{src}] * PCV.p_{dst}$ and $\vec{\sigma}_i^* + TCV.t_n)[t_{src}] \neq \vec{\sigma}_j^*[t_{src}]$, $R_E(a_k) \neq m_j + (\vec{\sigma}_i^* + TCV.t_n)[t_{src}] * PCV.p_{dst}$. Thus, $R_E(a_k) \notin A'$.

(3) From Definition 4.6 (3) and $a_k \in A \backslash ATF$, $R_E(a_k) \in \underline{A}$. From Proposition 4.1(3), $R_E(a_k) \in A'$.

We use the addTtoParc example in section 3.2 to explain Proposition 4.2. In the example:

(1) $a_7 \in ATF$ and $(\vec{\sigma}_4^* + TCV.t_9)[t_2] = \vec{\sigma}_7^*[t_2]$ cause $R_E(a_7) = a_7' \in A'$.

(2) $a_5 \in ATF$ and $(\vec{\sigma}_2^* + TCV.t_4)[t_2] \neq \vec{\sigma}_5^*[t_2]$ cause $R_E(a_5) = (m_2', t_4, m_5') \notin A'$. Also, $m_{12}' = m_5' + (\vec{\sigma}_2^* + TCV.t_4)[t_2] * PCV.p_4 = (0,0,0,0,0,1,0,0,0) + 0*(0,0,0,0,1,0,0,0,0)$ is constructed by $m_j + (\vec{\sigma}_i^* + TCV.t_n)[t_{src}] * PCV.p_{dst}$. Thus, $(m_2', t_4, m_{12}') = a_{14}' \in A'$.

(3) Because $a_3 \notin ATF$, $R_E(a_3) = a_3' \in A'$.

Proposition 4.2(2) detects the arcs which do not belong to A'. Proposition 4.2 certifies that the detections are complete.

Definition 4.7 (must re-generated marking set for addTtoParc MRG_E)
For a *temporal occurrence graph* $\underline{OG}_E$ and $\underline{M}$, the set of all markings in $\underline{OG}_E$. $\underline{OG}_E$ might have a *must re-generated marking set* $\boldsymbol{MRG_E} \subseteq \underline{M}$, $tpr = (t_{src}, p_{dst})$:

$$MRG_E = \{\ \underline{m}_i \in \underline{M} |\ \vec{\sigma}_i^*[t_{src}] > 0\ \}.$$

Since $\vec{\sigma}_1^*[t_2]$=1, $\vec{\sigma}_4^*[t_2]$=1, $\vec{\sigma}_5^*[t_2]$=1, and $\vec{\sigma}_7^*[t_2]$=1, MRG_E in Figure 4.1 is {$\underline{m_1}$, $\underline{m_4}$, $\underline{m_5}$, $\underline{m_7}$}.

Proposition 4.3

Let E = *addTtoParc* be defined as in Proposition 4.1. Then, ∀ a_k'=(m_i', t_n,m_j') ∈ A' :

(1) if m_i' ∉ $\underline{M}$, then a_k'∉ $\underline{A}$.

(2) if m_i' ∈ MRG_E and ∄ (m_i, t_n, m_j) ∈ $A \backslash ATF$, then a_k'∉ $\underline{A}$.

(3) if m_i' ∈ MRG_E and ∃ (m_i, t_n, m_j) ∈ $A \backslash ATF$, then a_k'∈$\underline{A}$.

(4) if m_i' ∈ $\underline{M}$ and m_i'∉ MRG_E, then a_k'∈$\underline{A}$ or ∃ (m_i, t_n, m_j) ∈ ATF.

Proof:

(1) If m_i' does not exist in $\underline{OG_E}$, the arcs starting from it to m_j' can not occur in $\underline{OG_E}$.

(2) Because ∄ (m_i, t_n, m_j) ∈ $A \backslash ATF$ and Definition 4.6(3), ∄ ($\underline{m_i}$, t_n, $\underline{m_j}$) ∈ $\underline{A}$. Thus, a_k'∉ $\underline{A}$.

(3) Since ∃ (m_i, t_n, m_j) ∈ $A \backslash ATF$ and Definition 4.6(3), ∃ ($\underline{m_i}$, t_n, $\underline{m_j}$) ∈ $\underline{A}$. From Proposition 4.1(3), a_k'∈$\underline{A}$.

(4) Because m_i' ∉ MRG_E, $\vec{\sigma}_i^*[t_{src}]$=0. Hence, m_i can enable all transitions which m_i' can enable. Since m_i can enable all transitions which m_i' can enable, each arc from m_i' belongs to $\underline{A}$ or ∃ (m_i, t_n, m_j) ∈ ATF.

We use the addTtoParc example in section 3.2 to explain Proposition 4.3. In the example:

(1) Because a_{16}'=(m_{11}',t_5,m_9') and m_{11}'∉ $\underline{M}$, a_{16}'∉ $\underline{A}$.

(2) m_1'∈ MRG_E, a_3=(m_1,t_6,m_4), and a_4=(m_1,t_5,m_5) cause a_{13}'=(m_1',t_7,m_{11}')∉ $\underline{A}$.

(3) m_1'∈ MRG_E and a_3=(m_1,t_6,m_4) cause a_3'=(m_1',t_6,m_4')∈ $\underline{A}$.

(4) Since m_3'∈ $\underline{M}$ and m_3' ∉ $MRG_E \backslash MS$, a_6'=(m_3',t_3,m_6') ∈ $\underline{A}$.

Proposition 4.3(1) and (2) detect the arcs which do not belong to $\underline{A}$. Proposition 4.3

certifies that the detections are complete.

Proposition 4.4

Let E = *addTtoParc* be defined as in Proposition 4.1. Then,

(1) $\forall v_j' \in V'$: if $\nexists\ (m_i', t_n, m_j') \in \underline{A}$ and $v_j' \neq v_0'$,

then $v_j' \notin \underline{V}$.

(2) $v_0' \in \underline{V}$

Proof:

(1) Since there is no arc pointing to the vertex v_j' in $\underline{OG_E}$, such marking do not exist in $\underline{OG_E}$ but the initial vertex.

(2) Since $m_0' = \underline{m_0}$ and $\vec{\sigma}_0^{*\prime} = \underline{\vec{\sigma}_0^*}$, $v_0' = \underline{v_0}$ belongs to $\underline{V}$.

We use the addTtoParc example in section 3.2 to explain Proposition 4.4. In the example:

(1) Because $v_{11}' \neq v_0'$ and $a_{13}' = (m_1', t_7, m_{11}') \notin \underline{A}$, $v_{11}' \notin \underline{V}$.

Proposition 4.5

Let E = *addTtoParc* be defined as in Proposition 4.1.

$\forall\ m_x, m_y \in M$: if $SSM_E(m_x) = SSM_E(m_y)$ and $\vec{\sigma}_y^*[t_{src}] > \vec{\sigma}_x^*[t_{src}]$, then $\forall\ (m_x, t_n, m_{xj}) \in A$, $\exists\ (m_y, t_n, m_{yj}) \in A$.

Proof:

Because $SSM_E(m_y) = m_y + \vec{\sigma}_y^*[t_{src}] * PCV.p_{dst}$, $SSM_E[m_x] = m_x + \vec{\sigma}_x^*[t_{src}] * PCV.p_{dst}$, $\vec{\sigma}_y^*[t_{src}] > \vec{\sigma}_x^*[t_{src}]$, and $SSM_E(m_y) = SSM_E(m_x)$, $m_y[p_{dst}] < m_x[p_{dst}]$ and the rest of m_y and m_x are equivalent. Thus, m_x can enable the transitions which m_y enables.

We use the addTtoParc example in section 3.2 to explain Proposition 4.5. In the example, $SSM_E(m_6) = SSM_E(m_5)$ and $\vec{\sigma}_5^*[t_2] > \vec{\sigma}_6^*[t_2]$. Both $a_8 = (m_5, t_8, m_7)$ and $a_9 = (m_6, t_8, m_8)$ are constructed by firing t_8.

4.1.1.2 Updating an Occurrence graph

Let a Petri net be $PN = (P, T, F, m_0)$, the O-graph $OG = (V, A)$ belong to PN, and M be the set of all markings of P. The following algorithm presents a method to update the corresponding O-graph OG' when a *TPR* relation is added into a Petri net. Each arc $a_i \in A$ has the flag atf_i = {true, false}. The flag atf_i = true means $a_i \in ATF$ (Definition 4.3), whereas $a_i \notin ATF$. Assume that PN is transformed into another Petri net PN' by applying basic editing action $E = addTtoParc$ according to $tpr=(t_{src}, p_{dst})$, i.e. $addTtoParc\ (PN, t_{src}, p_{dst})$. Algorithm 4.1 firstly modifies OG for $\underline{OG}_E$ as described in Definition 4.6, and then extends $\underline{OG}_E$ for the O-graph $OG' = (V', A')$ which belongs to PN'. $\underline{M}$ is the set of all markings in $\underline{OG}_E$ and M' is the set of all markings of P'.

Algorithm 4.1 ***Main_ATP* (*PN, OG, tpr*) → *PN'* and *OG'***

```
// Input     PN: the original net
//           OG: the O-graph of PN
//           tpr: a TPR relation
// Output    PN': the refined net with tpr
//           OG': the O-graph of PN'
● M_nr ← ∅ : a set of markings. M_nr ⊆ M' \ M̲. M_nr contains those markings for which
we have not yet found the successors.
● MRG_E ← ∅ : a set of markings as in Definition 4.7.
● O̲G̲_E : a temporal occurrence graph. V̲ ← ∅ and A̲ ← ∅ .
● OG' : an occurrence graph. V' ← ∅ and A' ← ∅ .

1    add an arc from t_src to p_dst     // construct PN'
2    ModifyOG_ATP                       // modify OG for O̲G̲_E
3    Findmarking
4    ExtendOG                           // extend O̲G̲_E for OG'
```

Line 1 in Algorithm 4.1 constructs the refined Petri net *PN'*. Line 2 calls the function *ModifyOG_ATP* (Algorithm 4.2) to modify the state space in *OG* and redirect the flow relations in *ATF*. For example, the redirection changes from a_5 to a'_{14} and generates node v'_{12}. Then, v'_{12} is added to M_{nr}. The function *Findmarking* (Algorithm 4.4) in line 3 can generate the new node $v'_j \in V' \setminus \underline{V}$ and arc $a'_k \in A' \setminus \underline{A}$ according to MRG_E. For example, the new nodes in Figure 3.7 are v'_{11} and v'_{13}. The new arcs are a'_{13} and a'_{15}. The new nodes generated by *Findmarking* are put in M_{nr}. Line 4 calls the function *ExtendOG* (Algorithm 4.5) to generate the rest of new nodes and arcs whose ancestors are in M_{nr}, e.g., the node v'_{14} and the arcs a'_{16}, a'_{17}, a'_{18}, and a'_{19} in Figure 3.7. The details of each step will be presented below.

Algorithm 4.2 ***ModifyOG_ATP***

```
1   $\underline{ATF} \leftarrow ATF$
2   $\underline{A} \leftarrow A$
3   for all $v_i \in TmapV(t_{src})$ do
4   begin
5       $NewV(\underline{v_i}, v_i)$
6       $\underline{m_i}[p_{dst}] \leftarrow \underline{m_i}[p_{dst}] + \vec{\sigma}^*_i[t_{src}]$
7       // $SSM_E(m_i)$ in Definition 4.4
8       if($\exists \underline{m_j} \in \underline{M}$ and $\underline{m_i} = \underline{m_j}$)
9           if($\vec{\sigma}^*_i[t_{src}] > \vec{\sigma}^*_j[t_{src}]$)
10              $Merge(\underline{v_i}, \underline{v_j})$
11          else
12              $Merge(\underline{v_j}, \underline{v_i})$
13          endif
14      else                // $\underline{m_i} \in MRG_E$
15          $MRG_E \leftarrow MRG_E \cup \{\underline{m_i}\}$
16      endif
17  endfor
18
```

```
19  for all $\underline{a}_k = (\underline{m}_i, t_n, \underline{m}_j) \in \underline{ATF}$ do
20  begin
21      if( $(\underline{\vec{\sigma}}_i^*+TCV.t_n)[t_{src}] \neq \underline{\vec{\sigma}}_j^*[t_{src}]$ )
22          $\underline{A} \leftarrow \underline{A} \setminus \{ \underline{a}_k \}$
23          $\underline{ATF} \leftarrow \underline{ATF} \setminus \{ \underline{a}_k \}$
24          $m_j'$ is the marking produced after $\underline{m}_i$ fires $t_n$
25          $v_j' \leftarrow Node\ (\underline{v}_i, t_n, m_j')$
26          $V' \leftarrow V' \cup \{v_j'\}$
27          $a_l' \leftarrow Arc\ (\underline{m}_i, t_n, m_j')$
28          $A' \leftarrow A' \cup \{a_l'\}$
29      endif
30  endfor
```

In Algorithm 4.2, the loop from line 3 to 17 selects a node v_i in *TmapV* (t_{src}). In line 3, *TmapV* (t_n) is a function that maps a transition t_n to a set of vertexes. $\forall\ v_i \in TmapV(t_n)$: $\vec{\sigma}_i^*[t_n]>0$. In line 5, ***NewV*** **($\underline{v}_i$,v_i)** is a function. The function creates a new vertex $\underline{v}_i$, sets its datum as the following, and adds $\underline{v}_i$ to $\underline{V}$.

- $\underline{m}_i \leftarrow m_i$
- $\underline{\vec{\sigma}}_i^* \leftarrow \vec{\sigma}_i^*$

In each turn, line 6 modifies the marking of v_i as described in Definition 4.4. Some of the markings after modification would be duplicated and codes from line 9 to 13 deal with this situation by calling *Merge* as in Algorithm 4.3 with corresponding parameters. After *Merge*, the node represented by the first parameter and its connected arc(s) are deleted. Because the parameters of *Merge* in line 10 and 12 are different, the node to be deleted is $\underline{v}_i$ or $\underline{v}_j$ respectively. Otherwise, $\underline{m}_i$ is added to MRG_E in line 15. The loop from line 19 to 30 deletes the arcs in $\underline{ATF}$ according to “if condition” in line 21. In line 25, ***Node*** **(v_i', t_n, m’)** is a function. If there is already a node v_j' whose marking is equal to m’, the function has no effect and returns address v_j'. Otherwise, the function adds m’ to M_{nr}, creates a new vertex v_j', sets its datum as the following, and returns its address.

- $m_j' \leftarrow m'$

- $\vec{\sigma}_j^{*'} \leftarrow \vec{\sigma}_i^{*'} + TCV.t_n$

In line 27, ***Arc*** **($\boldsymbol{m_i'}$, $\boldsymbol{t_n}$, $\boldsymbol{m_j'}$)** is a function that creates a new arc $a_k' = (m_i', t_n, m_j')$ in A', sets its datum as the following, and returns its address.

- $atf_k' \leftarrow$ true and $ATF' \leftarrow ATF' \cup \{a_k'\}$, if $\vec{\sigma}_j^{*'} \neq \vec{\sigma}_i^{*'} + TCV.t_n$
- $atf_k' \leftarrow$ false , otherwise

Then, line 25 generates the node with $\underline{v}_i$, t_n, and m_j'. Line 27 generates the arc based on $\underline{m}_i$ [t_n > m_j'.

For example, when the loop from line 3 to 17 selects v_1, line 6 modifies $\underline{m}_1$ as $\underline{m}_1$ + $\underline{\vec{\sigma}}_1^*[t_2]$ * $PCV.p_4$. Assume that $\underline{m}_5$ belongs to $\underline{M}$. When v_6 is selected, the marking $\underline{m}_5$ and $\underline{m}_6$ in Figure 4.1 are both (0,0,0,0,1,1,0,0,0) and $\underline{\vec{\sigma}}_5^*[t_2] > \underline{\vec{\sigma}}_6^*[t_2]$ which satisfy the conditions in line 8 and 9. Hence, *Merge*($\underline{v}_5$, $\underline{v}_6$) is called in line 10. When the loop from line 19 to 30 selects $\underline{a}_5$, $(\underline{\vec{\sigma}}_2^* + TCV.t_4)[t_2] \neq \underline{\vec{\sigma}}_5^*[t_2]$ satisfy the conditions in line 21. Thus, arc $\underline{a}_5$ in Figure 4.1 is deleted. v_{12}' and a_{14}' in Figure 3.7 are generated based on $\underline{m}_2$ [t_4 > m_{12}'.

```
Algorithm 4.3 Merge ($\underline{v}_x$, $\underline{v}_y$)
// Input        $\underline{v}_x$: a vertex after $SSM_E$
//              $\underline{V}_y$: a vertex after $SSM_E$

1    for all $\underline{a}_k = (\underline{m}_x, t_n, \underline{m}_{xj}) \in \underline{A}$          //outgoing arcs of $\underline{v}_x$
2        if($\underline{atf}_k$ = true)
3            $\underline{ATF} \leftarrow \underline{ATF} \setminus \{ \underline{a}_k \}$
4        endif
5        $\underline{A} \leftarrow \underline{A} \setminus \{ \underline{a}_k \}$
6    endfor
7
8    for all $\underline{a}_k = (m_{xi}, t_n, \underline{m}_x) \in \underline{A}$          //ingoing arcs of $\underline{v}_x$
9        change $\underline{a}_k$ from $(\underline{m}_{xi}, t_n, \underline{m}_x)$ to $(\underline{m}_{xi}, t_n, \underline{m}_y)$
10       if($\underline{\vec{\sigma}}_{xi}^* + TCV.t_n \neq \underline{\vec{\sigma}}_y^*$)
11           $\underline{atf}_k \leftarrow$ true
12           $\underline{ATF} \leftarrow \underline{ATF} \cup \{ \underline{a}_k \}$
13       endif
```

```
14  endfor
15  $\underline{V} \leftarrow \underline{V} \setminus \{ \underline{v}_x \}$
```

In Algorithm 4.3, the loop from line 1 to 6 deletes each outgoing arc of $\underline{v}_x$. The loop from line 8 to 14 redirects the ingoing arcs of $\underline{v}_x$. Line 15 merges node $\underline{v}_x$ and $\underline{v}_y$ by deleting $\underline{v}_x$ in $\underline{V}$.

For example, when $\underline{v}_x$=$\underline{v}_5$ and $\underline{v}_y$=$\underline{v}_6$, arc $\underline{a}_8$ = ($\underline{m}_5$, t_8, $\underline{m}_7$) in Figure 4.1 is deleted. When $\underline{v}_x$=$\underline{v}_7$ and $\underline{v}_y$=$\underline{v}_8$, arc a_7 is redirected from ($\underline{m}_4$, t_9, $\underline{m}_7$) to ($\underline{m}_4$, t_9, $\underline{m}_8$). $\underline{v}_5$ and $\underline{v}_6$ are merged by deleting $\underline{v}_5$.

Proposition 4.6

Assume that the markings of $\underline{v}_x$ and $\underline{v}_y$ are equivalent and $\vec{\underline{\sigma}}^*_x[t_{src}] > \vec{\underline{\sigma}}^*_y[t_{src}]$. Algorithm 4.3 merges $\underline{v}_x$ and $\underline{v}_y$ correctly, i.e., the arcs connected with $\underline{v}_x$ are redirected to $\underline{v}_y$ and $\underline{v}_x$ is deleted in $\underline{V}$

Proof:

The loop from line 1 to 6 deletes each outgoing arc of $\underline{v}_x$ and Proposition 4.5 holds that the redirected arcs of the outgoing arcs of $\underline{v}_x$ belong to $\underline{A}$. The loop from line 8 to 14 redirects the ingoing arcs of $\underline{v}_x$. Line 15 merges node $\underline{v}_x$ and $\underline{v}_y$ by deleting $\underline{v}_x$ in $\underline{V}$. With above statements, the correctness of Algorithm 4.3 holds.

Proposition 4.7

Assume that the arc from t_{src} to p_{dst} is added into *PN* and Algorithm 4.3 merges $\underline{v}_x$ and $\underline{v}_y$ correctly. Algorithm 4.2 modifies *OG* as $\underline{OG}_E$, generates MRG_E, and redirects the arcs in $\underline{ATF}$ correctly, i.e., $\underline{OG}_E$ as defined in Definition 4.6, MRG_E as defined in Definition 4.7, and the arcs after redirections belong to *OG'*.

Proof:

The temporal occurrence graph defined in Definition 4.6 is calculated from line 3 to 17. For each node $v_i \in V$, the conditions in line 8 and 9 equals to those in Proposition 4.5 and

Algorithm 4.3 merges $\underline{v}_x$ and $\underline{v}_y$ correctly. Otherwise, $\underline{m}_i$ is added to MRG_E. Line 19 to 30 deletes and generates the arcs according to the condition in Proposition 4.2(2). With above statements, the correctness of Algorithm 4.2 holds.

```
Algorithm 4.4 Findmarking

1   for all $\underline{m}_i \in MRG_E$
2       for (each transition $t_n$ can be enabled at $\underline{m}_i$ and $\nexists(m_i, t_n, m_j) \in A$)
3           $m_j'$ is the marking produced after $\underline{m}_i$ fires $t_n$
4           $v_j' \leftarrow Node(\underline{v}_i, t_n, m_j')$
5           $V' \leftarrow V' \cup \{v_j'\}$
6           $a_k' \leftarrow Arc(\underline{m}_i, t_n, m_j')$
7           $A' \leftarrow A' \cup \{a_k'\}$
8       endfor
9   endfor
```

In Algorithm 4.4, for each marking $\underline{m}_i$ in MRG_E, the loop from line 2 to 8 generates the node v_j' and the arc a_k'=($\underline{m}_i$, t_n, m_j') based on $\underline{m}_i$ [t_n > m_j'. Line 4 adds the new marking m_j' to M_{nr} if $m_j' \notin \underline{V}$.

For example, when the loop from line 1 to 9 selects $\underline{m}_1$ and the loop from line 2 to 8 selects t_7, v_{11}' is generated in line 4 and a_{13}' is generated in line 6.

Proposition 4.8

Given PN and PN' as in Definition 4.1 and MRG_E generated by Algorithm 4.2 correctly. Algorithm 4.4 generates the nodes and arcs from MRG_E correctly, i.e., the nodes and arcs generated by Algorithm 4.4 all belong to OG' but not $\underline{OG}_E$.

Proof:

For each marking $\underline{m}_i \in MRG_E$, the conditions in line 2 equals to those in Proposition 4.3 (2) and codes from line 4 to 7 generate the nodes and arcs for OG'. Since $\nexists(m_i, t_n, m_j) \in A$, the arc generated by Algorithm 4.4 do not belong to $\underline{OG_E}$. Thus, the correctness of Algorithm 4.4 holds.

Algorithm 4.5 ***ExtendOG***

```
1   $V' \leftarrow V' \cup \underline{V}$
2   $A' \leftarrow A' \cup \underline{A}$
3   $ATF' \leftarrow ATF' \cup \underline{ATF}$
4   repeat
5       select a markings $m'_i \in M_{nr}$
6       for (each transition $t_n$ enabled at $m'_i$ )
7           $m'_j$ is the marking produced after firing $t_n$
8           $v'_j \leftarrow Node(v'_i, t_n, m'_j)$
9           $V' \leftarrow V' \cup \{v'_j\}$
10          $a'_k \leftarrow Arc(m'_i, t_n, m'_j)$
11          $A' \leftarrow A' \cup \{a'_k\}$
12      endfor
13      $M_{nr} \leftarrow M_{nr} \setminus \{m'_i\}$
14  until $M_{nr} = §$
```

In Algorithm 4.5, for each m'_i in M_{nr}, the loop from line 4 to 12 generates the arcs $a'_k \in A'$ and nodes $v'_j \in V'$ from v'_i. The nodes generated by function *Node* are added to M_{nr}.

For example, when the loop from line 4 to 12 selects $\underline{m_{11}}$, the node generated from v'_{11} is v'_{13} and the arcs generated from v'_{11} are a'_{16} and a'_{17}, as shown in Figure 3.7.

Proposition 4.9

Assume that M_{nr} is generated by Algorithm 4.2 and Algorithm 4.4 correctly. Algorithm 4.5 generates the nodes and arcs whose ancestors are in M_{nr} correctly, i.e., the nodes and arcs generated by Algorithm 4.5 all belong to *OG'* and the nodes and arcs belonging to *OG'* but not generated by Algorithm 4.4 are all generated by Algorithm 4.5.

Proof:

For each marking $m_i' \in M_{nr}$, the loop from line 4 to 14 extends the graph from M_{nr} which has the properties discussed in Proposition 4.3(1) and Proposition 4.4(1). Codes from line 8 to 11 generate the nodes and arcs for *OG'*. Proposition 4.3 certifies that the detections corresponding to Proposition 4.3(1) are complete. Hence, the correctness of Algorithm 4.5 holds.

Theorem 4.1 (Correct O-graph Modification and Extension)
Algorithm 4.1 generates *OG'* correctly when adding a TtoP arc, i.e. *OG'* has the same characteristic as the one constructed from *PN'* directly.

Proof:

From Proposition 4.7, Algorithm 4.2 modifies *OG* as $\underline{OG_E}$ and redirects the arcs in $\underline{ATF}$ correctly. From Proposition 4.1, $\underline{OG_E}$ is the subnet of *OG'*. From 4.8 and 4.9, the nodes and arcs generated by Algorithm 4.4 and 4.5 all belong to *OG'* and the nodes and arcs belonging to *OG'* but not $\underline{OG_E}$ are all generated by Algorithm 4.4 and 4.5. Hence, the correctness of Algorithm 4.1 holds.

4.1.1.3 The time complexity

Let *t* be the number of transitions in the Petri net, *n* be the number of vertexes in the input O-graph, and *n'* be the number of vertexes in the result O-graph. In algorithm 4.2, it is assumed that all markings in the O-graph of the original net should be modified one time. Thus, the loop from line 3 to 17 will run *n* times. Line 8 checks if $\underline{m_i}$ is already in the temporal occurrence graph and the complexity of the search is $O(1)$. Because the loop from line 19 to 30 will run *n* times, the complexity of Algorithm 4.2 is $O(n)$. In algorithm 4.5, each node contained in the result O-graph but not contained in $\underline{OG_E}$ should be added into M_{nr}. The loop from line 4 to 14 selects a marking in M_{nr} and generates the arcs and nodes correspondingly. Hence, this loop repeats $|n' - n|$ times. Since each transition in the result net must be examined whether it is enabled at m_i' or not, the complexity of the examination is $O(t)$. The procedure *Node* (v_i', t_n, m_j') checks if m_j' is already in O-graph and the complexity of the search is $O(1)$. Thus, the complexity of Algorithm 4.5 is $O(|n' - n|*t)$. The complexity of Algorithm 4.1 is $O(n + |n' - n|*t)$. Since designers usually edit the Petri nets sequentially, the complexity of Algorithm 4.1 is $O(n + t)$.

4.1.2 PtoT arc insertion

4.1.2.1 Scenario of PtoT arc insertions

Let a relation which belongs to $P \times T$ be named as a *PTR* relation. The addPtoTarc example in section 3.2 can be translated into: *PN* is transformed into *PN'* by applying *addPtoTarc* with a *PTR* relation (p_1, t_3). A formal definition of a *PTR* addition is described in Definition 4.8.

Definition 4.8 (*PTR* addition)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by adding a *PTR* relation *ptr* into *PN* has the following properties:
(1) $P' = P$;
(2) $T' = T$;
(3) $F' = F \cup \{ ptr \}$.

In a *PTR* addition, both the original net *PN* and the refined net *PN'* have the same transitions, places, and tokens in the places, but PN' has one more arc representing the *PTR* relation added. Therefore, the initial markings of *PN* and *PN'* are equivalent, i.e. $m_0 = m_0'$.

Definition 4.9 (state space mapping function for addPtoTarc SSM_E)
With Definition 4.8 and *M'*, the set of all markings of *PN'*. The *state space mapping function* $\mathbf{SSM_E}$ is a function defined from *M* into *M'*, and $ptr = (p_{src}, t_{dst})$:

$\forall m_i \in M$: $\mathbf{SSM_E}(m_i) = m_i - \vec{\sigma}_i^*[t_{dst}] * PCV.p_{src}$

SSM_E is a function which maps the reachable markings of *PN* to the reachable markings of *PN'*. For example, while using marking m_1=(1,0,0,1) in Table A3.3(a) as input, SSM_E outputs marking m_1' = (1,0,0,1) = $m_1 - \vec{\sigma}_1^*[t_3] * PCV.p_1$ = (1,0,0,1) - 0 * (0,1,0,0) as shown in Table A3.4(a).

Definition 4.10 (arc mapping function for addPtoTarc R_E)
Let *E* = *addPtoTarc* be defined as in Definition 4.9, and *M'* be the set of all markings of *PN'*. The *arc mapping function* $\mathbf{R_E}$ is a function defined from *A* into $M' \times T' \times M'$, and $ptr = (p_{src}, t_{dst})$:

$\forall a_k = (m_i, t_n, m_j) \in A$: $\mathbf{R_E}(a_k) = (SSM_E(m_i), t_n, SSM_E(m_j))$;

For example, both marking m_0 and m_1 can enable the firing of transition t_1, i.e. a_1 and a_3 in Table A3.3(b). After *arc mapping function* R_E is applied with inputs a_1 and a_3, $R_E(a_1) = (m_0', t_1, m_2')$ and $R_E(a_3) = (m_1', t_1, m_4')$, as shown in Table A3.4(b).

Definition 4.11 (temporal occurrence graph for addPtoTarc $\underline{OG}_E$)
Let *E* = *addPtoTarc* be defined as in Definition 4.9, and $\underline{M}$ be the set of all markings generated from SSM_E. The *temporal occurrence graph* $\underline{OG}_E$ is the directed graph $\underline{OG}_E = (\underline{V}, \underline{A})$ with $ptr = (p_{src}, t_{dst})$, where
(1) $\forall v_i \in V, \underline{v}_i \in \underline{V}$: $\underline{m}_i = SSM_E(m_i)$
(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M} \times T \times \underline{M} \mid \underline{m}_i [t_n > \underline{m}_j \}$
(3) $\forall a_k \in A \backslash ATF, \underline{a}_k \in \underline{A}$: $\underline{a}_k = R_E(a_k)$

For example, Figure 4.2 is the $\underline{OG}_E$ after SSM_E and R_E are applied with input *OG* in Figure 3.9.

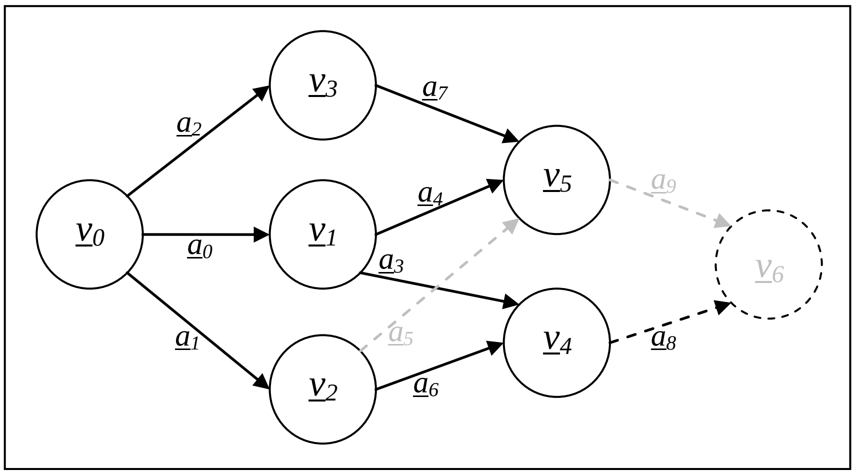


Figure 4.2 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.9.

4.1.2.2 Updating an Occurrence graph and the time complexity

Algorithm 4.6 presents a method to update the corresponding O-graph *OG'* for the insertion of a *PTR* relation. Based on Definition 4.8, Algorithm 4.6 firstly modifies *OG* for $\underline{OG}_E$ (Definition 4.11), and then extends $\underline{OG}_E$ for the O-graph *OG'* = (*V'*, *A'*).

```
Algorithm 4.6 Main_APT (PN, OG, ptr) → PN' and OG'
// Input      PN: the original net
//            OG: the O-graph of PN
//            ptr: a PTR relation
// Output     PN': the refined net with ptr
//            OG': the O-graph of PN'
● $M_{nr} \leftarrow \emptyset$ : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for which
we have not yet found the successors.
● $\underline{OG}_E$ : a temporal occurrence graph. $\underline{V} \leftarrow \emptyset$ and $\underline{A} \leftarrow \emptyset$ .
● OG' : an occurrence graph. $V' \leftarrow \emptyset$ and $A' \leftarrow \emptyset$ .

1    add an arc from $p_{src}$ to $t_{dst}$          // construct PN'
2    ModifyOG_APT                          // Algorithm 4.7
3    ExtendOG                              // Algorithm 4.5
```

Line 1 in Algorithm 4.6 constructs the refined Petri net *PN'*. Line 2 calls the function *ModifyOG_APT* (Algorithm 4.7) to modify the state space in *OG* and redirect the flow relations in *ATF*. For example, the redirection changes from a_5 to a'_{10} and generates node v'_7. Then, node v'_7 is put in M_{nr}. Line 3 calls the function *ExtendOG* (Algorithm 4.5) to generate the rest of new nodes and arcs whose ancestors are in M_{nr}. The details of each step will be presented below.

```
Algorithm 4.7   ModifyOG_APT

1   for all $v_i \in PTmapV(p_{src}, t_{dst})$ do
2   begin
3       $NewVA(\underline{v}_i, v_i)$
4       $\underline{m}_i[p_{src}] \leftarrow \underline{m}_i[p_{src}] - \vec{\sigma}_i^*[t_{dst}]$
5       // $SSM_E(m_i)$ in Definition 4.9
6       if($\exists \underline{m}_j \in \underline{M}$ where $\underline{m}_i = \underline{m}_j$)
7           $Merge(v_i, v_j)$      //Algorithm 4.3
8       endif
9   endfor
10
11  for all $\underline{a}_k = (\underline{m}_i, t_n, \underline{m}_j) \in \underline{ATF}$ do
12  begin
13      $\underline{A} \leftarrow \underline{A} \setminus \{ \underline{a}_k \}$
14      $\underline{ATF} \leftarrow \underline{ATF} \setminus \{ \underline{a}_k \}$
15      if($t_n$ can be enabled at $\underline{m}_i$)
16          $m'_j$ is the marking produced after $\underline{m}_i$ fires $t_n$
17          $v'_j \leftarrow Node(\underline{v}_i, t_n, m'_j)$
18          $V' \leftarrow V' \cup \{v'_j\}$
19          $a'_l \leftarrow Arc(\underline{m}_i, t_n, m'_j)$
20          $A' \leftarrow A' \cup \{a'_l\}$
21      endif
22  endfor
```

In line 1, ***PTmapV* (*p_m*, *t_n*)** is a function that maps place p_m and transition t_n to a set of vertexes. ∀ v_i ∈ *PTmapV*(p_m, t_n): $m_i[p_{src}] \geqq \vec{\sigma}_i^*[t_{dst}]$. In Algorithm 4.7, the loop from line 3 to 15 selects a node v_i in *PTmapV* (p_{src}, t_{dst}). In line 3, ***NewVA* ($\underline{v}_i$,v_i)** is a function. The function creates a new vertex $\underline{v}_i$, sets its datum as the following, adds $\underline{v}_i$ to $\underline{V}$, and adds arcs { $\underline{a}_k$=($\underline{m}_a$, t_n, $\underline{m}_b$) | ($\underline{m}_a$=$\underline{m}_i$ and $\underline{m}_b$∈$\underline{M}$) or ($\underline{m}_a$∈$\underline{M}$ and $\underline{m}_b$=$\underline{m}_i$) } into $\underline{A}$ and $\underline{ATF}$.

- $\underline{m}_i \leftarrow m_i$
- $\underline{\vec{\sigma}_i^*} \leftarrow \vec{\sigma}_i^*$

In each turn, line 4 modifies the marking of v_i as described in Definition 4.9. Some of the markings after modification would be duplicated and codes from line 6 to 8 deal with this situation by calling *Merge* as in Algorithm 4.3 with corresponding parameters. After *Merge*, the node represented by the first parameter and its connected arc(s) are deleted. The loop from line 11 to 22 deletes the arcs in $\underline{ATF}$. Then, line 17 generates the node and line 19 generates the arc based on $\underline{m}_i\ [t_n > m_j^{'}$.

For example, when the loop from line 1 to 9 selects v_1, line 4 modifies $\underline{m}_1$ as m_1 - $\vec{\sigma}_1^*[t_3]$ * $PCV.p_1$. When the loop from line 11 to 22 finds $\underline{a}_5$ in Figure 3.9, arc $\underline{a}_5$ is deleted then. $v_{12}^{'}$ and $a_{10}^{'}$ in Figure 3.11 are generated based on $\underline{m}_2\ [t_3 > m_7^{'}$.

The correctness proof process for Algorithm 4.6 is quite similar to that for Algorithm 4.1. Thus, it is not discussed here.

Let *t* be the number of transitions in the Petri net, *n* be the number of vertexes in the input O-graph, and *n'* be the number of vertexes in the result O-graph. In algorithm 4.7, it is assumed that all markings in the O-graph of the original net should be modified one time. Thus, the loop from line 1 to 9 will run *n'* times. Because the loop from line 11 to 22 will run at *n'* times, the complexity of Algorithm 4.7 is *O*(*n'*). Thus, the complexity of Algorithm 4.6 is *O*(*n'*).

4.2 Token insertion

This section discusses the maintenance of the O-graph for *Token* insertion.

4.2.1 Scenario of Token insertions

The addToken example in section 3.2 can be translated into: *PN* is transformed into *PN'* by applying *addToken* to place p_2. A formal definition of a *Token* addition is described in Definition 4.12.

Definition 4.12 (Token addition)
Let *PN* = (*P*, *T*, *F*, m_0) be a Petri net. A Petri net *PN'* = (*P'*, *T'*, *F'*, m'_0) constructed by adding a token to place p_{tar} in *PN* has the following properties:
(1) $P' = P$;
(2) $T' = T$;
(3) $F' = F$;
(4) $m'_0 = m_0 + PCV.p_{tar}$

In a *Token* addition, both the original net *PN* and the refined net *PN'* have the same transitions, places, and arcs, but the initial marking of *PN'* has one more token in p_{tar} than m_0.

Definition 4.13 (state space mapping function for addToken SSM_E)
With Definition 4.12 and *M*', the set of all markings of *PN'*, the *state space mapping function* $\boldsymbol{SSM_E}$ is a function defined from *M* into *M'*:
$\forall m_i \in M$: $\boldsymbol{SSM_E}(m_i) = m_i + PCV.p_{tar}$

SSM_E is a function which maps the reachable markings of *PN* to the reachable markings of *PN'*. For example, while using marking m_1=(0,1,0,0) in Table A3.5(a) as input, the output marking from SSM_E is m'_1 = (0,1,1,0) = m_1 + $PCV.p_2$ = (0,1,0,0) + (0,0,1,0), as shown in

Table A3.6(a).

Definition 4.14 (arc mapping function for addToken R_E)
Let E = *addToken* be defined as in Definition 4.13. The *arc mapping function* $\boldsymbol{R_E}$ is a function defined from A into $M' \times T' \times M'$:

$\forall a_k = (m_i, t_n, m_j) \in A$: $\boldsymbol{R_E}(a_k) = (SSM_E(m_i), t_n, SSM_E(m_j))$;

As in addToken example, marking m_1 can enable the firing of transition t_1, i.e. a_1 is (m_1, t_1, m_2). After R_E is applied with inputs a_1, $R_E(a_1)$ is (m'_1, t_1, m'_2), as shown in Table A3.6(b).

Definition 4.15 (temporal occurrence graph for addToken $\underline{OG}_E$)
Let E = *addToken* be defined as in Definition 4.13, and $\underline{M}$ be the set of all markings generated from SSM_E. The *temporal occurrence graph* $\underline{OG}_E$ is the directed graph $\underline{OG}_E = (\underline{V}, \underline{A})$, where
(1) $\forall v_i \in V, \underline{v}_i \in \underline{V}$: $\underline{m}_i = SSM_E(m_i)$
(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M} \times T \times \underline{M} \mid \underline{m}_i [t_n > \underline{m}_j \}$
(3) $\forall a_k \in A, \underline{a}_k \in \underline{A}$: $\underline{a}_k = R_E(a_k)$

For example, Figure 4.3 is the $\underline{OG}_E$ after SSM_E and R_E are applied with input OG in Figure 3.13.

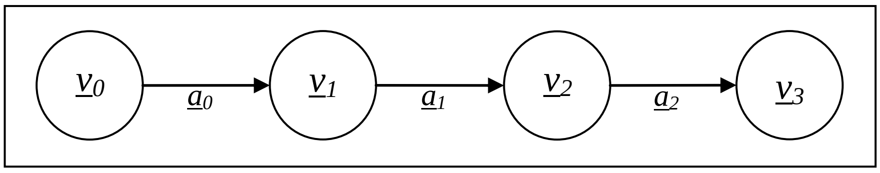


Figure 4.3 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.13.

Definition 4.16 (must re-generated marking set for addToken MRG_E)
For a *temporal occurrence graph* $\underline{OG}_E$ and $\underline{M}$, the set of all markings in $\underline{OG}_E$. $\underline{OG}_E$ might have a *must re-generated marking set* $\boldsymbol{MRG_E} = \underline{M}$:

MRG_E in Figure 4.3 is $\{\underline{m}_0, \underline{m}_1, \underline{m}_2, \underline{m}_3\}$.

4.2.2 Updating an Occurrence graph and the time complexity

The following algorithm presents a method to update the corresponding O-graph *OG'* when a token is added into a Petri net. Based on Definition 4.12, Algorithm 4.8 firstly modifies *OG* as $\underline{OG}_E$ (Definition 4.15), and then extends $\underline{OG}_E$ as the O-graph *OG'* = (*V'*, *A'*).

```
Algorithm 4.8 Main_ATo (PN, OG, $p_{tar}$) → PN' and OG'
// Input      PN: the original net
//            OG: the O-graph of PN
//            $p_{tar}$: a place in PN
// Output     PN': the refined net with $p_{tar}$
//            OG': the O-graph of PN'
● $M_{nr}$ ← ∮ : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for
which we have not yet found the successors.
● $MRG_E$ ← ∮ : a set of markings as in Definition 4.16.
● $\underline{OG}_E$ : a temporal occurrence graph. $\underline{V}$ ← ∮ and $\underline{A}$ ← ∮ .
● OG' : an occurrence graph. V' ← ∮ and A' ← ∮ .

1    add a token to $p_{tar}$              // construct PN'
2    ModifyOG_Ato                 // Algorithm 4.9
3    Findmarking                  // Algorithm 4.4
4    ExtendOG                     // Algorithm 4.5
```

Line 1 in Algorithm 4.8 constructs the refined Petri net *PN'*. Line 2 calls the function *ModifyOG_Ato* (Algorithm 4.9) to modify the state space in *OG*.

```
Algorithm 4.9   ModifyOG_ATo

1   $\underline{ATF} \leftarrow ATF$
2   $\underline{A} \leftarrow A$
3   for all $v_i \in TomapV\,(p_{tar})$ do          //the nodes affected by $addToken(PN, p_{tar})$
4   begin
5       $NewV(\underline{v}_i\ , v_i)$
6       $\underline{m}_i[p_{tar}] \leftarrow \underline{m}_i[p_{tar}] + 1$
7       // $SSM_E(m_i)$ in Definition 4.13
8       $MRG_E \leftarrow MRG_E \cup \{\underline{m}_i\}$
9   endfor
```

In Algorithm 4.9, the loop from line 3 to 9 selects a node v_i in $TomapV\,(p_{tar})$. In each turn, line 6 modifies the marking of v_i as described in Definition 4.13. $\underline{m}_i$ is added to MRG_E in line 8. For example, when the loop from line 3 to 9 selects v_1, line 6 modifies $\underline{m}_1$ as $\underline{m}_1 + PCV.p_2$.

Let t be the number of transitions in the Petri net, n be the number of vertexes in the input O-graph, and n' be the number of vertexes in the result O-graph. In algorithm 4.9, it is assumed that all markings in the O-graph of the original net should be modified one time. Thus, the loop from line 3 to 9 will run n times. Therefore, the complexity of Algorithm 4.9 is $O(n)$. The complexity of Algorithm 4.8 is $O(n + |n' - n| * t)$. Since designers usually edit the Petri nets sequentially, the complexity of Algorithm 4.8 is $O(n + t)$.

4.3 Transition/Place insertion

This section discusses the maintenance of the O-graph for *Transition* insertion. The O-graph maintenance for place insertion is adding a tuple into each marking.

4.3.1 Scenario of Transition insertions

Let a relation which belongs to $P \times t_{new} \times P$ be named as a *NTR* relation where t_{new} is the transition to be added. The *addTrans* example in section 3.2 can be translated into: *PN* is transformed into *PN'* by applying *addTrans* with a *NTR* relation (p_0, t_3, p_3). A formal definition of a *NTR* addition is described in Definition 4.17.

Definition 4.17 (*NTR* addition)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by adding a *NTR* relation $ntr = (p_{src}, t_{new}, p_{dst})$ into *PN* has the following properties.
(1) $P' = P$;
(2) $T' = T \cup \{ t_{new} \}$;
(3) $F' = F \cup \{ (p_{src}, t_{new}), (t_{new}, p_{dst}) \}$.

In a *NTR* addition, both the original net *PN* and the refined net *PN'* have the same places and tokens in the places, but *PN'* has two more arcs and one more transition representing the *NTR* relation added. Therefore, the initial markings of *PN* and *PN'* are equivalent, i.e. $m_0 = m_0'$.

Definition 4.18 ($\vec{\sigma}_i^*$ tuple adding function *ATT*)
The tuple adding function $\boldsymbol{ATT}(\vec{\sigma}_i^*, t_n)$ is a function in order to add one tuple for t_n in $\vec{\sigma}_i^*$. The newly added tuple is set 0.

Using Figure 4.4 as an example, $ATT(\vec{\sigma}_4^*, t_3)$ is $(1,1,0,\mathbf{0})$, where each dimension represents a transition with the firing counts.

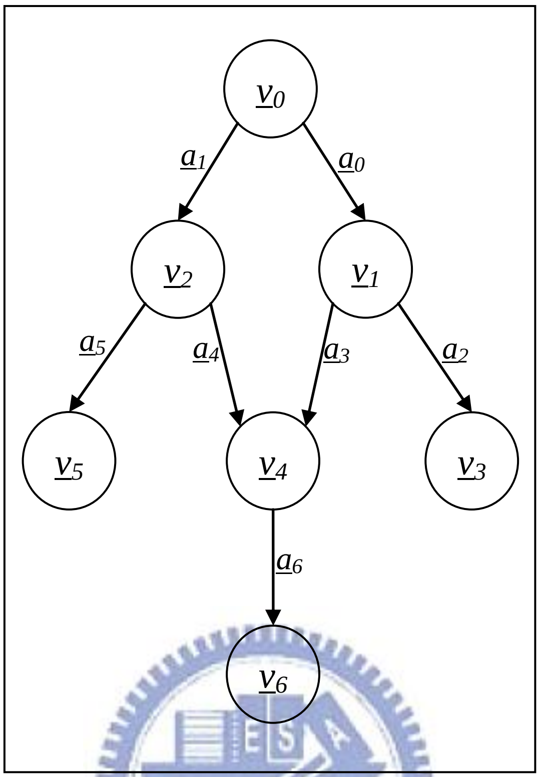


Figure 4.4 The temporal occurrence graph $\underline{OG_E}$ modified from Figure 3.17.

Definition 4.19 (temporal occurrence graph for addTrans $\underline{OG_E}$)
With Definition 4.17 and $\underline{M}$, the set of all markings generated from SSM_E. The *temporal occurrence graph* $\underline{OG_E}$ is the directed graph $\underline{OG_E} = (\underline{V}, \underline{A})$ with $ntr = (p_{src}, t_{new}, p_{dst})$, where

(1) $\forall v_i \in V, \underline{v}_i \in \underline{V}: \underline{m}_i = m_i$ and $\underline{\vec{\sigma}}_i^* = ATT(\vec{\sigma}_i^*, t_{new})$

(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M} \times T \times \underline{M} \mid \underline{m}_i [\ t_n > \underline{m}_j \}$

(3) $\forall a_k \in A, \underline{a}_k \in \underline{A}: \underline{a}_k = a_k$

For example, Figure 4.4 is the $\underline{OG_E}$ after *ATT* is applied with inputs *OG* in Figure 3.17 and $ntr = (p_0, t_3, p_3)$.

Definition 4.20 (must re-generated marking set for addTrans MRG_E)
Let a *temporal occurrence graph* $\underline{OG}_E$ and $\underline{M}$ be defined as above. $\underline{OG}_E$ might have a *must re-generated marking set* $\boldsymbol{MRG_E} \subseteq \underline{M}$, $ntr = (p_{src}, t_{new}, p_{dst})$:

$MRG_E = \{ \underline{m}_i \in \underline{M} \mid \underline{m}_i[p_{src}] > 0 \}$.

MRG_E in Figure 4.4 is $\{\underline{m}_0, \underline{m}_1, \underline{m}_2\}$. The outgoing arc, which does not belong to $\underline{A}$, from marking $\underline{m}_1$ is $(\underline{m}_1, t_3, m'_8)$, i.e. a'_8 in Figure 3.19.

4.3.2 Updating an Occurrence graph and the time complexity

The following algorithm presents a method to update the corresponding O-graph *OG'* when a *NTR* relation is added into a Petri net. Based on Definition 4.17, Algorithm 4.10 firstly modifies *OG* for $\underline{OG}_E$ (Definition 4.19), and then extends $\underline{OG}_E$ for the O-graph $OG' = (V', A')$.

```
Algorithm 4.10    Main_AT (PN, OG, ntr) → PN' and OG'
// Input      PN: the original net
//            OG: the O-graph of PN
//            ntr: a NTR relation
// Output     PN': the refined net with ntr
//            OG': the O-graph of PN'
● M_nr ← ∅ : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for which
we have not yet found the successors.
● MRG_E ← ∅ : a set of markings as in Definition 4.20.
● $\underline{OG}_E$ : a temporal occurrence graph. $\underline{V} \leftarrow \emptyset$ and $\underline{A} \leftarrow \emptyset$ .
● OG' : an occurrence graph. $V' \leftarrow \emptyset$ and $A' \leftarrow \emptyset$ .

1    add a transition $t_{new}$              // construct PN'
2    add an arc from $p_{src}$ to $t_{new}$
3    add an arc from $t_{new}$ to $p_{dst}$
4    ModifyOG_AT                    // Algorithm 4.11
5    Findmarking                    // Algorithm 4.4
6    ExtendOG                       // Algorithm 4.5
```

Line 1 to 3 in Algorithm 4.10 constructs the refined Petri net *PN'*. Line 4 calls the function *ModifyOG_AT* (Algorithm 4.11) to modify the state space in *OG*. The function *Findmarking* (Algorithm 4.4) in line 3 can generate the new node $v_j' \in V' \setminus \underline{V}$ and arc $a_k' \in A' \setminus \underline{A}$ according to MRG_E. For example, the new nodes in Figure 3.19 are v_7', v_8', and v_9'. The new arcs are a_7', a_8', and a_9'. Line 6 calls the function *ExtendOG* to generate the rest of new nodes and arcs whose ancestors are in M_{nr}, e.g., the node v_{10}' and the arcs a_{10}', a_{11}', and a_{12}' in Figure 3.19. The details of each step will be presented below.

```
Algorithm 4.11        ModifyOG_AT

1    $\underline{ATF} \leftarrow ATF$
2    $\underline{A} \leftarrow A$
3    for all $v_i \in TrmapV(t_{new})$ do
     //the nodes affected by $addTrans(PN, p_{src}, t_{new}, p_{dst})$
4    begin
5        $NewV(\underline{v}_i, v_i)$
6        $ATT(\vec{\sigma}_i^*, t_{new})$                // Definition 4.18
7        if $(\underline{m}_i(p_{src}) > 0)$
8            $MRG_E \leftarrow MRG_E \cup \{\underline{m}_i\}$
9        endif
10   endfor
```

In Algorithm 4.11, the loop from line 3 to 10 selects a node v_i in *TrmapV* (t_{new}). In each turn, line 6 modifies v_i as described in Definition 4.19(1). The markings which have tokens in place p_{src} as in Definition 4.20 would enable transition t_n. If $\underline{m}_i(p_{src}) > 0$, $\underline{m}_i$ is added to MRG_E in line 8. For example, when the loop from line 3 to 10 selects v_4, line 6 modifies $\vec{\sigma}_4^*$ as (1,1,0,**0**).

Let t be the number of transitions in the Petri net, n be the number of vertexes in the input O-graph, and n' be the number of vertexes in the result O-graph. In Algorithm 4.11, it is assumed that all markings in the O-graph of the original net should be modified one time. Thus, the loop from line 3 to 10 will run n times. Therefore, the complexity of Algorithm 4.11 is $O(n)$. The complexity of Algorithm 4.10 is $O(n + |n' - n| * t)$. Since designers usually edit the Petri nets sequentially, the complexity of Algorithm 4.10 is $O(n + t)$.

Chapter 5. Maintaining the O-graph for Item Deletion and Mergence

When editing a Petri net, every time designers can delete one of the following items: (1) an arc, (2) a token, (3) a transition, or (4) a place. Besides, designers can merge two places or two transitions. Correspondingly, the O-graph of the Petri net has to be modified and extended. The modifications and extensions are discussed in the following subsections respectively.

5.1 Arc Deletion

There are two types of arc deletions: (1) deleting an arc from a transition to a place and (2) deleting an arc from a place to a transition. Section 5.1.1 discusses the maintenance of the O-graph for the former and section 5.1.2 for the letter.

5.1.1 TtoP arc deletion

The delTtoParc example in section 3.3 can be translated into: *PN* is transformed into *PN'* by applying *delTtoParc* with a *TPR* relation (t_2, p_4). A formal definition of a *TPR* deletion is described in Definition 5.1.

Definition 5.1 (*TPR* deletion)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by deleting a *TPR* relation *tpr* in *PN* has the following properties:
(4) $P' = P$;
(5) $T' = T$;
(6) $F' = F \setminus \{ tpr \}$.

In a *TPR* deletion, both the original net *PN* and the refined net *PN'* have the same transitions, places, and tokens in the places, but *PN* has one more arc representing the *TPR* relation deleted. Therefore, the initial markings of *PN* and *PN'* are equivalent, i.e. $m_0 = m'_0$.

Definition 5.2 (state space mapping function for delTtoParc SSM_E)
With Definition 5.1 and *M'*, the set of all markings of *PN'*. The *state space mapping function* $\boldsymbol{SSM_E}$ is a function defined from *M* into *M'*, and $tpr = (t_{src}, p_{dst})$:
$\forall\, m_i \in M$: $\boldsymbol{SSM_E}(m_i) = m_i - \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst}$

SSM_E is a function which maps the reachable markings of *PN* to the reachable markings of *PN'*. For example, while using marking m_1=(0,0,0,1,**1**,0,0,0,0) in Table A3.9(a) as input, SSM_E outputs marking m'_1 = (0,0,0,1,0,0,0,0,0) = $m_1 - \vec{\sigma}_1^*[t_2] * PCV.p_4$ = (0,0,0,1,**1**,0,0,0,0) - 1 * (0,0,0,0,**1**,0,0,0,0), as shown in Table A3.10(a).

Definition 5.3 (arc mapping function for delTtoParc R_E)
Let *E* = *delTtoParc* be defined as in Definition 5.2, and *M'* be the set of all markings of *PN'*. The *arc mapping function* $\boldsymbol{R_E}$ is a function defined from *A* into $M' \times T' \times M'$, and $tpr = (t_{src}, p_{dst})$:
$\forall a_k = (m_i, t_n, m_j) \in A$: $\boldsymbol{R_E}(a_k) = (SSM_E(m_i), t_n, SSM_E(m_j))$;

For example, marking m_3 can enable the firing of transition t_3, i.e. a_5 in Table A3.9(b). After *arc mapping function* R_E is applied with inputs a_5, $R_E(a_5) = (m'_3, t_3, m'_5)$, as shown in Table A3.10(b).

Definition 5.4 (temporal occurrence graph for delTtoParc $\underline{OG}_E$)
Let *E* = *delTtoParc* be defined as in Definition 5.2, and $\underline{M}$ be the set of all markings from $SSM_E(M)$. The *temporal occurrence graph* $\underline{OG}_E$ is the directed graph $\underline{OG}_E = (\underline{V}, \underline{A})$ with $tpr = (t_{src}, p_{dst})$, where
(1) $\forall\, v_i \in V$, $\underline{v}_i \in \underline{V}$: $\underline{m}_i = SSM_E(m_i)$

(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M} \times T \times \underline{M} \mid \underline{m}_i [t_n > \underline{m}_j \}$
(3) $\forall a_k \in A \backslash ATF, \underline{a}_k \in \underline{A}: \underline{a}_k = R_E(a_k)$

For example, Figure 5.1 is the $\underline{OG}_E$ after SSM_E and R_E are applied with input OG in Figure 3.21.

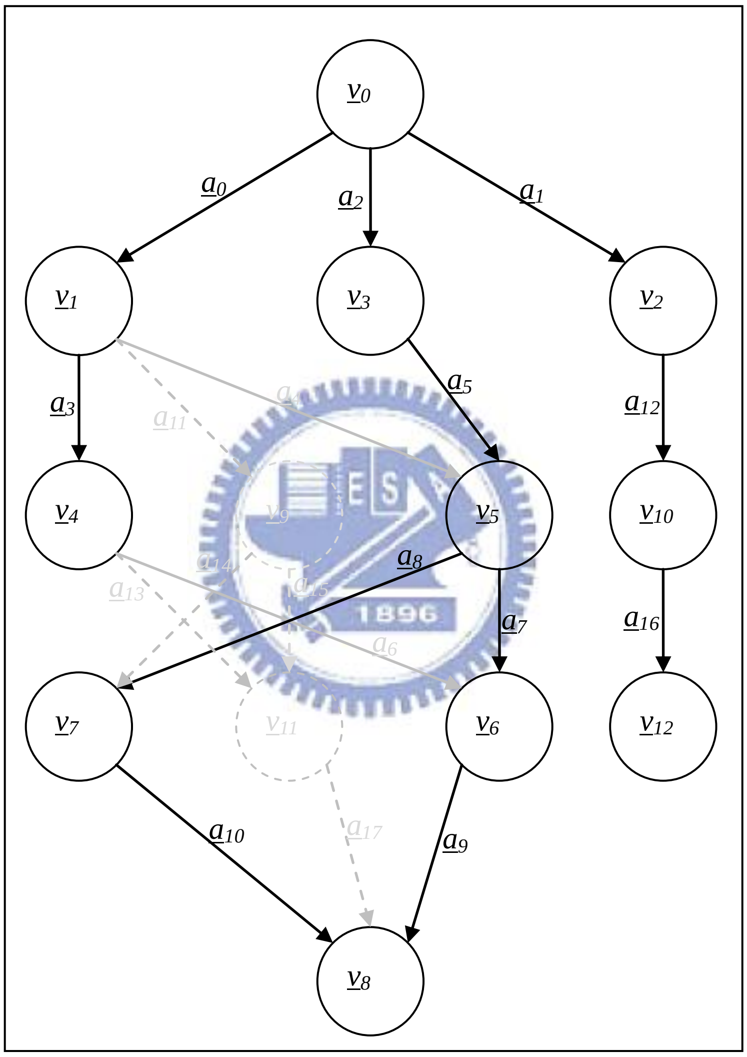


Figure 5.1 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.21.

5.1.1.2 Updating an Occurrence graph and the time complexity

The following algorithm presents a method to update the corresponding O-graph *OG'* when a *TPR* relation is deleted in a Petri net. Based on Definition 5.1, Algorithm 5.1 firstly modifies *OG* for $\underline{OG}_E$ (Definition 5.4), and then extends $\underline{OG}_E$ for the O-graph *OG'* = (*V'*, *A'*).

```
Algorithm 5.1 Main_DTP (PN, OG, tpr) → PN' and OG'
// Input      PN: the original net
//            OG: the O-graph of PN
//            tpr: a TPR relation
// Output     PN': the refined net without tpr
//            OG': the O-graph of PN'
● $M_{nr}$ ← § : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for which
we have not yet found the successors.
● $\underline{OG}_E$ : a temporal occurrence graph. $\underline{V}$ ← § and $\underline{A}$ ← § .
● OG' : an occurrence graph. V' ← § and A' ← § .

1   delete the arc from $t_{src}$ to $p_{dst}$      // construct PN'
2   ModifyOG_DTP                          // Algorithm 5.2
3   ExtendOG                              // Algorithm 4.5
```

Line 1 in Algorithm 5.1 constructs the refined Petri net *PN'*. Line 2 calls the function *ModifyOG_DTP* (Algorithm 5.2) to modify the state space in *OG* and redirect the flow relations in *ATF*. For example, the redirection changes from a_4 to a'_{18}. Line 3 calls the function *ExtendOG* (Algorithm 4.5) to generate the rest of new nodes and arcs whose ancestors are in M_{nr}. The details of each step will be presented below.

```
Algorithm 5.2 ModifyOG_DTP

1   for all $v_i \in PTmapV(p_{dst}, t_{src})$ do
2   begin
3       $NewVA(\underline{v}_i, v_i)$
4       $\underline{m}_i[p_{dst}] \leftarrow \underline{m}_i[p_{dst}] - \vec{\underline{\sigma}}^*_i[t_{src}]$
5       // $SSM_E(m_i)$ in Definition 5.2
6       if($\exists \underline{m}_j \in \underline{M}$ where $\underline{m}_i = \underline{m}_j$)
7           $Merge(\underline{v}_i, \underline{v}_j)$      //Algorithm 4.3
8       endif
```

```
9    endfor
10
11   for all $\underline{a}_k = (\underline{m}_i, t_n, \underline{m}_j) \in \underline{ATF}$ do
12   begin
13       $\underline{A} \leftarrow \underline{A} \setminus \{ \underline{a}_k \}$
14       $\underline{ATF} \leftarrow \underline{ATF} \setminus \{ \underline{a}_k \}$
15       if($t_n$ can be enabled at $\underline{m}_i$)
16           $m_j'$ is the marking produced after $\underline{m}_i$ fires $t_n$
17           $v_j' \leftarrow Node\ (\underline{v}_i, t_n, m_j')$
18           $V' \leftarrow V' \cup \{v_j'\}$
19           $a_l' \leftarrow Arc\ (\underline{m}_i, t_n, m_j')$
20           $A' \leftarrow A' \cup \{a_l'\}$
21       endif
22   endfor
```

In Algorithm 5.2, the loop from line 1 to 9 selects a node v_i in *PTmapV* (p_{dst}, t_{src}). In each turn, line 4 modifies the marking of v_i as described in Definition 5.2. Some of the markings after modification would be duplicated and codes from line 6 to 8 deal with this situation by calling *Merge* as in Algorithm 4.3 with corresponding parameters. After *Merge*, the node represented by the first parameter and its connected arc(s) are deleted. The loop from line 11 to 22 deletes the arcs in $\underline{ATF}$. Then, line 17 generates the node and line 19 generates the arc based on $\underline{m}_i\ [t_n > m_j'$.

For example, when the loop from line 1 to 9 selects v_1, line 4 modifies $\underline{m}_1$ as $\underline{m}_1$ - $\vec{\underline{\sigma}}_1^*[t_2]$ * $PCV.p_4$. When the loop from line 11 to 22 selects $\underline{a}_4$, arc $\underline{a}_4$ in Figure 5.1 is deleted. v_{10}' and a_{18}' in Figure 3.23 are generated based on $\underline{m}_1\ [t_5 > m_{10}'$.

Let t be the number of transitions in the Petri net, n be the number of vertexes in the input O-graph, and n' be the number of vertexes in the result O-graph. In algorithm 5.2, it is assumed that all markings in the O-graph of the original net should be modified one time. Thus, the loop from line 1 to 9 will run n' times. Because the loop from line 11 to 22 will run n' times, the complexity of Algorithm 5.2 is $O(n')$. Thus, the complexity of Algorithm 5.1 is

O(*n'*).

5.1.2 PtoT arc deletion

5.1.2.1 Scenario of PtoT arc deletions

The delPtoTarc example in section 3.3 can be translated into: *PN* is transformed into *PN'* by applying *delPtoTarc* with a *PTR* relation (p_1, t_3). A formal definition of a *PTR* deletion is described in Definition 5.5.

Definition 5.5 (*PTR* deletion)
Let *PN* = (*P*, *T*, *F*, m_0) be a Petri net. A Petri net *PN'* = (*P'*, *T'*, *F'*, m'_0) constructed by deleting a *PTR* relation *ptr* in *PN* has the following properties:
(1) $P' = P$;
(2) $T' = T$;
(3) $F' = F \setminus \{ ptr \}$.

In a *PTR* deletion, both the original net *PN* and the refined net *PN'* have the same transitions, places, and tokens in the places, but *PN* has one more arc representing the *PTR* relation deleted. Therefore, the initial markings of *PN* and *PN'* are equivalent, i.e. $m_0 = m'_0$.

Definition 5.6 (state space mapping function for delPtoTarc SSM_E)
With Definition 5.5 and *M*', the set of all markings of *PN'*. The *state space mapping function* $\boldsymbol{SSM_E}$ is a function defined from *M* into *M'*, and *ptr* = (p_{src}, t_{dst}):

$$\forall\ m_i \in M\text{: } \boldsymbol{SSM_E}(m_i) = m_i + \vec{\sigma}_i^*[t_{dst}] * PCV.p_{src}$$

SSM_E is a function which maps the reachable markings of *PN* to the reachable markings of *PN'*. For example, while using marking m_6=(0,0,0,1) in Table A3.11(a) as input, SSM_E outputs marking m'_6 = (0,1,0,1) = $m_6 + \vec{\sigma}_6^*[t_3] * PCV.p_1$ = (0,0,0,1) + 1 * (0,1,0,0) as shown in Table A3.12(a).

Definition 5.7 (arc mapping function for delPtoTarc R_E)
Let E = *delPtoTarc* be defined as in Definition 5.6, and M' be the set of all markings of PN'. The *arc mapping function* $\boldsymbol{R_E}$ is a function defined from A into $M' \times T' \times M'$,
and $ptr = (p_{src}, t_{dst})$:

$\forall a_k = (m_i, t_n, m_j) \in A$: $\boldsymbol{R_E}(a_k) = (SSM_E(m_i), t_n, SSM_E(m_j))$;

For example, both marking m_0 and m_1 can enable the firing of transition t_1, i.e. a_1 and a_3 in Table A3.11(b). After *arc mapping function* R_E is applied with inputs a_1 and a_3, $R_E(a_1) = (m'_0, t_1, m'_2)$ and $R_E(a_3) = (m'_1, t_1, m'_4)$, as shown in Table A3.12(b).

Definition 5.8 (temporal occurrence graph for delPtoTarc $\underline{OG}_E$)
Let E = *delPtoTarc* be defined as in Definition 5.6, and $\underline{M}$ be the set of all markings from $SSM_E(M)$. The *temporal occurrence graph* $\underline{OG}_E$ is the directed graph $\underline{OG}_E = (\underline{V}, \underline{A})$ with $ptr = (p_{src}, t_{dst})$, where
(1) $\forall v_i \in V, \underline{v}_i \in \underline{V}$: $\underline{m}_i = SSM_E(m_i)$
(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M} \times T \times \underline{M} \mid \underline{m}_i [t_n > \underline{m}_j \}$
(3) $\forall a_k \in A \backslash ATF, \underline{a}_k \in \underline{A}$: $\underline{a}_k = R_E(a_k)$

For example, Figure 5.2 is the $\underline{OG}_E$ after SSM_E and R_E are applied with input OG in Figure 3.25.

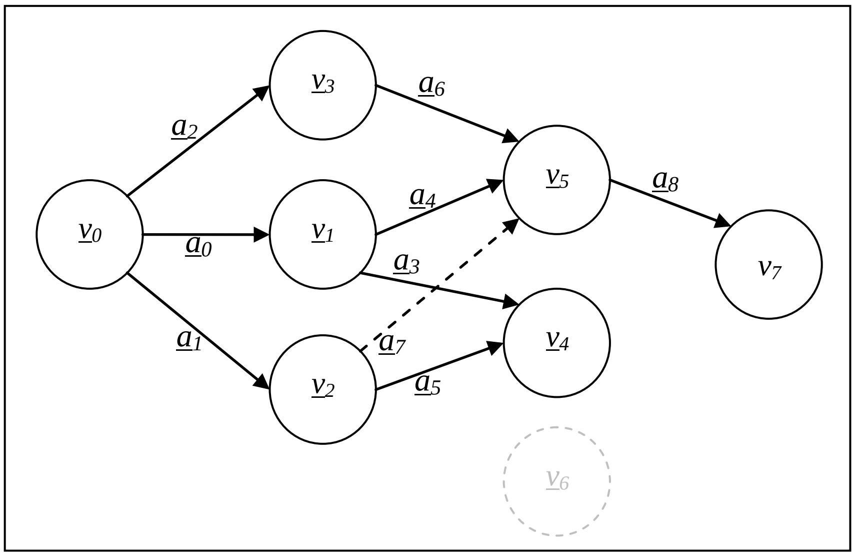


Figure 5.2 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.25.

Definition 5.9 (must re-generated marking set for delPtoTarc MRG_E)
For a *temporal occurrence graph* $\underline{OG}_E$ and $\underline{M}$, the set of all markings in $\underline{OG}_E$. $\underline{OG}_E$ might have a *must re-generated marking set* $\boldsymbol{MRG_E} \subseteq \underline{M}$, $ptr = (p_{src}, t_{dst})$:

$MRG_E = \{ \underline{m}_i \in \underline{M} | \ \vec{\underline{\sigma}}_i^*[t_{dst}]>0 \text{ or } \underline{m}_i[\bullet t_{dst}]>0 \}$.

Since $\underline{m}_2[p_2]$=1 and $\underline{m}_4[p_2]$=1, MRG_E in Figure 5.2 is $\{\underline{m}_2, \underline{m}_4\}$.

5.1.2.2 Updating an Occurrence graph and the time complexity

The following algorithm presents a method to update the corresponding O-graph *OG'* when a *PTR* relation is deleted in a Petri net. Based on Definition 5.5, Algorithm 5.3 firstly modifies *OG* for $\underline{OG}_E$ (Definition 5.8), and then extends $\underline{OG}_E$ for the O-graph *OG'* = (*V'*, *A'*).

```
Algorithm 5.3 Main_DPT (PN, OG, ptr) → PN' and OG'
// Input      PN: the original net
//            OG: the O-graph of PN
//            ptr: a PTR relation
// Output     PN': the refined net without ptr
//                    OG': the O-graph of PN'
● $\boldsymbol{M_{nr}} \leftarrow \S$ : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for which
we have not yet found the successors.
● $\boldsymbol{MRG_E} \leftarrow \S$ : a set of markings as in Definition 5.9.
● $\underline{\boldsymbol{OG_E}}$ : a temporal occurrence graph. $\underline{V} \leftarrow \S$ and $\underline{A} \leftarrow \S$ .
● $\boldsymbol{OG'}$ : an occurrence graph. $V' \leftarrow \S$ and $A' \leftarrow \S$ .

1   delete the arc from $p_{src}$ to $t_{dst}$     // construct PN'
2   ModifyOG_DPT                    // Algorithm 5.4
3   Findmarking                     // Algorithm 4.4
4   ExtendOG                        // Algorithm 4.5
```

Line 1 in Algorithm 5.3 constructs the refined Petri net *PN'*. Line 2 calls the function *ModifyOG _DPT* (Algorithm 5.4) to modify the state space in *OG* and redirect the flow relations in *ATF*. The details of each step will be presented below.

```
Algorithm 5.4 ModifyOG_DPT

1   $\underline{ATF} \leftarrow ATF$
2   $\underline{A} \leftarrow A$
3   for all $v_i \in TmapV(t_{dst})$ do
4   begin
5       $NewV(\underline{v}_i, v_i)$
6       $\underline{m}_i[p_{src}] \leftarrow \underline{m}_i[p_{src}] + \vec{\sigma}_i^*[t_{dst}]$
7       // $SSM_E(m_i)$ in Definition 5.6
8       if($\exists \underline{m}_j \in \underline{M}$ where $\underline{m}_i = \underline{m}_j$)
9           if($\vec{\sigma}_i^*[t_{dst}] > \vec{\sigma}_j^*[t_{dst}]$)
10              $Merge(\underline{v}_i, \underline{v}_j)$
11          else
12              $Merge(\underline{v}_j, \underline{v}_i)$
13          endif
14      else  if( $\vec{\sigma}_i^*[t_{dst}]>0$ or $\underline{m}_i[\bullet t_{dst}]>0$ )// $\underline{m}_i \in MRG_E$
15          $MRG_E \leftarrow MRG_E \cup \{\underline{m}_i\}$
16      endif
17  endfor
```

```
18
19  for all $\underline{a}_k = (\underline{m}_i, t_n, \underline{m}_j) \in \underline{ATF}$ do
20  begin
21      if( $(\vec{\sigma}_i^* + TCV.t_n)[t_{dst}] \neq \vec{\sigma}_j^*[t_{dst}]$ )
22          $\underline{A} \leftarrow \underline{A} \setminus \{ \underline{a}_k \}$
23          $\underline{ATF} \leftarrow \underline{ATF} \setminus \{ \underline{a}_k \}$
24          $m_j'$ is the marking produced after $\underline{m}_i$ fires $t_n$
25          $v_j' \leftarrow Node\ (\underline{v}_i, t_n, m_j')$
26          $V' \leftarrow V' \cup \{v_j'\}$
27          $a_l' \leftarrow Arc\ (\underline{m}_i, t_n, m_j')$
28          $A' \leftarrow A' \cup \{a_l'\}$
29      endif
30  endfor
```

In Algorithm 5.4, the loop from line 3 to 17 selects a node v_i in $TmapV\ (t_{dst})$. In each turn, line 6 modifies the marking of v_i as described in Definition 5.6. Some of the markings after modification would be duplicated and codes from line 9 to 13 deal with this situation by calling *Merge* as in Algorithm 4.3 with corresponding parameters. After *Merge*, the node represented by the first parameter and its connected arc(s) are deleted. Because the parameters of *Merge* in line 10 and 12 are different, the node to be deleted is $\underline{v}_i$ or $\underline{v}_j$ respectively. Otherwise, if $\vec{\sigma}_i^*[t_{dst}]>0$ or $\underline{m}_i[\bullet t_{dst}]>0$, $\underline{m}_i$ is added to MRG_E in line 15. The loop from line 19 to 30 deletes the arcs in $\underline{ATF}$ according to "if condition" in line 21. Then, line 25 generates the node with $\underline{v}_i$, t_n, and m_j'. Line 27 generates the arc based on $\underline{m}_i\ [t_n > m_j'$.

For example, when the loop from line 3 to 17 selects v_6, line 6 modifies $\underline{m}_6$ as $\underline{m}_6 + \vec{\sigma}_6^*[t_3]$ * $PCV.p_1$. Assume that $\underline{m}_5$ belongs to $\underline{M}$. When v_6 is selected, the marking $\underline{m}_5$ and $\underline{m}_6$ in Figure 5.2 are both (0,1,0,1) and $\vec{\sigma}_5^*[t_3] < \vec{\sigma}_6^*[t_3]$ which satisfy the conditions in line 8 and 11. Hence, $Merge(\underline{v}_6, \underline{v}_5)$ is called in line 12.

Let t be the number of transitions in the Petri net, n be the number of vertexes in the input O-graph, and n' be the number of vertexes in the result O-graph. In Algorithm 5.4, it is assumed that all markings in the O-graph of the original net should be modified one time. Thus, the loop from line 3 to 17 will run *n* times. Because the loop from line 19 to 30 will run *n* times, the complexity of Algorithm 5.4 is $O(n)$. Therefore, the complexity of Algorithm 5.4 is $O(n)$. The complexity of Algorithm 5.3 is $O(n + |n' - n| * t)$. Since designers usually edit the Petri nets sequentially, the complexity of Algorithm 5.3 is $O(n + t)$.

5.2 Token Deletion

This section discusses the maintenance of the O-graph for *Token* deletion.

5.2.1 Scenario of Token Deletion

The delToken example in section 3.3 can be translated into: *PN* is transformed into *PN'* by applying *delToken* with a place p_2. A formal definition of a *Token* deletion is described in Definition 5.10.

Definition 5.10 (*Token* deletion)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by deleting a token from p_{tar} in *PN* has the following properties:
(1) $P' = P$;
(2) $T' = T$;
(3) $F' = F$;
(4) $m'_0 = m_0 - PCV.p_{tar}$.

In a *Token* deletion, both the original net *PN* and the refined net *PN'* have the same transitions, places, and arcs, but the initial marking of *PN* has one more token in place p_{tar} representing the deleted token.

Definition 5.11 (state space mapping function for delToken SSM_E)
With Definition 5.10 and M', the set of all markings of *PN'*. The *state space mapping function* $\boldsymbol{SSM_E}$ is a function defined from M into M':
$\forall m_i \in M$: $\boldsymbol{SSM_E}(m_i) = m_i - PCV.p_{tar}$

SSM_E is a function which maps the reachable markings of *PN* to the reachable markings of *PN'*. For example, while using marking m_1=(0,1,**1**,0) in Table A3.13(a) as input, SSM_E outputs marking m_1' =(0,1,0,0) =m_1-$PCV.p_2$ =(0,1,**1**,0) - (0,0,**1**,0), shown in Table A3.14(a).

Definition 5.12 (arc mapping function for delToken R_E)
Let *E* = *delToken* be defined as in Definition 5.11, and M' be the set of all markings of *PN'*. The *arc mapping function* $\boldsymbol{R_E}$ is a function defined from A into $M' \times T' \times M'$:
$\forall a_k = (m_i, t_n, m_j) \in A$: $\boldsymbol{R_E}(a_k) = (SSM_E(m_i), t_n, SSM_E(m_j))$;

For example, marking m_2 can enable the firing of transition t_2, i.e. a_2 is (m_2, t_2, m_3). After *arc mapping function* R_E is applied with inputs a_2, $R_E(a_2)$ = (m_2',t_2,m_3'), shown in Table A3.14(b).

Definition 5.13 (temporal occurrence graph for delToken $\underline{OG}_E$)
Let *E* = *delToken* be defined as in Definition 5.11, and $\underline{M}$ be the set of all markings generated from SSM_E. The *temporal occurrence graph* $\underline{OG}_E$ is the directed graph $\underline{OG}_E = (\underline{V}, \underline{A})$, where
(1) $\forall v_i \in V$, $\underline{v}_i \in \underline{V}$: $\underline{m}_i = SSM_E(m_i)$
(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M} \times T \times \underline{M} \mid \underline{m}_i [t_n > \underline{m}_j \}$
(3) $\forall a_k \in A$, $\underline{a}_k \in \underline{A}$: $\underline{a}_k = R_E(a_k)$

For example, Figure 5.3 is the $\underline{OG}_E$ after SSM_E and R_E are applied with input *OG* in Figure 3.29.

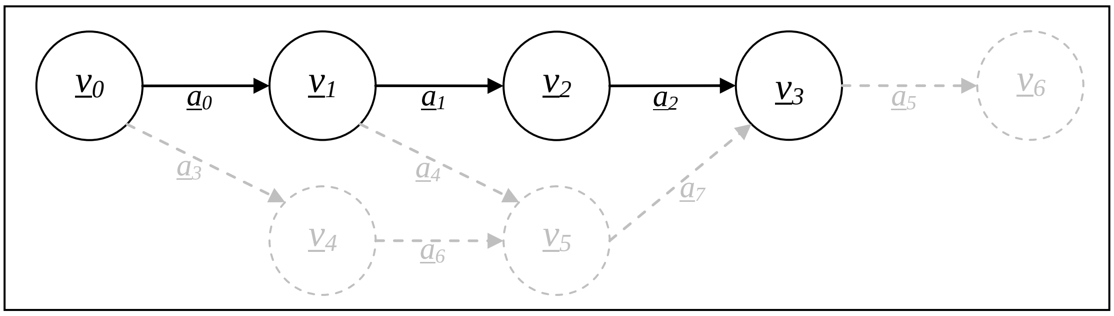


Figure 5.3 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.29.

5.2.2 Updating an Occurrence graph and the time complexity

The following algorithm presents a method to update the corresponding O-graph *OG’* when a token is deleted from a Petri net. Based on Definition 5.10, Algorithm 5.5 modifies *OG* as *OG’*.

Algorithm 5.5 ***Main_DTo* (*PN*, *OG*, p_{tar}) → *PN’* and *OG’***

```
// Input    PN: the original net
//          OG: the O-graph of PN
//          $p_{tar}$: a place in PN
// Output   PN’: the refined net without $p_{tar}$
//          OG’: the O-graph of PN’
● $M_{nr}$ ← § : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for which
we have not yet found the successors.
● OG’ : an occurrence graph. V’ ← § and A’ ← § .

1   delete a token from $p_{tar}$          // construct PN’
2   ModifyOG_DTo                  // Algorithm 5.6
```

Line 1 in Algorithm 5.5 constructs the refined Petri net *PN’*. Line 2 calls the function *ModifyOG_DTo* (Algorithm 5.6) to modify the state space in *OG*. The details of each step will be presented below.

```
Algorithm 5.6 ModifyOG_DTo

1   for all $v_i \in TomapV(p_{tar})$ do        //the nodes affected by $delToken(PN, p_{tar})$
2   begin
3       $NewVA(v_i', v_i)$
4       $m_i'[p_{tar}] \leftarrow m_i'[p_{tar}] - 1$
5       // $SSM_E(m_i)$ in Definition 5.11
6   endfor
```

In line 1, *TomapV* (p_{tar}) is a function that maps place p_{tar} to a set of vertexes. $\forall\ v_i \in$ *TomapV* (p_{tar}): $m_i[p_{tar}] > 0$. In Algorithm 5.6, the loop from line 1 to 6 selects a node v_i in *TomapV* (p_{tar}). In each turn, line 4 modifies the marking of v_i as described in Definition 5.11. For example, when the loop from line 1 to 6 selects v_1, line 7 modifies $\underline{m_1}$ as m_1-$PCV.p_2$.

Let *t* be the number of transitions in the Petri net, *n* be the number of vertexes in the input O-graph, and *n'* be the number of vertexes in the result O-graph. In algorithm 5.6, it is assumed that all markings in the O-graph of the original net should be modified one time. Because the loop from line 1 to 6 will run *n'* times, the complexity of Algorithm 5.6 is $O(n')$. Thus, the complexity of Algorithm 5.5 is $O(n')$.

5.3 Transition Deletion

This section discusses the maintenance of the O-graph for *Transition* deletion.

5.3.1 Scenario of Transition Deletion

The delTransition example in section 3.3 can be translated into: *PN* is transformed into *PN'* by applying *delTransition* with a transition t_3. A formal definition of a *Transition* deletion is described in Definition 5.14.

Definition 5.14 (*Transition* deletion)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by deleting a transition t_{tar} in PN has the following properties:
(1) $P' = P$;
(2) $T' = T \setminus \{t_{tar}\}$;
(3) $F' = F \setminus F_{cwt}$, where $F_{cwt} = \{f | (\forall(p_i, t_{tar}) \in F: f=(p_i, t_{tar}))$
or $(\forall(t_{tar}, p_j) \in F: f=(t_{tar}, p_j))\}$.

In a *Transition* deletion, both the original net PN and the refined net PN' have the same places and tokens in the places, but PN has one more transition representing the deleted transition. Therefore, the initial markings of PN and PN' are equivalent, i.e. $m_0 = m_0^{'}$. Also, the connected arcs are deleted automatically.

Definition 5.15 ($\vec{\sigma}_i^*$ tuple deleting function DTT)
The tuple deleting function $\boldsymbol{DTT}(\vec{\sigma}_i^*, t_n)$ is a function in order to delete one tuple for t_n in $\vec{\sigma}_i^*$.

Using Figure 3.33 as an example, $DTT(\vec{\sigma}_4^*, t_3)$ is (1,1,0), where each dimension represents a transition with the firing counts.

Definition 5.16 (temporal occurrence graph for delTransition $\underline{OG_E}$)
Let $E = delTransition$ be defined as in Definition 5.14, and $\underline{M}$ be the set of all markings of P. The *temporal occurrence graph* $\underline{OG_E}$ is the directed graph $\underline{OG_E} = (\underline{V}, \underline{A})$, where
(1) $\forall v_i \in V, \underline{v_i} \in \underline{V}: \underline{m_i} = m_i$ and $\underline{\vec{\sigma}_i^*} = DTT(\vec{\sigma}_i^*, t_{tar})$
(2) $\underline{A} = \{ (\underline{m_i}, t_n, \underline{m_j}) \in \underline{M} \times T \times \underline{M} \mid \underline{m_i} [t_n > \underline{m_j} \}$
(3) $\forall a_k \in A \setminus ATF, \underline{a_k} \in \underline{A}: \underline{a_k} = R_E(a_k)$

For example, Figure 5.4 is the $\underline{OG_E}$ after SSM_E and R_E are applied with input OG in

Figure 3.33.

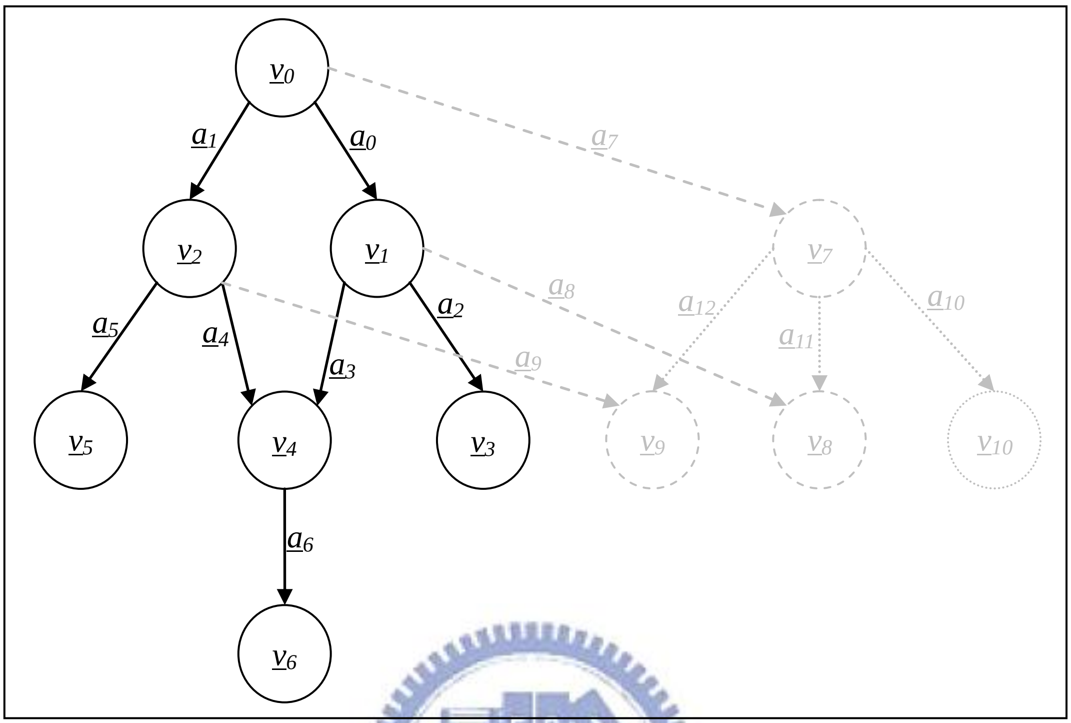


Figure 5.4 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.33.

5.3.2 Updating an Occurrence graph and the time complexity

The following algorithm presents a method to update the corresponding O-graph *OG'* when a transition is deleted from a Petri net. Based on Definition 5.14, Algorithm 5.7 firstly modifies *OG* for $\underline{OG}_E$ (Definition 5.16), and then extends $\underline{OG}_E$ for the O-graph *OG'* = (*V'*, *A'*).

```
Algorithm 5.7 Main_DT (PN, OG, $t_{tar}$) → PN' and OG'
// Input     PN: the original net
//           OG: the O-graph of PN
//           $t_{tar}$: a transition in PN
// Output    PN': the refined net without $t_{tar}$
//           OG': the O-graph of PN'
● $M_{nr}$ ← § : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for which
we have not yet found the successors.
```

```
● OG' : an occurrence graph. $V' \leftarrow$ § and $A' \leftarrow$ § .

1    delete transition $t_{tar}$ and the connected arcs      // construct PN'
2    ModifyOG_DT                                              // Algorithm 5.8
3    ExtendOG                                                 // Algorithm 4.5
```

Line 1 in Algorithm 5.7 constructs the refined Petri net *PN'*. Line 2 calls the function *ModifyOG_DT* (Algorithm 5.8) to modify the state space in *OG* and redirect the flow relations in *ATF*. The details of each step will be presented below.

```
Algorithm 5.8 ModifyOG_DT

1    for all $v_i \in TrmapV(t_{tar})$ do    //the nodes affected by delTransition(PN, $t_{tar}$)
2    begin
3        NewV($\underline{v}_i$ ,$v_i$)
4        DTT($\underline{\vec{\sigma}_i^*}$, $t_{tar}$)
5    endfor
6
7    for all $\underline{a}_k = (\underline{m}_i, t_n, \underline{m}_j) \in \underline{ATF}$ do
8    begin
9        $\underline{A} \leftarrow \underline{A} \setminus \{ \underline{a}_k \}$
10       $\underline{ATF} \leftarrow \underline{ATF} \setminus \{ \underline{a}_k \}$
11       if($t_n$ can be enabled at $\underline{m}_i$)
12           $m_j'$ is the marking produced after $\underline{m}_i$ fires $t_n$
13           $v_j' \leftarrow Node(\underline{v}_i, t_n, m_j')$
14           $V' \leftarrow V' \cup \{v_j'\}$
15           $a_l' \leftarrow Arc(\underline{m}_i, t_n, m_j')$
16           $A' \leftarrow A' \cup \{a_l'\}$
17       endif
18   endfor
```

In line 1, *TrmapV* (t_{tar}) is a function that maps transition t_{tar} to a set of vertexes. $\forall\ v_i \in$ *TrmapV* (t_{tar}): $\vec{\sigma}_i^*[t_{tar}] = 0$. In Algorithm 5.8, the loop from line 1 to 5 selects a node v_i in *TrmapV* (t_{tar}). In each turn, line 4 modifies $\underline{v}_i$ as described in Definition 5.16(1). The loop from line 7 to 18 deletes the arcs in <u>*ATF*</u>. Then, line 13 generates the node and line 15 generates the arc based on $\underline{m}_i\ [t_n > m_j'$. For example, when the loop from line 1 to 5 selects v_4, line 4 modifies $\vec{\sigma}_4^*$ as (1,1,0).

Let t be the number of transitions in the Petri net, n be the number of vertexes in the input O-graph, and n' be the number of vertexes in the result O-graph. In algorithm 5.8, it is assumed that all markings in the O-graph of the original net should be modified one time. Because the loop from line 1 to 5 will run n' times, the complexity of Algorithm 5.8 is $O(n')$. Thus, the complexity of Algorithm 5.7 is $O(n')$.

5.4 Place Deletion

This section discusses the maintenance of the O-graph for *Place* deletion.

5.4.1 Scenario of Place Deletion

The delPlace example in section 3.3 can be translated into: *PN* is transformed into *PN'* by applying *delPlace* with a place p_3. A formal definition of a *Place* deletion is described in Definition 5.17.

Definition 5.17 (*Place* deletion)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by deleting a place p_{tar} in *PN* has the following properties:
(1) $P' = P \setminus \{ p_{tar} \}$;
(2) $T' = T$;
(3) $F' = F \setminus (F_{cpt} \cup F_{cph})$, where $F_{cpt} = \{ f=(t_i, p_j) \in F \mid p_j=p_{tar} \}$ and
$F_{cph} = \{ f=(p_i, t_j) \in F \mid p_i=p_{tar} \}$.

In a *Place* deletion, both the original net *PN* and the refined net *PN'* have the same transitions and tokens in the places, but *PN* has one more place representing the deleted place. Therefore, the initial markings of *PN* and *PN'* are equivalent, i.e. $m_0 = m_0'$. Also, the connected arcs are deleted automatically.

Definition 5.18 (m_i tuple deleting function *DPT*)
The tuple deleting function ***DPT***(m_i, p_n) is a function in order to delete one tuple for p_n in m_i.

Using Figure 3.37 as an example, *DPT*(m_1, p_3) is (0,1,0,1,0), where each dimension represents a place with the number of tokens.

Definition 5.19 (arc mapping function for delPlace R_E)
Let *E* = *delPlace* be defined as in Definition 5.17, and *M*' be the set of all markings of *PN'*. The *arc mapping function* $\boldsymbol{R_E}$ is a function defined from *A* into *M*' × *T'* × *M*':

$\forall a_k = (m_i, t_n, m_j) \in A$: $\boldsymbol{R_E}(a_k) = (DPT(m_i, p_{tar}), t_n, DPT(m_j, p_{tar}))$;

For example, marking m_1 can enable the firing of transition t_3, i.e. $a_2=(m_1, t_3, m_2)$. After *arc mapping function* R_E is applied with inputs a_2, $R_E(a_2) = (m_1', t_3, m_2')$ as shown in Table A3.18(b).

Definition 5.20 (temporal occurrence graph for delPlace $\underline{OG_E}$)
Let *E* = *delPlace* be defined as in Definition 5.17, and $\underline{M}$ be the set of all markings generated from SSM_E. The *temporal occurrence graph* $\underline{OG_E}$ is the directed graph $\underline{OG_E} = (\underline{V}, \underline{A})$, where

(1) $\forall v_i \in V, \underline{v_i} \in \underline{V}: \underline{m_i} = DPT(m_i, p_{tar})$

(2) $\underline{A} = \{ (\underline{m_i}, t_n, \underline{m_j}) \in \underline{M} \times T \times \underline{M} \mid \underline{m_i} [\, t_n > \underline{m_j} \}$

(3) $\forall a_k \in A, \underline{a_k} \in \underline{A}: \underline{a_k} = R_E(a_k)$

For example, Figure 5.5 is the $\underline{OG_E}$ after SSM_E and R_E are applied with input *OG* in

Figure 3.37.

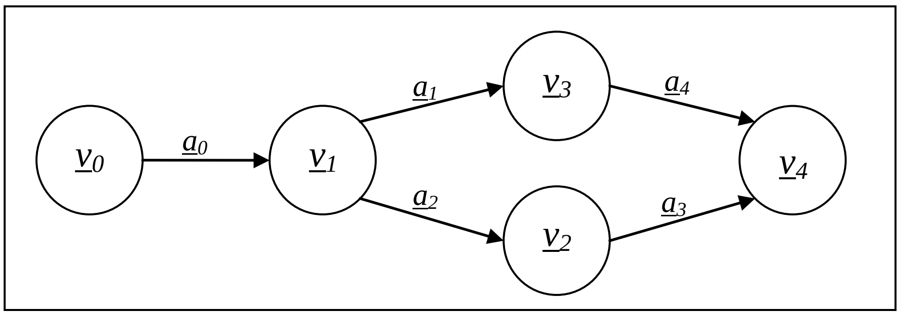


Figure 5.5 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.37.

Definition 5.21 (must re-generated marking set for delPlace MRG_E)
For a *temporal occurrence graph* $\underline{OG}_E$ and $\underline{M}$, the set of all markings in $\underline{OG}_E$. $\underline{OG}_E$ might have a *must re-generated marking set* $\mathbf{MRG_E} \subseteq \underline{M}$:

$$MRG_E = \{ \underline{m}_i \in \underline{M} | \underline{m}_i[\bullet(p_{tar}\bullet)]>0\}.$$

Since $\underline{m}_1[p_4]=1$ and $\underline{m}_3[p_4]=1$, MRG_E in Figure 5.5 is $\{\underline{m}_1, \underline{m}_3\}$.

5.4.2 Updating an Occurrence graph and the time complexity

The following algorithm presents a method to update the corresponding O-graph *OG'* when a place is deleted from a Petri net. Based on Definition 5.17, Algorithm 5.10 modifies *OG* for *OG*'.

Algorithm 5.9 ***Main_DP* (*PN*, *OG*, p_{tar}) → *PN'* and *OG'***

```
// Input     PN: the original net
//           OG: the O-graph of PN
//           p_tar: a place in PN
// Output    PN': the refined net
//           OG': the O-graph of PN'
```
● $\mathbf{M_{nr}} \leftarrow \S$: a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. M_{nr} contains those markings for which

```
we have not yet found the successors.
● **$MRG_E$** ← § : a set of markings as in Definition 5.21.
● **$\underline{OG_E}$** : a temporal occurrence graph. $\underline{V}$ ← § and $\underline{A}$ ← § .
● **$OG'$** : an occurrence graph. $V'$ ← § and $A'$ ← § .

1   delete place $p_{tar}$ and the connected arcs     // construct $PN'$
2   $ModifyOG\_DP$                                     // Algorithm 5.10
3   $Findmarking$                                      // Algorithm 4.4
4   $ExtendOG$                                         // Algorithm 4.5
```

Line 1 in Algorithm 5.9 constructs the refined Petri net *PN'*. Line 2 calls the function *ModifyOG_DP* (Algorithm 5.10) to modify the state space in *OG*. The details of each step will be presented below.

```
Algorithm 5.10    **ModifyOG_DP**

1   $ATF' \leftarrow ATF$
2   $A' \leftarrow A$
3   for all $v_i \in V$ do
4   begin
5       $NewV(v_i', v_i)$
6       $DPT(m_i', p_{tar})$
7       if($\exists m_j' \in M'$ where $m_i' = m_j'$)
8           if($m_i[p_{tar}] > m_j[p_{tar}]$)
9               $Merge(v_i', v_j')$
10          else
11              $Merge(v_j', v_i')$
12          endif
13      else if( $\underline{m}_i[{}^\bullet(p_{tar}{}^\bullet)] > 0$ )
14          $MRG_E \leftarrow MRG_E \cup \{\underline{m}_i\}$
15      endif
16  endfor
```

In Algorithm 5.10, the loop from line 3 to 16 selects a node v_i in *V*. In each turn, line 6 modifies v_i as described in Definition 5.20(1). Some of the markings after modification would be duplicated and codes from line 8 to 12 deal with this situation by calling *Merge* as in Algorithm 4.3 with corresponding parameters. After *Merge*, the node represented by the first parameter and its connected arc(s) are deleted. Otherwise, if $\underline{m}_i[\bullet(p_{tar}\bullet)]>0$, $\underline{m}_i$ is added to MRG_E in line 14. For example, when the loop from line 3 to 16 selects v_1, line 7 modifies $\underline{m}_1$ as (0,1,0,1,0).

Let t be the number of transitions in the Petri net, n be the number of vertexes in the input O-graph, and n' be the number of vertexes in the result O-graph. In Algorithm 5.10, it is assumed that all markings in the O-graph of the original net should be modified one time. Since the loop from line 3 to 16 will run n times, the complexity of Algorithm 5.10 is $O(n)$. Thus, the complexity of Algorithm 5.9 is $O(n + |n' - n|*t)$. Since designers usually edit the Petri nets sequentially, the complexity of Algorithm 5.9 is $O(n + t)$.

5.5 Place Mergence

This section discusses the maintenance of the O-graph for *Place* Mergence.

5.5.1 Scenario of Place Mergence

Let a relation which belongs to $P\times P$ be named as a *MPR* relation. The mergePlace example in section 3.3 can be translated into: *PN* is transformed into *PN'* by applying *mergePlace* with a *MPR* relation (p_4, p_1). A formal definition of a *MPR* addition is described in Definition 5.22.

Definition 5.22 (*MPR* mergence)
Let $PN = (P, T, F, m_0)$ be a Petri net. A Petri net $PN' = (P', T', F', m'_0)$ constructed by applying a *MPR* relation $mpr = (p_{from}, p_{to})$ into *PN* has the following properties:

(1) $P' = P \setminus \{ p_{from} \}$;

(2) $T' = T$;

(3) $F' = F \setminus (F_{oldt} \cup F_{oldh}) \cup (F_{newt} \cup F_{newh})$, where
$F_{oldt} = \{ f = (t_i, p_j) \in F \mid p_j = p_{from} \}$,
$F_{oldh} = \{ f = (p_i, t_j) \in F \mid p_i = p_{from} \}$,
$F_{newt} = \{ f = (t_i, p_{to}) \mid \exists (t_i, p_{from}) \in F \}$, and
$F_{newh} = \{ f = (p_{to}, t_j) \mid \exists (p_{from}, t_j) \in F \}$;

(4) $m_0' = m_0 + m_0[p_{from}] * PCV.p_{to}$, and then
$DPT(m_0', p_{from})$.

In a *MPR* mergence, both the original net *PN* and the refined net *PN'* have the same transitions, but *PN* has one more place representing the *MPR* relation merged. Also, item (3) shows the redirections described in section 3.1. The tokens in p_{from} and p_{to} of the original net are put in p_{to} of the refined net.

Definition 5.23 (state space mapping function for mergePlace SSM_E)
With Definition 5.22 and *M*', the set of all markings of *PN'*. The *state space mapping function* $\boldsymbol{SSM_E}$ is a function defined from *M* into *M'*, and $mpr = (p_{from}, p_{to})$:
$\forall m_i \in M$: $\boldsymbol{SSM_E}(m_i) = m_i + m_i[p_{from}] * PCV.p_{to}$

SSM_E is a function which maps the reachable markings of *PN* to the reachable markings of *PN'*. For example, while using marking m_4=(0,**1**,0,0,1,0) in Table A3.19(a) as input, SSM_E outputs marking m_4' = (0,2,0,0,0) = $m_4 + m_4[p_4] * PCV.p_1$ = (0,**1**,0,0,1,0) + 1 * (0,0,0,0,**1**,0), as shown in Table A3.20(a).

Definition 5.24 (arc mapping function for mergePlace R_E)
Let E = *mergePlace* be defined as in Definition 5.23, and M' be the set of all markings of PN'. The *arc mapping function* $\boldsymbol{R_E}$ is a function defined from A into $M' \times T' \times M'$, and $mpr = (p_{from}, p_{to})$: $\forall a_k = (m_i, t_n, m_j) \in A$:

$\boldsymbol{R_E}(a_k) = (\ DPT(SSM_E(m_i), p_{from})\ , t_n, DPT(SSM_E(m_j), p_{from})\ )$.

For example, marking m_3 can enable the firing of transition t_0, i.e. $a_6=(m_3, t_0, m_6)$ in Table A3.19(b). After *arc mapping function* R_E is applied with inputs a_6, $R_E(a_6) = (m'_3, t_0, m'_6)$, as shown in Table A3.20(b).

Definition 5.25 (temporal occurrence graph for mergePlace $\underline{OG}_E$)
Let E = *mergePlace* be defined as in Definition 5.23, and $\underline{M}$ be the set of all markings generated from SSM_E. The *temporal occurrence graph* $\underline{OG}_E$ is the directed graph $\underline{OG}_E = (\underline{V}, \underline{A})$ with $mpr = (p_{from}, p_{to})$, where

(1) $\forall v_i \in V, \underline{v}_i \in \underline{V}$: $\underline{m}_i = (\ DPT(SSM_E(m_i), p_{from})$,

(2) $\underline{A} = \{ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{M} \times T \times \underline{M} \mid \underline{m}_i\ [\ t_n > \underline{m}_j \}$, and

(3) $\forall a_k \in A, \underline{a}_k \in \underline{A}$: $\underline{a}_k = R_E(a_k)$.

For example, Figure 5.6 is the $\underline{OG}_E$ after DPT, SSM_E, and R_E are applied with input OG in Figure 3.41.

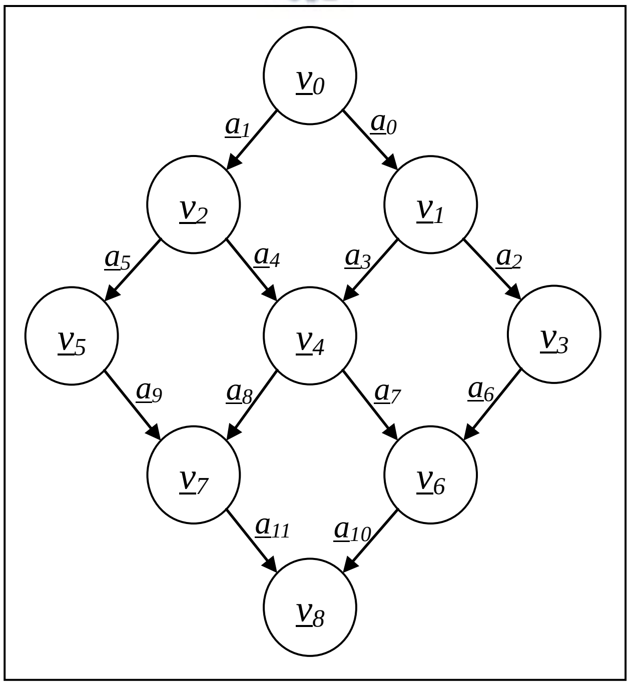


Figure 5.6 The temporal occurrence graph $\underline{OG}_E$ modified from Figure 3.41.

Definition 5.26 (must re-generated marking set for mergePlace MRG_E)
For a *temporal occurrence graph* $\underline{OG_E}$ and $\underline{M}$, the set of all markings in $\underline{OG_E}$. $\underline{OG_E}$ might have a *must re-generated marking set* $\mathbf{MRG_E} \subseteq \underline{M}$, $mpr = (p_{from}, p_{to})$:

$$MRG_E = \{ \underline{m_i} \in \underline{M} | \ \underline{m_i}[p_{to}]>0 \}.$$

MRG_E in Figure 5.6 is $\{\underline{m_1}, \underline{m_2}, \underline{m_4}, \underline{m_6}, \underline{m_7}\}$.

Proposition 5.1

Let E = *mergePlace* be defined as in Definition 5.23, C be the incidence matrix of PN, C' be the incidence matrix of PN', and a *temporal occurrence graph* $\underline{OG_E}$ be defined as in Definition 5.25. Then,
(1) $v_0 = v_0'$,
(2) $\underline{V} \subseteq V'$, and
(3) $\underline{A} \subseteq A'$.

Proof:

(1) Since the initial markings of PN and PN' are the same, $v_0 = v_0'$.

(2) For each node $\underline{v_i}$ in $\underline{V}$,

$$\underline{m_i} = DPT(SSM_E(m_i), p_{from}).$$

Since the firing sequence of $\vec{\sigma}_i^*$ can be fired in PN', there is a marking

$$m_i' = m_0' + C' * \vec{\sigma}_i^*.$$

Because there might be some token(s) generated after mergence from p_{from} to p_{to}, the difference between $m_i[p_{to}]$ and $m_i'[p_{to}]$ is $m_i[p_{from}]$. Besides, the rest of m_i are equivalent to those of m_i', i.e.

$$m_i' = DPT(SSM_E(m_i), p_{from}).$$

According to Definition 5.23,

$$\underline{m_i} = DPT(SSM_E(m_i), p_{from}) \text{ is } m_i'.$$

From Definition 5.25 and $\underline{\vec{\sigma}_i^*} = \vec{\sigma}_i^{*\prime}$, $\underline{v_i} = v_i'$.

Thus, and $\underline{V}$ is the subset of V'.

(3) For each arc $\underline{a}_k = (\underline{m}_i, t_n, \underline{m}_j)$ in $\underline{A}$, there is a node $v_i' \in V'$ and a marking $m_i' = \underline{m}_i$ based on the discussions in (2). Since $m_i'[p_{to}] \geqq m_i[p_{to}]$ and the token numbers in the rest places do not change, transition t_n enabled at m_i can also be fired at m_i'. From Definition 5.25(1), another marking $m_j \in M$ can be modified as

$$\underline{m}_j = DPT(SSM_E(m_j), p_{from}).$$

From Definition 2.4,

$$m_j = m_i + C * TCV.t_n.$$

From Definition 4.2,

$$\vec{\sigma}_j^* = \vec{\sigma}_i^* + TCV.t_n.$$

Therefore, $\underline{m}_j = DPT(SSM_E(m_i), p_{from}) + DPT(C * TCV.t_n + (C * TCV.t_n)[p_{from}] * PCV.p_{to}, p_{from})$. Because $(C * TCV.t_n)[p_{from}] * PCV.p_{to}$ is the number of new tokens in the place p_{to}, $C' * TCV.t_n = DPT(C * TCV.t_n + (C * TCV.t_n)[p_{from}] * PCV.p_{to}, p_{from})$.

Thus, there exists an arc $a_k' = (m_i', t_n, m_j') \in A'$ and $\underline{m}_j = m_i' + C' * TCV.t_n = m_j'$.

Hence, $\underline{a}_k = a_k'$ and $\underline{A}$ is the subset of A'.

Proposition 5.2

Let $E = mergePlace$ be defined as in Proposition 5.1. Then, $\forall\ a_k' = (m_i', t_n, m_j') \in A'$:

(1) if $m_i' \notin \underline{M}$, then $a_k' \notin \underline{A}$.

(2) if $m_i' \in MRG_E$ and $\nexists\ (m_i, t_n, m_j) \in A$, then $a_k' \notin \underline{A}$.

(3) if $m_i' \in MRG_E$ and $\exists\ (m_i, t_n, m_j) \in A$, then $a_k' \in \underline{A}$.

(4) if $m_i' \in \underline{M}$ and $m_i' \notin MRG_E$, then $a_k' \in \underline{A}$.

Proof:

(1) If m_i' does not exist in $\underline{OG_E}$, the arcs starting from it to m_j' can not occur in $\underline{OG_E}$.

(2) Because $\nexists\ (m_i, t_n, m_j) \in A$ and Definition 5.25(3), $\nexists\ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{A}$. Thus, $a_k' \notin \underline{A}$.

(3) Since $\exists\ (m_i, t_n, m_j) \in A$ and Definition 5.25(3), $\exists\ (\underline{m}_i, t_n, \underline{m}_j) \in \underline{A}$. Thus, $a_k' \in \underline{A}$.

(4) Because $m_i' \notin MRG_E$, $\underline{m}_i[p_{to}]>0$. Hence, m_i can enable all transitions which m_i' can enable. Since m_i can enable all transitions which m_i' can enable, each arc from m_i' belongs to $\underline{A}$.

We use the mergePlace example in section 3.3 to explain Proposition 5.2. In the example:

(1) Because $a_{14}'=(m_9',t_0,m_7')$ and $m_9' \notin \underline{M}$, $a_{14}' \notin \underline{A}$.

(2) $m_1' \in MRG_E$, $a_2=(m_1,t_3,m_3)$, and $a_3=(m_1,t_0,m_4)$ cause $a_{12}'=(m_1',t_1,m_9') \notin \underline{A}$.

(3) $m_1' \in MRG_E$ and $a_3=(m_1,t_0,m_4)$ cause $a_3'=(m_1',t_0,m_4') \in \underline{A}$.

(4) Since $m_3' \in \underline{M}$ and $m_3' \notin MRG_E$, $a_6'=(m_3',t_0,m_6') \in \underline{A}$.

Proposition 5.2(1) and (2) detect the arcs which do not belong to $\underline{A}$. Proposition 5.2 certifies that the detections are complete.

5.5.2 Updating an Occurrence graph and the time complexity

Algorithm 5.11 presents a method to update the corresponding O-graph *OG'* for the mergence of a *MPR* relation. Based on Definition 5.22, Algorithm 5.11 firstly modifies *OG* for $\underline{OG}_E$ (Definition 5.25), and then extends $\underline{OG}_E$ for the O-graph *OG'* = (*V'*, *A'*).

```
Algorithm 5.11      Main_MP (PN, OG, mpr) → PN' and OG'
// Input     PN: the original net
//           OG: the O-graph of PN
//           mpr: a MPR relation
// Output    PN': the refined net with mpr
//           OG': the O-graph of PN'
● $M_{nr} \leftarrow \S$ : a set of markings. $M_{nr} \subseteq M' \setminus \underline{M}$. $M_{nr}$ contains those markings for which
  we have not yet found the successors.
```

- ***MRG*** $_E \leftarrow \S$: a set of markings as in Definition 5.26.
- ***OG*** $\underline{OG}_E$: a temporal occurrence graph. $\underline{V} \leftarrow \S$ and $\underline{A} \leftarrow \S$.
- ***OG'*** : an occurrence graph. $V' \leftarrow \S$ and $A' \leftarrow \S$.

```
1   delete place to $p_{src}$                          // construct $PN'$
2   the tokens in $p_{from}$ and $p_{to}$ of $PN$ are put in $p_{to}$ of $PN'$
3   redirect the arcs connected with $p_{from}$
4   ModifyOG_MP                                        // Algorithm 5.12
5   Findmarking                                        // Algorithm 4.4
6   ExtendOG                                           // Algorithm 4.5
```

Line 1 to line 3 in Algorithm 5.11 constructs the refined Petri net *PN'*. Line 4 calls the function *ModifyOG_MP* (Algorithm 5.12) to modify the state space in *OG*. Line 6 calls the function *ExtendOG* (Algorithm 4.5) to generate the new nodes and arcs whose ancestors are in M_{nr}. The details of each step will be presented below.

Algorithm 5.12 ***ModifyOG_MP***

```
1   $\underline{ATF} \leftarrow ATF$
2   $\underline{A} \leftarrow A$
3   for all $v_i \in PmapV(p_{from}, p_{to})$ do
    //the nodes affected by $mergePlace(PN, p_{from}, p_{to})$
4   begin
5       $NewV(\underline{v}_i, v_i)$
6       $\underline{m}_i \leftarrow \underline{m}_i + \underline{m}_i[p_{from}] * PCV.p_{to}$          // Definition 5.23
7       $DPT(\underline{m}_i, p_{from})$                                 // Definition 5.18
8       if($\exists \underline{m}_j \in \underline{M}$ where $\underline{m}_i = \underline{m}_j$)
9           $Merge(\underline{v}_i, \underline{v}_j)$
10      else if ($\underline{m}_i[p_{to}]>0$)                          // Definition 5.26
11          $MRG_E \leftarrow MRG_E \cup \{\underline{m}_i\}$
12      endif
13  endfor
```

In Algorithm 5.12, the loop from line 3 to 13 selects a node v_i in *PmapV* (p_{from}, p_{to}). In each turn, lines 6 and 7 modify the marking of v_i as described in Definition 5.25(1). Some of the markings after modification would be duplicated and codes from line 8 to 10 deal with this situation by calling *Merge* as in Algorithm 4.3 with corresponding parameters. After *Merge*, the node represented by the first parameter and its connected arc(s) are deleted. Otherwise, if $\underline{m}_i[p_{to}]>0$, $\underline{m}_i$ is added to MRG_E in line 11.

For example, when the loop from line 3 to 13 selects v_4, lines 6 and 7 modify $\underline{m}_4$ as $\underline{m}_4 + \underline{m}_4[p_4] * PCV.p_1$. Since $\underline{m}_4[p_1]>0$, $\underline{m}_4$ is added to MRG_E in line 11.

Proposition 5.3

Assume that the mergence from p_{from} to p_{to} is applied in *PN* and Algorithm 4.3 merges $\underline{v}_x$ and $\underline{v}_y$ correctly. Algorithm 5.12 modifies *OG* as $\underline{OG}_E$ and generates MRG_E correctly, i.e., $\underline{OG}_E$ as defined in Definition 5.25 and MRG_E as defined in Definition 5.26.

Proof:

The temporal occurrence graph defined in Definition 5.25 is calculated from line 3 to 13. Algorithm 4.3 merges $\underline{v}_x$ and $\underline{v}_y$ correctly. Otherwise, if $\underline{m}_i[p_{to}]>0$, $\underline{m}_i$ is added to MRG_E. Hence, the correctness of Algorithm 5.12 holds.

Theorem 5.1 (Correct O-graph Modification and Extension)

Algorithm 5.11 generates *OG'* correctly when merging p_{from} and p_{to}, i.e. *OG'* has the same characteristic as the one constructed from *PN'* directly.

Proof:

From Proposition 5.3, Algorithm 5.12 modifies *OG* as $\underline{OG}_E$ correctly. From Proposition 5.1, $\underline{OG}_E$ is the subnet of *OG'*. From 4.8 and 4.9, the nodes and arcs generated by Algorithm 4.4 and 4.5 all belong to *OG'* and the nodes and arcs belonging to *OG'* but not $\underline{OG}_E$ are all generated by Algorithm 4.4 and 4.5. Hence, the correctness of Algorithm 5.11 holds.

Let *t* be the number of transitions in the Petri net, *n* be the number of vertexes in the input O-graph, and *n’* be the number of vertexes in the result O-graph. In algorithm 5.12, it is assumed that all markings in the O-graph of the original net should be modified one time. Thus, the loop from line 3 to 13 will run *n* times. Therefore, the complexity of Algorithm 5.12 is $O(n)$. The complexity of Algorithm 5.11 is $O(n + |n' - n| * t)$. Since designers usually edit the Petri nets sequentially, the complexity of Algorithm 5.11 is $O(n + t)$.

Chapter 6. Comparison

This thesis gives a quick approach to maintain the occurrence graph immediately after applying a basic editing action on the Petri net. Let the original net *PN* (e.g., Figure 3.4) be the Petri net at present and O-graph of *PN* be *OG*. Assume that a designer edits *PN* with a basic editing action in Table 3.1 (e.g., adding a new arc from transition t_2 to place p_4) and the refined net is called *PN'*. The occurrence graph after being modified and extended in our editor is called *OG'* (e.g., Figure 3.7).

Basic edit action	Vertexes modifications	MRG_E
$addPtoTarc(PN, p_{src}, t_{dst})$	$\underline{m}_i = m_i - \vec{\sigma}_i^*[t_{dst}] * PCV.p_{src}$	No
$addTtoParc(PN, t_{src}, p_{dst})$	$\underline{m}_i = m_i + \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst}$	$\vec{\sigma}_i^*[t_{src}]>0$
$addTrans(PN, p_{src}, t_{new}, p_{dst})$	$\underline{m}_i = m_i$ and $\underline{\vec{\sigma}}_i^* = ATT(\vec{\sigma}_i^*, t_{new})$	$\underline{m}_i[p_{src}]>0$
$addPlace(PN, p_{new})$	$\underline{m}_i = APT(m_i, p_{new})$	No
$addToken(PN, p_{tar})$	$\underline{m}_i = m_i + PCV.p_{tar}$	$\underline{M}$
$delTransition(PN, t_{tar})$	$\underline{m}_i = m_i$ and $\underline{\vec{\sigma}}_i^* = DTT(\underline{\vec{\sigma}}_i^*, t_{tar})$	No
$delTtoParc(PN, t_{src}, p_{dst})$	$\underline{m}_i = m_i - \vec{\sigma}_i^*[t_{src}] * PCV.p_{dst}$	No
$delPtoTarc(PN, p_{src}, t_{dst})$	$\underline{m}_i = m_i + \vec{\sigma}_i^*[t_{dst}] * PCV.p_{src}$	$\vec{\sigma}_i^*[t_{dst}]>0$ or $\underline{m}_i[\bullet t_{dst}]>0$
$delToken(PN, p_{tar})$	$\underline{m}_i = m_i - PCV.p_{tar}$	No
$delPlace(PN, p_{tar})$	$\underline{m}_i = DPT(m_i, p_{tar})$	$\underline{m}_i[\bullet(p_{tar}\bullet)]>0$
$mergeTrans(PN, t_{from}, t_{to})$	$\underline{m}_i = m_i$ and $\underline{\vec{\sigma}}_i^* = DTT(\underline{\vec{\sigma}}_i^*, t_{from})$	-
$mergePlace(PN, p_{from}, p_{to})$	$SSM_E(m_i) = m_i + m_i[p_{from}] * PCV.p_{to}$ $\underline{m}_i = (DPT(SSM_E(m_i), p_{from})$	$\underline{m}_i[p_{from}]>0$

Table 6.1 Vertexes modifications and MRG_E for basic edit actions

Table 6.1 shows the modification rules and the MRG_E sets for each basic edit action. *addPtoTarc*, *addPlace*, *delTransition*, *delTtoParc*, and *delToken* do not have to extend O-graphs.

Basic edit action	*ATF*	Time complexity
$addPtoTarc(PN, p_{src}, t_{dst})$	Yes	$O(n')$
$addTtoParc(PN, t_{src}, p_{dst})$	Yes	$O(n+\|n'-n\|*t)$
$addTrans(PN, p_{src}, t_{new}, p_{dst})$	No	$O(n+\|n'-n\|*t)$
$addPlace(PN, p_{new})$	No	$O(n')$
$addToken(PN, p_{tar})$	No	$O(n+\|n'-n\|*t)$
$delTransition(PN, t_{tar})$	Yes	$O(n')$
$delTtoParc(PN, t_{src}, p_{dst})$	Yes	$O(n')$
$delPtoTarc(PN, p_{src}, t_{dst})$	Yes	$O(n+\|n'-n\|*t)$
$delToken(PN, p_{tar})$	No	$O(n')$
$delPlace(PN, p_{tar})$	No	$O(n+\|n'-n\|*t)$
$mergeTrans(PN, t_{from}, t_{to})$	-	$O(n')$
$mergePlace(PN, p_{from}, p_{to})$	No	$O(n+\|n'-n\|*t)$

Table 6.2 *ATF* and time complexity for basic edit actions

Table 6.2 shows whether each basic edit action needs *ATF* and time complexity for each basic edit action. *addPtoTarc, addTtoParc*, *delTransition*, *delTtoParc,* and *delPtoTarc* need to deal with the arcs in *ATF*.

Now, algorithms in section 4 and 5 can be compared with the original algorithm ([2]) that constructs the O-graph directly from its corresponding Petri net.

```
Algorithm 6.1
// Input     PN: a Petri net
// Output    OG: the O-graph of PN

1   Waiting ← ∅
2   Node($m_0$)
3   WHILE (Waiting ≠ ∅)
4       select a node $v_1 \in$ Waiting
5       FOR ALL $(t, v_2) \in$ Next($v_1$)
6           Node($v_2$)
7           Arc($v_1$, $t$, $v_2$)
8       END
9       Waiting ← Waiting \ {$v_1$}
10  END
```

Node(v) is a function. If v exists in the O-graph already, the function does nothing. Otherwise, the function creates a new node v and adds v into *Waiting*, a set of nodes, and the O-graph. *Next*(v_1) is used to denote the set of all possible "next moves" from v_1, i.e., *Next*(v_1) = $\{(t, v_2) \in T \times V \mid v_2$ is the marking after firing t on $v_1\}$. *Arc*(v_1, t, v_2) is a function that creates a new arc (v_1, t, v_2) and adds it to the O-graph.

Let n' be the number of nodes in the result O-graph and t be the number of transitions in the Petri net. Each node contained in the result O-graph should be added into *Waiting*. The loop from line 3 to 10 selects a node in *Waiting* and generates the arcs and nodes correspondingly. In line 5, since each transition in the result net must be examined whether it is enabled by v_1 or not, the complexity of the examination is $O(t)$. The function *Node*(v) checks if v is already in the O-graph and the complexity of this search is $O(1)$. Thus, the complexity of Algorithm 6.1 is $O(n' * t)$. This complexity is also shown in [17].

Chapter 7. Conclusion and Future Works

The major contribution of this thesis is to introduce a method that reduces the O-graph building time by maintaining instead of re-constructing the O-graph when an edit action is done. O-graph building time reduction contributes to the incremental analyses of Petri net. To simplify the discussions, the thesis introduces a general editor containing three groups of edit actions for designers to draw Petri nets.

We are currently continuing our research in several directions. First, there are many kinds of edit actions which section 3 does not mention. Some of these actions can be done by combining basic edit actions. The rest can be discussed with the idea of this thesis. Second, we will adapt our method to handle high-level Petri nets, such as coloured Petri nets ([1],[2], and [12])and timed Petri nets ([12]). Third, incremental analysis for Petri Net based on occurrence graph can be improved by applying the modification and extension discussed in our work.

Reference

[1] Kurt Jensen, "Coloured Petri Nets: Basic Concepts," Springer, 1992.

[2] Kurt Jensen, "Coloured Petri Nets: Analysis Methods," Springer, 1995.

[3] S. Christensen and L. Petrucci, "Modular Analysis of Petri Nets", *The Computer Journal*, 43(3):224–242, 2000.

[4] G. Lewis and C. Lakos. "Incremental state space construction for coloured Petri nets". In *ICATPN'01*, volume 2075 of *LNCS*, pages 263–282. Springer, 2001.

[5] Rinderle, S., Reichert, M., Dadam, P.: "On dealing with structural conflicts between process type and instance changes". In: Proc. BPM'04. (2004) 274–289

[6] Poitr Chrza¸stowski-Wachtel, Boualem Benatallah, Rachid Hamadi, Milton O'Dell, and Adi Susanto. "A top-down petri net-based approach for dynamic workflow modeling". In W.M.P van der Aalst et al., editor, *International Conference on Business Process Management (BPM'03)*, volume 2678 of *LNCS*, pages 336–353, Eindhoven, The Netherlands, June 2003.

[7] A. S. Miner and G. Ciardo. "Efficient reachability set generation and storage using decision diagrams". In H. Kleijn and S. Donatelli, editors, *Application and Theory of Petri Nets 1999, Lecture Notes in Computer Science 1639 Proc.20th Int. Conf. on Applications and Theory of Petri Nets, Williamsburg, VA, USA*. Springer-Verlag, June 1999. To appear.

[8] Y.-H Lin, "A Technique for Reducing Occurrence Graph Building Time", National Chiao-Tung University, M.S. Thesis, 2007.

[9] Giovanni Chiola and Ruben Carvajal-Schiaffino, "A Reachability Graph Construction Algorithm Based on Canonical Transition Firing Count Vectors", IEEE, 2001.

[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. Reisig and G. Rozenberg, “Lectures on Petri Nets I: Basic Models,” Springer, 1998.

[12] W. Reisig and G. Rozenberg, “Lectures on Petri Nets II: Applications,” Springer, 1998.

[13] Jörg Desel and Gabriel Juhás, “What Is a Petri Net,” Unifying Petri Nets, Lecture Notes in Computer Science, pp. 1-25, Springer, 2001.

[14] T. Murata, “Petri Nets: Properties, Analysis and Applications,” Proceedings of the IEEE, vol. 77, issue 4, pp. 541 – 580, April 1989.

[15] C.A. Petri, “Kommunikation mit Automaten,” PhD thesis, University of Bonn, Bonn, Germany, 1962.

[16] http://pipe2.sourceforge.net/

[17] 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.

Appendix

addTtoParc

V			*A*		
v_i	m_i	$\vec{\sigma}_i^*$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,0,0,0,0,0,0,0,0)	(0,0,0,0,0,0,0,0,0,0)	a_0	(m_0,t_2,m_1)	false
v_1	(0,0,0,1,**0**,0,0,0,0)	(0,0,**1**,0,0,0,0,0,0,0)	a_1	(m_0,t_1,m_2)	false
v_2	(0,0,1,0,0,0,0,0,0)	(0,1,0,0,0,0,0,0,0,0)	a_2	(m_0,t_0,m_3)	false
v_3	(0,1,0,0,0,0,0,0,0)	(1,0,0,0,0,0,0,0,0,0)	a_3	(m_1,t_6,m_4)	false
v_4	(0,0,0,0,**0**,0,1,0,0)	(0,0,**1**,0,0,0,1,0,0,0)	a_4	(m_1,t_5,m_5)	false
$\boldsymbol{v_5}$	(0,0,0,0,**0**,1,0,0,0)	(0,0,**1**,0,0,1,0,0,0,0)	a_5	(m_2,t_4,m_5)	true
v_6	(0,0,0,0,1,1,0,0,0)	(1,0,0,1,0,0,0,0,0,0)	a_6	(m_3,t_3,m_6)	false
$\boldsymbol{v_7}$	(0,0,0,0,**0**,0,0,0,1)	(0,0,**1**,0,0,1,0,0,1,0)	a_7	(m_4,t_9,m_7)	true
v_8	(0,0,0,0,1,0,0,0,1)	(1,0,0,1,0,0,0,0,1,0)	$\boldsymbol{a_8}$	(m_5,t_8,m_7)	false
v_9	(0,0,0,0,0,1,0,1,0)	(1,0,0,1,0,0,0,1,0,0)	$\boldsymbol{a_9}$	(m_6,t_8,m_8)	false
v_{10}	(0,0,0,0,0,0,0,1,1)	(1,0,0,1,0,0,0,1,1,0)	a_{10}	(m_6,t_7,m_9)	false
			a_{11}	(m_8,t_7,m_{10})	false
			a_{12}	(m_9,t_8,m_{10})	false

Table A3.1 The nodes (a) and arcs (b) of Figure 3.5

V′			*A′*		
v_i'	m_i'	$\vec{\sigma}_i^{*\prime}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(1,0,0,0,0,0,0,0,0)	(0,0,0,0,0,0,0,0,0,0)	a_0'	(m_0',t_2,m_1')	false
v_1'	(0,0,0,1,**1**,0,0,0,0)	(0,0,**1**,0,0,0,0,0,0,0)	a_1'	(m_0',t_1,m_2')	false
v_2'	(0,0,1,0,0,0,0,0,0)	(0,1,0,0,0,0,0,0,0,0)	a_2'	(m_0',t_0,m_3')	false
v_3'	(0,1,0,0,0,0,0,0,0)	(1,0,0,0,0,0,0,0,0,0)	a_3'	(m_1',t_6,m_4')	false
v_4'	(0,0,0,0,**1**,0,1,0,0)	(0,0,**1**,0,0,0,1,0,0,0)	a_4'	(m_1',t_5,m_6')	true
v_6'	(0,0,0,0,1,1,0,0,0)	(1,0,0,1,0,0,0,0,0,0)	a_6'	(m_3',t_3,m_6')	false
v_8'	(0,0,0,0,1,0,0,0,1)	(1,0,0,1,0,0,0,0,1,0)	a_7'	(m_4',t_9,m_8')	true
v_9'	(0,0,0,0,0,1,0,1,0)	(1,0,0,1,0,0,0,1,0,0)	$\boldsymbol{a_9}$	(m_6',t_8,m_8')	false
v_{10}'	(0,0,0,0,0,0,0,1,1)	(1,0,0,1,0,0,0,1,1,0)	a_{10}'	(m_6',t_7,m_9')	false
$\boldsymbol{v_{11}'}$	(0,0,0,1,0,0,0,1,0)	(0,0,1,0,0,0,0,1,0,0)	a_{11}'	(m_8',t_7,m_{10}')	false
$\boldsymbol{v_{12}'}$	(0,0,0,0,0,1,0,0,0)	(0,1,0,0,1,0,0,0,0,0)	a_{12}'	(m_9',t_8,m_{10}')	false
$\boldsymbol{v_{13}'}$	(0,0,0,0,0,0,1,1,0)	(0,0,1,0,0,0,1,1,0,0)	a_{13}'	(m_1',t_7,m_{11}')	false
$\boldsymbol{v_{14}'}$	(0,0,0,0,0,0,0,0,1)	(0,1,0,0,1,0,0,0,1,0)	a_{14}'	(m_2',t_4,m_{12}')	false
			a_{15}'	(m_4',t_7,m_{13}')	true
			a_{16}'	(m_{11}',t_5,m_9')	true

a'_{17}	(m'_{11},t_6,m'_{13})	false
a'_{18}	(m'_{12},t_8,m'_{14})	false
a'_{19}	(m'_{13},t_9,m'_{10})	true

Table A3.2 The nodes (a) and arcs (b) of Figure 3.7

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\underline{\vec{\sigma}_i^*}$	$\underline{a}_k$	$(\underline{m}_i,t_n,\underline{m}_j)$	$\underline{atf}_k$
$\underline{v}_0$	(1,0,0,0,0,0,0,0,0)	(0,0,0,0,0,0,0,0,0,0)	$\underline{a}_0$	$(\underline{m}_0,t_2,\underline{m}_1)$	false
$\underline{v}_1$	(0,0,0,1,**1**,0,0,0,0)	(0,0,1,0,0,0,0,0,0,0)	$\underline{a}_1$	$(\underline{m}_0,t_1,\underline{m}_2)$	false
$\underline{v}_2$	(0,0,1,0,0,0,0,0,0)	(0,1,0,0,0,0,0,0,0,0)	$\underline{a}_2$	$(\underline{m}_0,t_0,\underline{m}_3)$	false
$\underline{v}_3$	(0,1,0,0,0,0,0,0,0)	(1,0,0,0,0,0,0,0,0,0)	$\underline{a}_3$	$(\underline{m}_1,t_6,\underline{m}_4)$	false
$\underline{v}_4$	(0,0,0,0,**1**,0,1,0,0)	(0,0,1,0,0,0,1,0,0,0)	$\underline{a}_4$	$(\underline{m}_1,t_5,\boldsymbol{\underline{m}_6})$	true
$\boldsymbol{\underline{v}_6}$	(0,0,0,0,1,1,0,0,0)	(1,0,0,1,0,0,0,0,0,0)	$\underline{a}_6$	$(\underline{m}_3,t_3,\underline{m}_6)$	false
$\boldsymbol{\underline{v}_8}$	(0,0,0,0,1,0,0,0,1)	(1,0,0,1,0,0,0,0,1,0)	$\underline{a}_7$	$(\underline{m}_4,t_9,\boldsymbol{\underline{m}_8})$	true
$\underline{v}_9$	(0,0,0,0,0,1,0,1,0)	(1,0,0,1,0,0,0,1,0,0)	$\underline{a}_9$	$(\underline{m}_6,t_8,\underline{m}_8)$	false
$\underline{v}_{10}$	(0,0,0,0,0,0,0,1,1)	(1,0,0,1,0,0,0,1,1,0)	$\underline{a}_{10}$	$(\underline{m}_6,t_7,\underline{m}_9)$	false
			$\underline{a}_{11}$	$(\underline{m}_8,t_7,\underline{m}_{10})$	false
			$\underline{a}_{12}$	$(\underline{m}_9,t_8,\underline{m}_{10})$	false

Table A4.1 The nodes and arcs of Figure 4.1

addPtoTarc

V			A		
v_i	m_i	$\vec{\sigma}_i^*$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,1,0,0)	(0,0,0,0)	a_0	(m_0,t_2,m_1)	false
v_1	(1,0,0,1)	(0,0,1,0)	a_1	(m_0,t_1,m_2)	false
v_2	(0,1,1,0)	(0,1,0,0)	a_2	(m_0,t_0,m_3)	false
v_3	(0,2,0,0)	(1,0,0,0)	a_3	(m_1,t_1,m_4)	false
v_4	(0,0,1,1)	(0,1,1,0)	a_4	(m_1,t_0,m_5)	false
v_5	(0,1,0,1)	(1,0,1,0)	a_5	(m_2,t_3,m_5)	true
v_6	(0,**0**,0,2)	(0,1,1,**1**)	a_6	(m_2,t_2,m_4)	false
			a_7	(m_3,t_2,m_5)	false
			a_8	(m_4,t_3,m_6)	false
			a_9	(m_5,t_2,m_6)	true

Table A3.3 The nodes (a) and arcs (b) of Figure 3.9

V'			*A'*		
v_i'	m_i'	$\vec{\sigma}_i^{*'}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(1,1,0,0)	(0,0,0,0)	a_0'	(m_0',t_2,m_1')	false
v_1'	(1,0,0,1)	(0,0,1,0)	a_1'	(m_0',t_1,m_2')	false
v_2'	(0,1,1,0)	(0,1,0,0)	a_2'	(m_0',t_0,m_3')	false
v_3'	(0,2,0,0)	(1,0,0,0)	a_3'	(m_1',t_1,m_4')	false
v_4'	(0,0,1,1)	(0,1,1,0)	a_4'	(m_1',t_0,m_5')	false
v_5'	(0,1,0,1)	(1,0,1,0)	a_6'	(m_2',t_2,m_4')	false
v_6'	(0,**-1**,0,2)	(0,1,1,**1**)	a_7'	(m_3',t_2,m_5')	false
$\boldsymbol{v_7'}$	(0,0,0,1)	(0,1,0,1)	a_8'	(m_4',t_3,m_6')	false
$\boldsymbol{v_8'}$	(0,0,0,2)	(1,0,2,0)	$\boldsymbol{a_{10}'}$	(m_2',t_3,m_7')	false
			$\boldsymbol{a_{11}'}$	(m_5',t_2,m_8')	false

Table A3.4 The nodes (a) and arcs (b) of Figure 3.11

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\underline{\vec{\sigma}}_i^*$	$\underline{a}_k$	$(\underline{m}_i,t_n,\underline{m}_j)$	$\underline{atf}_k$
$\underline{v}_0$	(1,1,0,0)	(0,0,0,0)	$\underline{a}_0$	$(\underline{m}_0,t_2,\underline{m}_1)$	false
$\underline{v}_1$	(1,0,0,1)	(0,0,1,0)	$\underline{a}_1$	$(\underline{m}_0,t_1,\underline{m}_2)$	false
$\underline{v}_2$	(0,1,1,0)	(0,1,0,0)	$\underline{a}_2$	$(\underline{m}_0,t_0,\underline{m}_3)$	false
$\underline{v}_3$	(0,2,0,0)	(1,0,0,0)	$\underline{a}_3$	$(\underline{m}_1,t_1,\underline{m}_4)$	false
$\underline{v}_4$	(0,0,1,1)	(0,1,1,0)	$\underline{a}_4$	$(\underline{m}_1,t_0,\underline{m}_5)$	false
$\underline{v}_5$	(0,1,0,1)	(1,0,1,0)	$\underline{a}_6$	$(\underline{m}_2,t_2,\underline{m}_4)$	false
			$\underline{a}_7$	$(\underline{m}_3,t_2,\underline{m}_5)$	false

Table A4.2 The nodes and arcs of Figure 4.2

addToken

V			*A*		
v_i	m_i	$\vec{\sigma}_i^*$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,0,0,0)	(0,0,0)	a_0	(m_0,t_0,m_1)	false
v_1	(0,1,0,0)	(1,0,0)	a_1	(m_1,t_1,m_2)	false
v_2	(0,0,1,0)	(1,1,0)	a_2	(m_2,t_2,m_3)	false
v_3	(0,0,0,1)	(1,1,1)			

Table A3.5 The nodes (a) and arcs (b) of Figure 3.13

V′			A′		
$v_i^{'}$	$m_i^{'}$	$\vec{\sigma}_i^{*'}$	$a_k^{'}$	$(m_i^{'},t_n,m_j^{'})$	$atf_k^{'}$
$v_0^{'}$	(1,0,1,0)	(0,0,0)	$a_0^{'}$	$(m_0^{'},t_0,m_1^{'})$	false
$v_1^{'}$	(0,1,1,0)	(1,0,0)	$a_1^{'}$	$(m_1^{'},t_1,m_2^{'})$	false
$v_2^{'}$	(0,0,2,0)	(1,1,0)	$a_2^{'}$	$(m_2^{'},t_2,m_3^{'})$	false
$v_3^{'}$	(0,0,1,1)	(1,1,1)	$\boldsymbol{a_3^{'}}$	$(m_0^{'},t_2,m_4^{'})$	false
$\boldsymbol{v_4^{'}}$	(1,0,0,1)	(0,0,1)	$\boldsymbol{a_4^{'}}$	$(m_1^{'},t_2,m_5^{'})$	false
$\boldsymbol{v_5^{'}}$	(0,1,0,1)	(1,0,1)	$\boldsymbol{a_5^{'}}$	$(m_3^{'},t_2,m_6^{'})$	false
$\boldsymbol{v_6^{'}}$	(0,0,0,2)	(1,1,2)	$\boldsymbol{a_6^{'}}$	$(m_4^{'},t_0,m_5^{'})$	false
			$\boldsymbol{a_7^{'}}$	$(m_5^{'},t_1,m_3^{'})$	false

Table A3.6 The nodes (a) and arcs (b) of Figure 3.15

$\underline{V}$			$\underline{A}$		
$\underline{v_i}$	$\underline{m_i}$	$\underline{\vec{\sigma}_i^{*}}$	$\underline{a_k}$	$(\underline{m_i},t_n,\underline{m_j})$	$\underline{atf_k}$
$\underline{v_0}$	(1,0,**1**,0)	(0,0,0)	$\underline{a_0}$	$(\underline{m_0},t_0,\underline{m_1})$	false
$\underline{v_1}$	(0,1,**1**,0)	(1,0,0)	$\underline{a_1}$	$(\underline{m_1},t_1,\underline{m_2})$	false
$\underline{v_2}$	(0,0,**2**,0)	(1,1,0)	$\underline{a_2}$	$(\underline{m_2},t_2,\underline{m_3})$	false
$\underline{v_3}$	(0,0,**1**,1)	(1,1,1)			

Table A4.3 The nodes and arcs of Figure 4.3

addTrans

V			A		
v_i	m_i	$\vec{\sigma}_i^{*}$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(2,0,0,0)	(0,0,0)	a_0	(m_0,t_1,m_1)	false
v_1	(1,0,1,0)	(0,1,0)	a_1	(m_0,t_0,m_2)	false
v_2	(1,1,0,0)	(1,0,0)	a_2	(m_1,t_1,m_3)	false
v_3	(0,0,2,0)	(0,2,0)	a_3	(m_1,t_0,m_4)	false
v_4	(0,1,1,0)	(1,1,0)	a_4	(m_2,t_1,m_4)	false
v_5	(0,2,0,0)	(2,0,0)	a_5	(m_2,t_0,m_5)	false
v_6	(0,0,0,1)	(1,1,1)	a_6	(m_4,t_2,m_6)	false

Table A3.7 The nodes (a) and arcs (b) of Figure 3.17

V′			A′		
$v_i^{'}$	$m_i^{'}$	$\vec{\sigma}_i^{*'}$	$a_k^{'}$	($m_i^{'}$,t_n,$m_j^{'}$)	$atf_k^{'}$
$v_0^{'}$	(2,0,0,0)	(0,0,0,**0**)	$a_0^{'}$	($m_0^{'}$,t_1,$m_1^{'}$)	false
$v_1^{'}$	(1,0,1,0)	(0,1,0,**0**)	$a_1^{'}$	($m_0^{'}$,t_0,$m_2^{'}$)	false
$v_2^{'}$	(1,1,0,0)	(1,0,0,**0**)	$a_2^{'}$	($m_1^{'}$,t_1,$m_3^{'}$)	false
$v_3^{'}$	(0,0,2,0)	(0,2,0,**0**)	$a_3^{'}$	($m_1^{'}$,t_0,$m_4^{'}$)	false
$v_4^{'}$	(0,1,1,0)	(1,1,0,**0**)	$a_4^{'}$	($m_2^{'}$,t_1,$m_4^{'}$)	false
$v_5^{'}$	(0,2,0,0)	(2,0,0,**0**)	$a_5^{'}$	($m_2^{'}$,t_0,$m_5^{'}$)	false
$v_6^{'}$	(0,0,0,1)	(1,1,1,**0**)	$a_6^{'}$	($m_4^{'}$,t_2,$m_6^{'}$)	false
$\boldsymbol{v_7^{'}}$	(1,0,0,1)	(0,0,0,1)	$\boldsymbol{a_7^{'}}$	($m_0^{'}$,t_3,$m_7^{'}$)	false
$\boldsymbol{v_8^{'}}$	(0,0,1,1)	(0,1,0,1)	$\boldsymbol{a_8^{'}}$	($m_1^{'}$,t_3,$m_8^{'}$)	false
$\boldsymbol{v_9^{'}}$	(0,1,0,1)	(1,0,0,1)	$\boldsymbol{a_9^{'}}$	($m_2^{'}$,t_3,$m_9^{'}$)	false
$\boldsymbol{v_{10}^{'}}$	(0,0,0,2)	(0,0,0,2)	$\boldsymbol{a_{10}^{'}}$	($m_7^{'}$,t_3,$m_{10}^{'}$)	false
			$\boldsymbol{a_{11}^{'}}$	($m_7^{'}$,t_1,$m_8^{'}$)	false
			$\boldsymbol{a_{12}^{'}}$	($m_7^{'}$,t_0,$m_9^{'}$)	false

Table A3.8 The nodes (a) and arcs (b) of Figure 3.19

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\underline{\vec{\sigma}}_i^{*}$	$\underline{a}_k$	($\underline{m}_i$,t_n,$\underline{m}_j$)	$\underline{atf}_k$
$\underline{v}_0$	(2,0,0,0)	(0,0,0,0)	$\underline{a}_0$	($\underline{m}_0$,t_1,$\underline{m}_1$)	false
$\underline{v}_1$	(1,0,1,0)	(0,1,0,0)	$\underline{a}_1$	($\underline{m}_0$,t_0,$\underline{m}_2$)	false
$\underline{v}_2$	(1,1,0,0)	(1,0,0,0)	$\underline{a}_2$	($\underline{m}_1$,t_1,$\underline{m}_3$)	false
$\underline{v}_3$	(0,0,2,0)	(0,2,0,0)	$\underline{a}_3$	($\underline{m}_1$,t_0,$\underline{m}_4$)	false
$\underline{v}_4$	(0,1,1,0)	(1,1,0,0)	$\underline{a}_4$	($\underline{m}_2$,t_1,$\underline{m}_4$)	false
$\underline{v}_5$	(0,2,0,0)	(2,0,0,0)	$\underline{a}_5$	($\underline{m}_2$,t_0,$\underline{m}_5$)	false
$\underline{v}_6$	(0,0,0,1)	(1,1,1,0)	$\underline{a}_6$	($\underline{m}_4$,t_2,$\underline{m}_6$)	false

Table A4.4 The nodes and arcs of Figure 4.4

delTtoParc

V			A		
v_i	m_i	$\vec{\sigma}_i^{*}$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,0,0,0,0,0,0,0,0)	(0,0,0,0,0,0,0,0,0,0)	a_0	(m_0,t_2,m_1)	false
v_1	(0,0,0,1,**1**,0,0,0,0)	(0,0,**1**,0,0,0,0,0,0,0)	a_1	(m_0,t_1,m_2)	false
v_2	(0,0,1,0,0,0,0,0,0)	(0,1,0,0,0,0,0,0,0,0)	a_2	(m_0,t_0,m_3)	false
v_3	(0,1,0,0,0,0,0,0,0)	(1,0,0,0,0,0,0,0,0,0)	a_3	(m_1,t_6,m_4)	false
v_4	(0,0,0,0,**1**,0,1,0,0)	(0,0,**1**,0,0,0,1,0,0,0)	a_4	(m_1,t_5,m_5)	true

v_5	(0,0,0,0,1,1,0,0,0)	(1,0,0,1,0,0,0,0,0,0)	a_5	(m_3,t_3,m_5)	false
v_6	(0,0,0,0,1,0,0,0,1)	(1,0,0,1,0,0,0,0,1,0)	a_6	(m_4,t_9,m_6)	true
v_7	(0,0,0,0,0,1,0,1,0)	(1,0,0,1,0,0,0,1,0,0)	a_7	(m_5,t_8,m_6)	false
v_8	(0,0,0,0,0,0,0,1,1)	(1,0,0,1,0,0,0,1,1,0)	a_8	(m_5,t_7,m_7)	false
v_9	(0,0,0,1,**0**,0,0,1,0)	(0,0,**1**,0,0,0,0,1,0,0)	a_9	(m_6,t_7,m_8)	false
v_{10}	(0,0,0,0,0,1,0,0,0)	(0,1,0,0,1,0,0,0,0,0)	a_{10}	(m_7,t_8,m_8)	false
v_{11}	(0,0,0,0,**0**,0,1,1,0)	(0,0,**1**,0,0,0,1,1,0,0)	a_{11}	(m_1,t_7,m_9)	false
v_{12}	(0,0,0,0,0,0,0,0,1)	(0,1,0,0,1,0,0,0,1,0)	a_{12}	(m_2,t_4,m_{10})	false
			a_{13}	(m_4,t_7,m_{11})	true
			a_{14}	(m_9,t_5,m_7)	true
			a_{15}	(m_9,t_6,m_{11})	false
			a_{16}	(m_{10},t_8,m_{12})	false
			a_{17}	(m_{11},t_9,m_8)	true

Table A3.9 The nodes (a) and arcs (b) of Figure 3.21

V′			A′		
v_i'	m_i'	$\vec{\sigma}_i^{*\prime}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(1,0,0,0,0,0,0,0,0)	(0,0,0,0,0,0,0,0,0,0)	a_0'	(m_0',t_2,m_1')	false
v_1'	(0,0,0,1,**0**,0,0,0,0)	(0,0,**1**,0,0,0,0,0,0,0)	a_1'	(m_0',t_1,m_2')	false
v_2'	(0,0,1,0,0,0,0,0,0)	(0,1,0,0,0,0,0,0,0,0)	a_2'	(m_0',t_0,m_3')	false
v_3'	(0,1,0,0,0,0,0,0,0)	(1,0,0,0,0,0,0,0,0,0)	a_3'	(m_1',t_6,m_4')	false
v_4'	(0,0,0,0,**0**,0,1,0,0)	(0,0,**1**,0,0,0,1,0,0,0)	a_5'	(m_3',t_3,m_5')	true
v_5'	(0,0,0,0,1,1,0,0,0)	(1,0,0,1,0,0,0,0,0,0)	a_7'	(m_5',t_8,m_6')	false
v_6'	(0,0,0,0,1,0,0,0,1)	(1,0,0,1,0,0,0,0,1,0)	a_8'	(m_5',t_7,m_7')	true
v_7'	(0,0,0,0,0,1,0,1,0)	(1,0,0,1,0,0,0,1,0,0)	$\boldsymbol{a_9}$	(m_6',t_7,m_8')	false
v_8'	(0,0,0,0,0,0,0,1,1)	(1,0,0,1,0,0,0,1,1,0)	a_{10}'	(m_7',t_8,m_8')	false
v_{10}'	(0,0,0,0,0,1,0,0,0)	(0,1,0,0,1,0,0,0,0,0)	a_{12}'	(m_2',t_4,m_{10}')	false
v_{12}'	(0,0,0,0,0,0,0,0,1)	(0,1,0,0,1,0,0,0,1,0)	a_{16}'	(m_{10}',t_8,m_{12}')	false
			a_{18}'	(m_1',t_5,m_{10}')	false
			a_{19}'	(m_4',t_9,m_{12}')	true

Table A3.10 The nodes (a) and arcs (b) of Figure 3.23

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\underline{\vec{\sigma}_i^{*}}$	$\underline{a}_k$	$(\underline{m}_i,t_n,\underline{m}_j)$	$\underline{atf}_k$
$\underline{v}_0$	(1,0,0,0,0,0,0,0,0)	(0,0,0,0,0,0,0,0,0,0)	$\underline{a}_0$	$(\underline{m}_0,t_2,\underline{m}_1)$	false
$\underline{v}_1$	(0,0,0,1,**0**,0,0,0,0)	(0,0,**1**,0,0,0,0,0,0,0)	$\underline{a}_1$	$(\underline{m}_0,t_1,\underline{m}_2)$	false
$\underline{v}_2$	(0,0,1,0,0,0,0,0,0)	(0,1,0,0,0,0,0,0,0,0)	$\underline{a}_2$	$(\underline{m}_0,t_0,\underline{m}_3)$	false

$\underline{v}_{3}$	(0,1,0,0,0,0,0,0,0)	(1,0,0,0,0,0,0,0,0,0)	$\underline{a}_{3}$	($\underline{m}_{1}$,t_{6},$\underline{m}_{4}$)	false
$\underline{v}_{4}$	(0,0,0,0,**0**,0,1,0,0)	(0,0,**1**,0,0,0,1,0,0,0)	$\underline{a}_{5}$	($\underline{m}_{3}$,t_{3},$\underline{m}_{5}$)	false
$\underline{v}_{5}$	(0,0,0,0,1,1,0,0,0)	(1,0,0,1,0,0,0,0,0,0)	$\underline{a}_{7}$	($\underline{m}_{5}$,t_{8},$\underline{m}_{6}$)	false
$\underline{v}_{6}$	(0,0,0,0,1,0,0,0,1)	(1,0,0,1,0,0,0,0,1,0)	$\underline{a}_{8}$	($\underline{m}_{5}$,t_{7},$\underline{m}_{7}$)	false
$\underline{v}_{7}$	(0,0,0,0,0,1,0,1,0)	(1,0,0,1,0,0,0,1,0,0)	$\underline{a}_{9}$	($\underline{m}_{6}$,t_{7},$\underline{m}_{8}$)	false
$\underline{v}_{8}$	(0,0,0,0,0,0,0,1,1)	(1,0,0,1,0,0,0,1,1,0)	$\underline{a}_{10}$	($\underline{m}_{7}$,t_{8},$\underline{m}_{8}$)	false
$\underline{v}_{9}$	(0,0,0,1,**-1**,0,0,1,0)	(0,0,**1**,0,0,0,0,1,0,0)	$\underline{a}_{11}$	($\underline{m}_{1}$,t_{7},$\underline{m}_{9}$)	false
$\underline{v}_{10}$	(0,0,0,0,0,1,0,0,0)	(0,1,0,0,1,0,0,0,0,0)	$\underline{a}_{12}$	($\underline{m}_{2}$,t_{4},$\underline{m}_{10}$)	false
$\underline{v}_{11}$	(0,0,0,0,**-1**,0,1,1,0)	(0,0,**1**,0,0,0,1,1,0,0)	$\underline{a}_{13}$	($\underline{m}_{4}$,t_{7},$\underline{m}_{11}$)	true
$\underline{v}_{12}$	(0,0,0,0,0,0,0,0,1)	(0,1,0,0,1,0,0,0,1,0)	$\underline{a}_{14}$	($\underline{m}_{9}$,t_{5},$\underline{m}_{7}$)	true
			$\underline{a}_{15}$	($\underline{m}_{9}$,t_{6},$\underline{m}_{11}$)	false
			$\underline{a}_{16}$	($\underline{m}_{10}$,t_{8},$\underline{m}_{12}$)	false
			$\underline{a}_{17}$	($\underline{m}_{11}$,t_{9},$\underline{m}_{8}$)	true

Table A5.1 The nodes and arcs of Figure 5.1

delPtoTarc

V			A		
v_i	m_i	$\vec{\sigma}_i^*$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,1,0,0)	(0,0,0,0)	a_0	(m_0,t_2,m_1)	false
v_1	(1,0,0,1)	(0,0,1,0)	a_1	(m_0,t_1,m_2)	false
v_2	(0,1,1,0)	(0,1,0,0)	a_2	(m_0,t_0,m_3)	false
v_3	(0,2,0,0)	(1,0,0,0)	a_3	(m_1,t_1,m_4)	false
v_4	(0,0,1,1)	(0,1,1,0)	a_4	(m_1,t_0,m_5)	false
v_5	(0,1,0,1)	(1,0,1,0)	a_5	(m_2,t_2,m_4)	false
v_6	(0,**0**,0,1)	(0,1,0,**1**)	a_6	(m_3,t_2,m_5)	false
v_7	(0,0,0,2)	(1,0,2,0)	a_7	(m_2,t_3,m_6)	false
			a_8	(m_5,t_2,m_7)	false

Table A3.11 The nodes (a) and arcs (b) of Figure 3.25

V'			A'		
v_i'	m_i'	$\vec{\sigma}_i^{*\prime}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(1,1,0,0)	(0,0,0,0)	a_0'	(m_0',t_2,m_1')	false
v_1'	(1,0,0,1)	(0,0,1,0)	a_1'	(m_0',t_1,m_2')	false
v_2'	(0,1,1,0)	(0,1,0,0)	a_2'	(m_0',t_0,m_3')	false
v_3'	(0,2,0,0)	(1,0,0,0)	a_3'	(m_1',t_1,m_4')	false
v_4'	(0,0,1,1)	(0,1,1,0)	a_4'	(m_1',t_0,m_5')	false
v_5'	(0,1,0,1)	(1,0,1,0)	a_5'	(m_2',t_2,m_4')	false

v_7'	(0,**0**,0,2)	(1,0,2,0)	a_6'	(m_3',t_2,m_5')	false
			a_7'	(m_2',t_3,m_5')	true
			a_8'	(m_5',t_2,m_7')	false
			a_9'	(m_4',t_3,m_7')	true

Table A3.12 The nodes (a) and arcs (b) of Figure 3.27

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\vec{\sigma}_i^*$	$\underline{a}_k$	$(\underline{m}_i,\underline{t}_n,\underline{m}_j)$	$\underline{atf}_k$
$\underline{v}_0$	(1,1,0,0)	(0,0,0,0)	$\underline{a}_0$	$(\underline{m}_0,t_2,\underline{m}_1)$	false
$\underline{v}_1$	(1,0,0,1)	(0,0,1,0)	$\underline{a}_1$	$(\underline{m}_0,t_1,\underline{m}_2)$	false
$\underline{v}_2$	(0,1,1,0)	(0,1,0,0)	$\underline{a}_2$	$(\underline{m}_0,t_0,\underline{m}_3)$	false
$\underline{v}_3$	(0,2,0,0)	(1,0,0,0)	$\underline{a}_3$	$(\underline{m}_1,t_1,\underline{m}_4)$	false
$\underline{v}_4$	(0,0,1,1)	(0,1,1,0)	$\underline{a}_4$	$(\underline{m}_1,t_0,\underline{m}_5)$	false
$\underline{v}_5$	(0,1,0,1)	(1,0,1,0)	$\underline{a}_5$	$(\underline{m}_2,t_2,\underline{m}_4)$	false
$\underline{v}_6$	(0,**1**,0,1)	(0,1,0,**1**)	$\underline{a}_6$	$(\underline{m}_3,t_2,\underline{m}_5)$	false
$\underline{v}_7$	(0,0,0,2)	(1,0,2,0)	$\underline{a}_7$	$(\underline{m}_2,t_3,\underline{\mathbf{m}}_5)$	true
			$\underline{a}_8$	$(\underline{m}_5,t_2,\underline{m}_7)$	false

Table A5.2 The nodes (a) and arcs (b) of Figure 5.2

delToken

V			A		
v_i	m_i	$\vec{\sigma}_i^*$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,0,1,0)	(0,0,0)	a_0	(m_0,t_0,m_1)	false
v_1	(0,1,1,0)	(1,0,0)	a_1	(m_1,t_1,m_2)	false
v_2	(0,0,2,0)	(1,1,0)	a_2	(m_2,t_2,m_3)	false
v_3	(0,0,1,1)	(1,1,1)	a_3	(m_0,t_2,m_4)	false
v_4	(1,0,0,1)	(0,0,1)	a_4	(m_1,t_2,m_5)	false
v_5	(0,1,0,1)	(1,0,1)	a_5	(m_3,t_2,m_6)	false
v_6	(0,0,0,2)	(1,1,2)	a_6	(m_4,t_0,m_5)	false
			a_7	(m_5,t_1,m_3)	false

Table A3.13 The nodes (a) and arcs (b) of Figure 3.29

V’			A’		
v_i'	m_i'	$\vec{\sigma}_i^{*'}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(1,0,0,0)	(0,0,0)	a_0'	(m_0',t_0,m_1')	false

v_1'	(0,1,0,0)	(1,0,0)
v_2'	(0,0,1,0)	(1,1,0)
v_3'	(0,0,0,1)	(1,1,1)

a_1'	(m_1',t_1,m_2')	false
a_2'	(m_2',t_2,m_3')	false

Table A3.14 The nodes (a) and arcs (b) of Figure 3.31

$\underline{V}$			$\underline{A}$		
$\underline{v_i}$	$\underline{m_i}$	$\underline{\vec{\sigma}_i^*}$	$\underline{a_k}$	$(\underline{m_i},t_n,\underline{m_j})$	$\underline{atf_k}$
$\underline{v_0}$	(1,0,**0**,0)	(0,0,0)	$\underline{a_0}$	$(\underline{m_0},t_0,\underline{m_1})$	false
$\underline{v_1}$	(0,1,**0**,0)	(1,0,0)	$\underline{a_1}$	$(\underline{m_1},t_1,\underline{m_2})$	false
$\underline{v_2}$	(0,0,**1**,0)	(1,1,0)	$\underline{a_2}$	$(\underline{m_2},t_2,\underline{m_3})$	false
$\underline{v_3}$	(0,0,**0**,1)	(1,1,1)	$\underline{a_3}$	$(\underline{m_0},t_2,\underline{m_4})$	false
$\underline{v_4}$	(1,0,**-1**,1)	(0,0,1)	$\underline{a_4}$	$(\underline{m_1},t_2,\underline{m_5})$	false
$\underline{v_5}$	(0,1,**-1**,1)	(1,0,1)	$\underline{a_5}$	$(\underline{m_3},t_2,\underline{m_6})$	false
$\underline{v_6}$	(0,0,**-1**,2)	(1,1,2)	$\underline{a_6}$	$(\underline{m_4},t_0,\underline{m_5})$	false
			$\underline{a_7}$	$(\underline{m_5},t_1,\underline{m_3})$	false

Table A5.3 The nodes and arcs of Figure 5.3

delTransition

V			*A*		
v_i	m_i	$\vec{\sigma}_i^*$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(2,0,0,0)	(0,0,0,**0**)	a_0	(m_0,t_1,m_1)	false
v_1	(1,0,1,0)	(0,1,0,**0**)	a_1	(m_0,t_0,m_2)	false
v_2	(1,1,0,0)	(1,0,0,**0**)	a_2	(m_1,t_1,m_3)	false
v_3	(0,0,2,0)	(0,2,0,**0**)	a_3	(m_1,t_0,m_4)	false
v_4	(0,1,1,0)	(1,1,0,**0**)	a_4	(m_2,t_1,m_4)	false
v_5	(0,2,0,0)	(2,0,0,**0**)	a_5	(m_2,t_0,m_5)	false
v_6	(0,0,0,1)	(1,1,1,**0**)	a_6	(m_4,t_2,m_6)	false
v_7	(1,0,0,1)	(0,0,0,1)	a_7	(m_0,t_3,m_7)	false
v_8	(0,0,1,1)	(0,1,0,1)	a_8	(m_1,t_3,m_8)	false
v_9	(0,1,0,1)	(1,0,0,1)	a_9	(m_2,t_3,m_9)	false
v_{10}	(0,0,0,2)	(0,0,0,2)	a_{10}	(m_7,t_3,m_{10})	false
			a_{11}	(m_7,t_1,m_8)	false
			a_{12}	(m_7,t_0,m_9)	false

Table A3.15 The nodes (a) and arcs (b) of Figure 3.33

V'			A'		
v_i'	m_i'	$\vec{\sigma}_i^{*\prime}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(2,0,0,0)	(0,0,0)	a_0'	(m_0',t_1,m_1')	false
v_1'	(1,0,1,0)	(0,1,0)	a_1'	(m_0',t_0,m_2')	false
v_2'	(1,1,0,0)	(1,0,0)	a_2'	(m_1',t_1,m_3')	false
v_3'	(0,0,2,0)	(0,2,0)	a_3'	(m_1',t_0,m_4')	false
v_4'	(0,1,1,0)	(1,1,0)	a_4'	(m_2',t_1,m_4')	false
v_5'	(0,2,0,0)	(2,0,0)	a_5'	(m_2',t_0,m_5')	false
v_6'	(0,0,0,1)	(1,1,1)	a_6'	(m_4',t_2,m_6')	false

Table A3.16 The nodes (a) and arcs (b) of Figure 3.35

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\underline{\vec{\sigma}_i^*}$	$\underline{a}_k$	$(\underline{m}_i,t_n,\underline{m}_j)$	$\underline{atf}_k$
$\underline{v}_0$	(2,0,0,0)	(0,0,0)	$\underline{a}_0$	$(\underline{m}_0,t_1,\underline{m}_1)$	false
$\underline{v}_1$	(1,0,1,0)	(0,1,0)	$\underline{a}_1$	$(\underline{m}_0,t_0,\underline{m}_2)$	false
$\underline{v}_2$	(1,1,0,0)	(1,0,0)	$\underline{a}_2$	$(\underline{m}_1,t_1,\underline{m}_3)$	false
$\underline{v}_3$	(0,0,2,0)	(0,2,0)	$\underline{a}_3$	$(\underline{m}_1,t_0,\underline{m}_4)$	false
$\underline{v}_4$	(0,1,1,0)	(1,1,0)	$\underline{a}_4$	$(\underline{m}_2,t_1,\underline{m}_4)$	false
$\underline{v}_5$	(0,2,0,0)	(2,0,0)	$\underline{a}_5$	$(\underline{m}_2,t_0,\underline{m}_5)$	false
$\underline{v}_6$	(0,0,0,1)	(1,1,1)	$\underline{a}_6$	$(\underline{m}_4,t_2,\underline{m}_6)$	False
$\underline{v}_7$	(1,0,0,1)	(0,0,0,**1**)	$\underline{a}_7$	$(\underline{m}_0,t_3,\underline{m}_7)$	false
$\underline{v}_8$	(0,0,1,1)	(0,1,0,**1**)	$\underline{a}_8$	$(\underline{m}_1,t_3,\underline{m}_8)$	false
$\underline{v}_9$	(0,1,0,1)	(1,0,0,**1**)	$\underline{a}_9$	$(\underline{m}_2,t_3,\underline{m}_9)$	false
$\underline{v}_{10}$	(0,0,0,2)	(0,0,0,**2**)	$\underline{a}_{10}$	$(\underline{m}_7,t_3,\underline{m}_{10})$	false
			$\underline{a}_{11}$	$(\underline{m}_7,t_1,\underline{m}_8)$	false
			$\underline{a}_{12}$	$(\underline{m}_7,t_0,\underline{m}_9)$	false

Table A5.4 The nodes and arcs of Figure 5.4

delPlace

V			A		
v_i	m_i	$\vec{\sigma}_i^*$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,0,0,**0**,0,0)	(0,0,0)	a_0	(m_0,t_2,m_1)	false
v_1	(0,1,0,**1**,1,0)	(1,0,0)	a_1	(m_1,t_0,m_3)	false
v_2	(0,1,0,**0**,0,1)	(1,0,1)	a_2	(m_1,t_3,m_2)	false
v_3	(0,0,1,**2**,1,0)	(1,1,0)	a_3	(m_2,t_0,m_4)	false
v_4	(0,0,1,**1**,0,1)	(1,1,1)	a_4	(m_3,t_2,m_4)	false

Table A3.17 The nodes (a) and arcs (b) of Figure 3.37

V'			*A'*		
v_i'	m_i'	$\vec{\sigma}_i^{*'}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(1,0,0,0,0)	(0,0,0)	a_0'	(m_0',t_2,m_1')	false
v_1'	(0,1,0,1,0)	(1,0,0)	a_1'	(m_1',t_0,m_3')	false
v_2'	(0,1,0,0,1)	(1,0,1)	a_2'	(m_1',t_3,m_2')	false
v_3'	(0,0,1,1,0)	(1,1,0)	a_3'	(m_2',t_0,m_4')	false
v_4'	(0,0,1,0,1)	(1,1,1)	a_4'	(m_3',t_2,m_4')	false

Table A3.18 The nodes (a) and arcs (b) of Figure 3.39

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\vec{\underline{\sigma}}_i^{*}$	$\underline{a}_k$	$(\underline{m}_i,t_n,\underline{m}_j)$	$\underline{atf}_k$
$\underline{v}_0$	(1,0,0,0,0)	(0,0,0)	$\underline{a}_0$	$(\underline{m}_0,t_2,\underline{m}_1)$	false
$\underline{v}_1$	(0,1,0,1,0)	(1,0,0)	$\underline{a}_1$	$(\underline{m}_1,t_0,\underline{m}_3)$	false
$\underline{v}_2$	(0,1,0,0,1)	(1,0,1)	$\underline{a}_2$	$(\underline{m}_1,t_3,\underline{m}_2)$	false
$\underline{v}_3$	(0,0,1,1,0)	(1,1,0)	$\underline{a}_3$	$(\underline{m}_2,t_0,\underline{m}_4)$	false
$\underline{v}_4$	(0,0,1,0,1)	(1,1,1)	$\underline{a}_4$	$(\underline{m}_3,t_2,\underline{m}_4)$	false

Table A5.5 The nodes (a) and arcs (b) of Figure 5.5

mergePlace

V			*A*		
v_i	m_i	$\vec{\sigma}_i^{*}$	a_k	(m_i,t_n,m_j)	atf_k
v_0	(1,**0**,0,1,**0**,0)	(0,0,0,0)	a_0	(m_0,t_2,m_1)	false
v_1	(1,**0**,0,0,**1**,0)	(0,0,1,0)	a_1	(m_0,t_0,m_2)	false
v_2	(0,**1**,0,1,**0**,0)	(1,0,0,0)	a_2	(m_1,t_3,m_3)	false
v_3	(1,**0**,0,0,**0**,1)	(0,0,1,1)	a_3	(m_1,t_0,m_4)	false
v_4	(0,**1**,0,0,**1**,0)	(1,0,1,0)	a_4	(m_2,t_2,m_4)	false
v_5	(0,**0**,1,1,**0**,0)	(1,1,0,0)	a_5	(m_2,t_1,m_5)	false
v_6	(0,**1**,0,0,**0**,1)	(1,0,1,1)	a_6	(m_3,t_0,m_6)	false
v_7	(0,**0**,1,0,**1**,0)	(1,1,1,0)	a_7	(m_4,t_3,m_6)	false
v_8	(0,**0**,1,0,**0**,1)	(1,1,1,1)	a_8	(m_4,t_1,m_7)	false
			a_9	(m_5,t_2,m_7)	false
			a_{10}	(m_6,t_1,m_8)	false
			a_{11}	(m_7,t_3,m_8)	false

Table A3.19 The nodes (a) and arcs (b) of Figure 3.41

V′			*A′*		
v_i'	m_i'	$\vec{\sigma}_i^{*'}$	a_k'	(m_i',t_n,m_j')	atf_k'
v_0'	(1,**0**,0,1,0)	(0,0,0,0)	a_0'	(m_0',t_2,m_1')	false
v_1'	(1,**1**,0,0,0)	(0,0,1,0)	a_1'	(m_0',t_0,m_2')	false
v_2'	(0,**1**,0,1,0)	(1,0,0,0)	a_2'	(m_1',t_3,m_3')	false
v_3'	(1,**0**,0,0,1)	(0,0,1,1)	a_3'	(m_1',t_0,m_4')	false
v_4'	(0,**2**,0,0,0)	(1,0,1,0)	a_4'	(m_2',t_2,m_4')	false
v_5'	(0,**0**,1,1,0)	(1,1,0,0)	a_5'	(m_2',t_1,m_5')	false
v_6'	(0,**1**,0,0,1)	(1,0,1,1)	a_6'	(m_3',t_0,m_6')	false
v_7'	(0,**1**,1,0,0)	(1,1,1,0)	a_7'	(m_4',t_3,m_6')	false
v_8'	(0,**0**,1,0,1)	(1,1,1,1)	a_8'	(m_4',t_1,m_7')	false
$\boldsymbol{v_9'}$	(1,0,1,0,0)	(0,1,1,0)	a_9'	(m_5',t_2,m_7')	false
$\boldsymbol{v_{10}'}$	(0,0,0,1,1)	(1,0,0,1)	a_{10}'	(m_6',t_1,m_8')	false
$\boldsymbol{v_{11}'}$	(0,0,0,0,2)	(1,0,1,2)	a_{11}'	(m_7',t_3,m_8')	false
$\boldsymbol{v_{12}'}$	(0,0,2,0,0)	(1,2,1,0)	$\boldsymbol{a_{12}'}$	(m_1',t_1,m_9')	false
			$\boldsymbol{a_{13}'}$	(m_2',t_3,m_{10}')	false
			$\boldsymbol{a_{14}'}$	(m_9',t_0,m_7')	false
			$\boldsymbol{a_{15}'}$	(m_{10}',t_2,m_6')	false
			$\boldsymbol{a_{16}'}$	(m_6',t_3,m_{11}')	false
			$\boldsymbol{a_{17}'}$	(m_7',t_1,m_{12}')	false

Table A3.20 The nodes (a) and arcs (b) of Figure 3.43

$\underline{V}$			$\underline{A}$		
$\underline{v}_i$	$\underline{m}_i$	$\underline{\vec{\sigma}}_i^*$	$\underline{a}_k$	$(\underline{m}_i,t_n,\underline{m}_j)$	$\underline{atf}_k$
$\underline{v}_0$	(1,**0**,0,1,0)	(0,0,0,0)	$\underline{a}_0$	$(\underline{m}_0,t_2,\underline{m}_1)$	false
$\underline{v}_1$	(1,**1**,0,0,0)	(0,0,1,0)	$\underline{a}_1$	$(\underline{m}_0,t_0,\underline{m}_2)$	false
$\underline{v}_2$	(0,**1**,0,1,0)	(1,0,0,0)	$\underline{a}_2$	$(\underline{m}_1,t_3,\underline{m}_3)$	false
$\underline{v}_3$	(1,**0**,0,0,1)	(0,0,1,1)	$\underline{a}_3$	$(\underline{m}_1,t_0,\underline{m}_4)$	false
$\underline{v}_4$	(0,**2**,0,0,0)	(1,0,1,0)	$\underline{a}_4$	$(\underline{m}_2,t_2,\underline{m}_4)$	false
$\underline{v}_5$	(0,**0**,1,1,0)	(1,1,0,0)	$\underline{a}_5$	$(\underline{m}_2,t_1,\underline{m}_5)$	false
$\underline{v}_6$	(0,**1**,0,0,1)	(1,0,1,1)	$\underline{a}_6$	$(\underline{m}_3,t_0,\underline{m}_6)$	false
$\underline{v}_7$	(0,**1**,1,0,0)	(1,1,1,0)	$\underline{a}_7$	$(\underline{m}_4,t_3,\underline{m}_6)$	false
$\underline{v}_8$	(0,**0**,1,0,1)	(1,1,1,1)	$\underline{a}_8$	$(\underline{m}_4,t_1,\underline{m}_7)$	false
			$\underline{a}_9$	$(\underline{m}_5,t_2,\underline{m}_7)$	false
			$\underline{a}_{10}$	$(\underline{m}_6,t_1,\underline{m}_8)$	false
			$\underline{a}_{11}$	$(\underline{m}_7,t_3,\underline{m}_8)$	false

Table A5.6 The nodes and arcs of Figure 5.6