DISCRETE EVENT SYSTEM TOOLS FOR FAULT DIAGNOSIS AND COLLISION PREVENTION

XU WANG

Ph.D
Informítica e Ingeniería de Sistemas
Escuela de Ingenierá y Arquitectura
Universidad de Zaragoza

November 2016

Xu Wang: *Discrete event system tools for fault diagnosis and collision prevention,* © November 2016

SUPERVISORS:
Mahulea, Cristian Florentín
Silva Suárez, Manuel

LOCATION:
Zaragoza, Spain

RESUMEN

Los Sistemas de Eventos Discretos (DES por sus siglas en inglés), son herramientas utilizadas ampliamente en sistemas prácticos, incluyendo sistemas de manufactura, movimientos de robots, sistemas de logísticas, etc. Se proponen las redes de Petri para modelar y analizar DES de una manera compacta y eficiente. Esta tesis se centra en dos cuestiones importantes para los sistemas de eventos discretos.

El primero de ellos es el diagnóstico de fallos. Se propone un enfoque en línea para sistemas temporizados, modelado con redes de Petri con tiempo (TPN, por sus siglas en inglés). El conjunto de transiciones se parte en dos subconjuntos que contienen transiciones observables e inobservables, respectivamente. Los fallos corresponden a un subconjunto de las transiciones inobservables. De acuerdo con la mayor parte de la literatura sobre sistemas de eventos discretos, se definen tres estados en el diagnóstico: normal, defectuoso e incierto, respectivamente. El enfoque propuesto usa grafos de diagnóstico de fallos, que se calculan de manera incremental mediante grafos de clase de estado de la parte no observable de la TPN. Después de cada observación, si la parte del grafo de diagnóstico de fallos necesaria para calcular los estados de diagnóstico no está disponible, el grafo de clase de estado de la TPN no observable se calcula empezando desde los estados coherentes disponibles. Después, este grafo se optimiza y se añade al grafo de diagnóstico de fallos parcial, conservando solamente la información necesaria para el cálculo del diagnóstico de los estados. Se proporcionan algoritmos para calcular el grafo de diagnóstico de fallos y los estados de diagnóstico. El método ha sido implementado como un paquete de software, incluyéndose resultados de simulación.

El segundo problema concierne la evitación de colisiones en la planificación de movimientos de robots, abordándose dos sub-problemas: la prevención de bloqueos mutuos con control en tiempo real, y sin él. En el caso de la prevención de bloqueos mutuos en sistemas temporizados, se considera el diseño de una política de control de prevención de bloqueos mutuos para un equipo de robots móviles que ha de seguir unas trayectorias para completar una tarea determinada. Se acepta que algunas regiones tienen capacidad limitada (el número de robots que puede haber simultáneamente en esas regiones es limitado), lo que puede verse como una limitación de los recursos disponibles en un sistema de adjudicación de recursos (RAS por sus siglas en inglés). Se propone un método basado en arcos inhibidores que puede aplicarse de forma descentralizada. Se trata de una alternativa a la estrategia de prevención de bloqueos mutuos basada en monitorear ubicaciones que podrían usarse en varias aplicaciones, ya que el coste de implementación puede ser menor. En el segundo caso, se trata el problema de la prevención de colisiones en un sistema con múltiples robots, en el que no se aplica control en tiempo real. Cada robot tiene un conjunto de trayectorias posibles y cada una de ellas cumple una tarea individual. Las trayectorias consisten en secuencias de regiones que han de seguirse y se conoce el tiempo para desplazarse por cada región. El problema consiste en evitar colisiones entre los robots al imponer retrasos en el tiempo inicial para cada trayectoria del conjunto de caminos disponibles para cada robot. La solución se aborda como optimización mediante programación lineal mixta (entera y real) que devuelve los retrasos de tiempo iniciales y, cuando es necesario, la trayectoria elegida para cada robot. Finalmente, se realiza un estudio estadístico de las soluciones propuestas y se concluye que una formulación es preferible a las otras.

ABSTRACT

Discrete Event Systems (DES) are widely used tools in practical systems including manufacturing systems, robot motion, logistics systems and so on. *Petri Nets* (PN) are proposed to model and analyze DES in a compact and efficient way. In this thesis, we focus on two topics in DES and deal with them using PN.

One problem is *fault diagnosis*. We propose an on-line approach for fault diagnosis of timed discrete event systems modeled by Time Petri Net (TPN). The set of transitions is partitioned into two subsets containing *observable* and *unobservable* transitions, respectively. Faults correspond to a subset of unobservable transitions. In accordance with most of the literature on discrete event systems, we define three diagnosis states, namely *normal*, *faulty* and *uncertain* states, respectively. The proposed approach uses *fault diagnosis graph*, which is incrementally computed using the state class graph of the unobservable TPN. After each observation, if the part of FDG necessary to compute the diagnosis states is not available, the state class graph of the unobservable TPN is computed starting from the consistent states. This graph is then optimized and added to the partial FDG keeping only the necessary information for computation of the diagnosis states. We provide algorithms to compute the FDG and the diagnosis states. The method is implemented as a software package and simulation results are included.

The other problem is *collision avoidance* in robot planning and we deal with two problems: deadlock prevention with and without real time control. In the case of deadlock prevention with real time control, we consider the problem of design a *deadlock prevention control policy* for a team of mobile robots that should follow some trajectories in order to accomplish a given task. We assume that some regions have limited capacity (i.e., the numbers of robots that can be simultaneously in that regions are limited) that can be seen as limited available resources in a *resource allocation system* (RAS). We propose a method, based on inhibitor arcs that can be applied in a decentralized way. This is an alternative to the deadlock prevention strategy based on monitor places that could be used in several applications since the implementation cost could be smaller. In the second case, we address a collision avoidance problem in a multi-robot system, where real time control is not applicable. Each robot has a set of possible trajectories, each trajectory fulfilling its individual task. The trajectories consist of sequences of regions to be followed, and the time for moving inside each region is known. The problem is to avoid inter-robot collisions by imposing initial time delays for each trajectory. Two solutions are developed, depending on the possibility of imposing a certain trajectory from the available set of paths for each robot. The solutions have the form of mixed integer linear programming optimizations that return the initial time delays and, when necessary, the chosen trajectory for each robot. Finally, we perform a statistical study on the proposed solutions and we conclude that one formulation is preferable to the others.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

Part I

INTRODUCTION AND PETRI NETS

1 BRIEF INTRODUCTION

In this thesis, we discuss two critical problems in *Discrete Event Systems* (DES): fault diagnosis in Part II and *collision avoidance* in multiple robots planning in Part III. They are important in many real world systems, e.g., manufacturing systems and multiple robots systems. In this chapter, we give a brief introduction of the two parts. More detailed introductions and reviews of related works are in each part.

Let us first discuss the fault diagnosis problem given in Part II. We consider fault behaviors as faults, where fault behaviors in DES equals to fault events. In manufacturing systems, a fault behavior is a potential risk. It may let the system produce mis-assembled products or in the worst case, crash the whole system. Therefore, a fault diagnosis approach is important. In order to analyze a system, *Time Petri Net* (TPN) is proposed, which is a widely used modeling tool in real time system analysis. A detailed discussion on Petri net is given in Chapter 2. We partition the set of transitions into *observable* and *unobservable* ones. Some unobservable transitions are *fault* transitions corresponding to fault behaviors. Hence, the task in Part II is to estimate the firing of fault transitions by using the firings of observable transitions.

In Part II, we investigate the problem on *detection* and *isolation* of faults. In Part III, we focus on faults avoidance, and we consider two types of faults: deadlock and collision. The faults are discussed in multiple robots planning, which means multiple robots move from source places to their destinations in a shared environment and each robot has one or more path to follow. The shared environment is a map partitioned into *regions*. In this system, deadlock means some robot cannot reach their destinations, while we define collision as two robots appear in a region in which only one robot is allowed. In the solution of deadlock prevention, we propose a decentralized control policy with real time control. The decentralized control policy does not need any additional monitor to be implemented. In collision avoidance, we assume no real time control is applied, while the decentralized deadlock control policy is in real time. The control policy of collision avoidance consists of initial delays to each robot. The initial delay of a robot let the robot wait for a given time delay before it moves, and when it is moving, no additional control is applied.

2 TIME PETRI NET AND STATE CLASS GRAPH

This chapter introduces basic definitions, concepts and techniques about Petri net with and without time. Petri net is a bipartite graph containing places and transitions connected by arcs. It is a fundamental tool in many domains, e.g., transportation and manufacturing. In Petri net without time, structural analysis is very important, in order to avoid the state explosion problem. Structural concepts are given with examples, and then subclasses of Petri net are defined based on structural properties. In general cases, some properties of Petri net system can only be verified using the reachability set, for example, liveness. Therefore, reachability set and liveness are discussed. In order to analyze timed systems, Petri nets are extended. Time Peri net, a kind of Petri net with time, is defined such that time interval delays are associated with transitions. We propose a state estimation algorithm on timed Petri net. In the last part of this chapter, *state class graph*, an abstracted reachability graph of time Petri net, is defined. Algorithms to computing state class graph are given with an illustrative example.

2.1 INTRODUCTION

A *model* can come in many shapes, sizes and styles. It is important to emphasize that a model is not the real world but merely a human construct to help in better understanding of real world systems [1]. Many man made systems can naturally be modeled as Discrete Event Systems (DES), e.g., manufacturing systems [57, 52], transportation systems [36] and business processes [62]. A DES is an event driven system with discrete states, where the state evolution depends on the occurrence of discrete events [17]. DES has been applied to applications in various domains. Meanwhile, research on several disciplines, including system theory, computer science and operational research, improves the expressibility of DES. The events can represent both *regular* (or normal) and *faulty* (or abnormal) behaviors. For example, in a transportation system, a fault may be a car entering into a wrong lane or a malfunction of a traffic light. It is obviously important to ensure the safe and correct functioning of large-scale systems. The (fault) diagnosis is the process to detect and isolate the occurrence of faulty events. In the last decades, fault diagnosis of DES attracts a significant attention. An introduction to the subject and an overview can be found in [71], while more detailed results of fault diagnosis and state estimation in both untimed and timed DES are [47, 59, 65, 66, 67, 22, 54, 68].

Petri nets (PN) are used in this thesis to model complex systems. PN is a DES paradigm introduced by Carl Adam Petri in his Ph.D. dissertation [50] in early 1960's. Comparing with other modeling tools, PN has some advantages [55]:

1. Its formalism let both human and computer can analyze PN modeled systems conveniently. The mathematical formalism consists of matrices, which can be integrated into classical mathematical analysis methods, such as Linear Programming Problems (LPP). Using the graphical representation, the systems' features described by PN models, including concurrency, conflict, and synchronization, can easily be interpreted and understood by human.

2. PN models are compact representations of DES, due to its *bipartite* structure including *places* as local states and *transitions* meaning events. Even the state space of a system is enormous (sometimes infinite), a compact PN system can be obtained.

3. Formalisms like *automata* and *Markov chains* use symbolic unstructured global state, while in PN, states are distributed in places and they are numerical. Particularly, they are vectors of non-negative numbers.

4. Modeling in PN can be *top-down* or *bottom-up* based on the locality of places and transitions. In a bottom-up way, PN models can be built by constructing first the models of each subsystem and then, by composition, the global model is obtained. In order to build a PN model using top-down method, a high level PN

can be constructed first, and then expand places and transitions in it to represent the details in the target system.

In this chapter, we define PN and some extensions. First, Place/-Transition nets are defined and they are the basis of other classes of PN. Some structure related properties and subclasses of PN are introduced, which are widely used in the analysis of PN described systems. After that, we define Time Petri Nets (TPN), which have the ability to interpret timing information. TPN will be used in Part II. Because our works in Part II uses reachability analysis of TPN, an efficient and compact representation of the reachability space is important and is introduced in Section 2.5. Last, in Part III, we use a subclass of PN called S^3PR, which is a class of Place/Transition net. The common notation system used in S^3PR is slightly different from the one defined in Section 2.2. Therefore, we introduction the popular notations followed by a formal definition of S^3PR using these notations.

2.2 (UNTIMED) PETRI NET

2.2.1 *Basic concepts*

Petri net [48, 58] is one of the most widely used modeling paradigm in the field of Discrete Event Systems (DES). This section focuses on Place/Transition (P/T) nets and its structural and behavior properties.

Definition 2.1. *A* Petri net *(PN) is a* 4*-tuple* $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$*, where:*

- P *is the finite set of places,*
- T *is the finite set of transitions such that* $P \cup T \neq \emptyset$*,* $P \cap T = \emptyset$*,*
- $\mathbf{Pre}, \mathbf{Post} \in \mathbb{N}_{\geqslant 0}^{|P|\times|T|}$ *are the pre and post incidence matrices, where* $\mathbb{N}_{\geqslant 0}$ *is the set of non-negative integers.*

Let $p \in P$ and $t \in T$ be a place and a transition, respectively. The input places of t are indexed by $\mathbf{Pre}[\cdot, t]$ (the column of $\mathbf{Pre}$ correspondinf to t) and $\mathbf{Post}[\cdot, t]$ (the column in $\mathbf{Post}$ corresponding to t) denotes the indices of the output places of t.

Definition 2.2. *Let* $\mathcal{N}$ *be a PN. The support set of a place vector* $\overrightarrow{u} \in \mathbb{N}_{\geqslant 0}^{|P|}$[*] *is* $||\overrightarrow{u}|| = \{p|p \in P, \overrightarrow{u}[p] > 0\}$*, and the support set of a transition vector* $\overrightarrow{v} \in \mathbb{N}_{\geqslant 0}^{|T|}$ *is* $||\overrightarrow{v}|| = \{t|t \in T, \overrightarrow{v}[t] > 0\}$*.*

The input nodes of t (p) are ${}^\bullet t = ||\mathbf{Pre}[\cdot, t]||$ (${}^\bullet p = ||\mathbf{Post}[p, \cdot]||$) and the output nodes of t (p) are $t^\bullet = ||\mathbf{Post}[\cdot, t]||$ ($p^\bullet = ||\mathbf{Pre}[p, \cdot]||$). Let $X \subseteq P \cup T$, ${}^\bullet X = \bigcup_{n \in x} {}^\bullet n$ and $X^\bullet = \bigcup_{n \in X} n^\bullet$.

Definition 2.3. *A* Petri net system *(PNs) is a pair* $\langle \mathcal{N}, \mathbf{m}_0\rangle$*, where:*

[*] The vector $\overrightarrow{u}$ is also written as $\sum_{p \in P, \overrightarrow{u}[p]>0} \overrightarrow{u}[p]p$

- *$\mathcal{N}$ is a PN,*
- *$\mathbf{m}_0 \in \mathbb{N}_{\geqslant 0}^{|P|}$ is the initial marking.*

The state of a PNs is a marking representing the number of tokens in each place. The initial marking is the initial state of the system. The firings of transitions denotes the evolution of a PNs. A transition can be fired if it is *enabled* at a marking.

Definition 2.4. *Let $\mathcal{N}$ be a PN and $\mathbf{m}$ is a marking of $\mathcal{N}$. A transition $t \in T$ is* enabled *at the marking $\mathbf{m}$ if $\mathbf{m} \geqslant \mathbf{Pre}[\cdot, t]$ and denoted as $\mathbf{m}[t\rangle$.*

The marking $\mathbf{m}$ enables t when $\forall p \in {}^\bullet t$, $\mathbf{m}[p] \geqslant \mathbf{Pre}[p, t]$. Firing t at $\mathbf{m}$ generates a new marking $\mathbf{m}'$ and denoted as $\mathbf{m}[t\rangle\mathbf{m}'$. The *state equation* computes $\mathbf{m}'$ from $\mathbf{m}$ such that $\mathbf{m}' = \mathbf{m} - \mathbf{Pre}[\cdot, t] + \mathbf{Post}[\cdot, t]$.

A marking $\mathbf{m}'$ is reachable from $\mathbf{m}$ by firing a sequence of transitions $\sigma = t_{i1}t_{i_2} \cdots t_{in} \in T^*$, where $t_{ij} \in T$, $j = 1, 2, \ldots, n$ and T^* is the *Kleene closure* of T, is written as $\mathbf{m}[\sigma\rangle\mathbf{m}'$ and σ is a *firing sequence*. When $\mathbf{m}[\sigma\rangle\mathbf{m}'$, the fundamental state equation becomes $\mathbf{m}' = \mathbf{m} - \mathbf{Pre} \cdot \overrightarrow{\sigma} + \mathbf{Post} \cdot \overrightarrow{\sigma}$, where *firing count vector* $\overrightarrow{\sigma}$ counts how many times each transition in σ is fired.

An *inhibitor arc* connects a place to a transition. It disables the transition when the place has at least one token, while it enables the transition when the place has no token and the normal input places satisfy the classical enabling rule. We use ${}^\bullet X$ and $X^\bullet$, $X \subseteq P \cup T$ to denote the presets and postsets of X according to the normal arcs, respectively. For example, ${}^\bullet t = \{p|(p, t) \in F\}$ and $p^\bullet = \{t|(p, t) \in F\}$. An inhibitor arc starting from a bounded place can be transformed into a self-loop on its complementary place, which represents its capacity.

2.2.2 *Structural concepts*

PN has some structural objects and properties benefiting both modeling and analysis. The first one is *semiflow*. Right $((\mathbf{Post} - \mathbf{Pre}) \cdot \mathbf{x})$ and left $(\mathbf{y}^\mathsf{T} \cdot (\mathbf{Post} - \mathbf{Pre}))$ natural annullers of the token flow matrix are t- and p-semiflows, respectively. A semiflow $\boldsymbol{w}$ is minimal if not exists $\boldsymbol{w}'$ is a semiflow such that $||\boldsymbol{w}'|| \subset ||\boldsymbol{w}||$. A PN is *conservative* if $\exists \mathbf{y} > 0$ such that $\mathbf{y}^\mathsf{T} \cdot (\mathbf{Post} - \mathbf{Pre}) = 0$. If $\exists \mathbf{x} > 0$, $(\mathbf{Post} - \mathbf{Pre}) \cdot x = 0$, then the net is *consistent*.

Given p-semiflow $\mathbf{y}$, two related but different notions are:

- *conservation laws*: the equation $\mathbf{y}^\mathsf{T} \cdot \mathbf{m}_0 = \mathbf{y}^\mathsf{T} \cdot \mathbf{m}$ hold for any arbitrary initial marking $\mathbf{m}_0$ and every reachable marking $\mathbf{m}$ from $\mathbf{m}_0$.
- *conservative component*: $||\mathbf{y}||$, a set of places in the net conserving its weighted token content (the part of the net is noted as $\mathcal{N}_\mathbf{y}$.

T-semiflows identify potential cyclic behaviors. For example, let $\mathbf{x} \gneq 0$, $(\mathbf{Post} - \mathbf{Pre}) \cdot \mathbf{x} = 0$. If $\mathbf{x}$ is fireable from a marking $\mathbf{m}$, then $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}$, where the firing count vector of σ is $\mathbf{x}$.

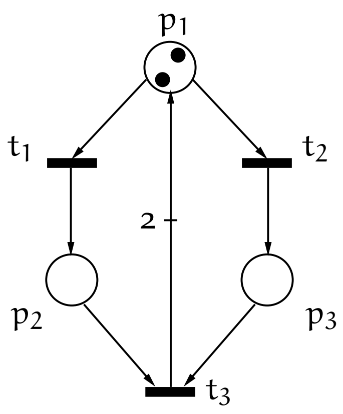


Figure 2.1: A PN has a p-semiflow and a t-semiflow.

Example 2.5. *Consider the PN in Figure 2.1 in which the initial marking is* $\mathbf{m}_0 = 2p_1$*, the weight of the arc from* t_3 *to* p_1 *is 2 and weights of other arcs are* 1*. The PN has a p-semiflow* $\mathbf{y} = [1\ 1\ 1]^T$ *and* $||\mathbf{y}|| = \{p_1, p_2, p_3\}$*. For any reachable marking from* $\mathbf{m}_0$*, there is* $\mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_3] = \mathbf{m}_0[p_1] + \mathbf{m}_0[p_2] + \mathbf{m}_0[p_3]$*. The PN has also a t-semiflow* $\mathbf{x} = [1\ 1\ 1]^T$ *meaning that if every transition is fired once at a marking, then the system returns to that marking.*

Second, *siphons* and *traps* are interesting structural concepts. A set of place $\mathcal{S} \subseteq P$ is a siphon if ${}^\bullet\mathcal{S} \subseteq \mathcal{S}^\bullet$. While a trap $\Theta \subseteq P$ satisfies $\Theta^\bullet \subseteq {}^\bullet\Theta$. A net system $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ is *deadlock-free* if $\forall \mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0), \exists t \in T$ such that $\mathbf{m}[t\rangle$. A transition t is *live* if $\forall \mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0)$, $\exists \mathbf{m}' \in \mathcal{R}(\mathcal{N}, \mathbf{m})$ so that $\mathbf{m}'[t\rangle$. A net system is live if every transition is live. We say that a transition t is *dead for a reachable marking* $\mathbf{m}$ if $\nexists \mathbf{m}' \in \mathcal{R}(\mathcal{N}, \mathbf{m})$, so that $\mathbf{m}'[t\rangle$. Notice that a transition is live if it is not dead for every reachable marking. A *P-semiflow* $\mathbf{y}$ is a P-indexed vector so that $\forall p \in P$, $\mathbf{y}[p] \in \mathbb{N}_{\geqslant 0}$. The *support* of a P-indexed vector $\mathbf{y}$ is the set $||y|| = \{p \in P | \mathbf{y}[p] \neq 0\}$. A P-semiflow $\mathbf{y}$ is *minimal* if there does not exist another P-semiflow $\mathbf{y}'$ such that $\mathbf{y}' \lneqq \mathbf{y}$. The *support* of a marking $\mathbf{m}$ is $||\mathbf{m}|| = \{p \in P | \mathbf{m}[p] > 0\}$. A place whose removal does not affect the behavior (here the set of fireable sequences) of the net system is called an *implicit place*.

2.2.3 *Subclasses of PN*

Structural constrains classify subclasses in PN. Some subclasses are defined below:

- *Ordinary PN* are PN whose arc weights are 1, i.e., $\mathbf{Pre}, \mathbf{Post} \in \{0, 1\}^{|P| \times |T|}$.
- *State machines* (SM) are ordinary PN in which each transition has only one input and one output transitions, i.e., $\forall t \in T, |{}^\bullet t| = |t^\bullet| = 1$.
- *Marked graphs* (MG) are ordinary PN in which each place has only one input place and one output place, i.e., $\forall p \in P, |{}^\bullet p| = |p^\bullet| = 1$.

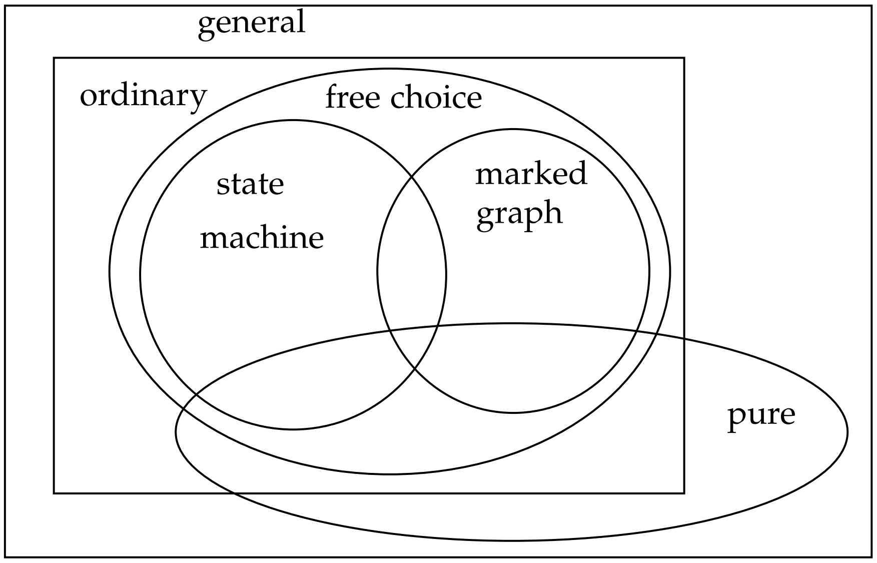


Figure 2.2: Subclasses of PN

- *Join free* (JF) nets are PN in which each transition has at most one input place, i.e., $\forall t \in T, |^\bullet t| \leqslant 1$.
- *Choice free* (CF) nets are PNs in which each place has at most one output transition, i.e., $\forall p \in P, |p^\bullet| \leqslant 1$.
- *Free choice* (FC) nets are ordinary PNs in which conflicts are always equal, i.e., $\forall t, t' \in T$, if $^\bullet t \cap {^\bullet t'} \neq \emptyset$, then $^\bullet t = {^\bullet t'}$.
- *Pure* nets have no self-loop. The *incidence matrix* of a pure net is $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$.

The graph in Figure 2.2 summaries these subclasses.

2.2.4 *Reachability and behavioral concepts*

The *reachability space* or (*reachability set*) of a PN system $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ contains markings reachable from $\mathbf{m}_0$.

Definition 2.6. *Let $\mathcal{N}$ be a PN and $\mathbf{m}$ be a marking of PN. The set of* reachable markings *of $\mathbf{m}$ is $\mathcal{R}(\mathcal{N}, \mathbf{m})$ such that $\forall \mathbf{m}' \in \mathcal{R}(\mathcal{N}, \mathbf{m})$, $\exists \sigma \in T^+$, $\mathbf{m}[\sigma\rangle\mathbf{m}'$.*

Definition 2.7. *Let $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ be a PN system. Its* reachability set *is $\mathcal{R}(\mathcal{N}, \mathbf{m}_0)$.*

Using the notation of reachable markings, some concepts related to the reachability set is defined.

Definition 2.8. *Let $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ be a PN system.*

- *A place $p \in P$ is bounded if $\exists b \in \mathbb{N}, \forall \mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0)$, $\mathbf{m}[p] \leqslant b$.*
- *The PN system is bounded if $\forall p \in P$, p is bounded.*

Definition 2.9. *Let $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ be a PN system. It is live if $\forall t \in T$ and $\forall \mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0)$, $\exists \mathbf{m}' \in \mathcal{R}(\mathcal{N}, \mathbf{m})$ such that $\mathbf{m}'[t\rangle$.*

If a siphon is emptied, no token can enter in it and transitions that have input places in the siphon are dead. Chapter 10 discusses how to avoid deadlocks by controlling siphons in some classes of PN.

Definition 2.10. *Let $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ be a PN system. A marking $\mathbf{m}_h$ is a* home state *if $\forall \mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0)$, $\mathbf{m}_h \in \mathcal{R}(\mathcal{N}, \mathbf{m})$.*

A PN system $\langle \mathcal{N}, \mathbf{m}_0 \rangle$ is *reversible* if $\mathbf{m}_0$ is a home state.

2.3 PETRI NET WITH TIME

2.3.1 *Motivation*

Place/Transition net has been introduced in the previous section for analyzing logical properties of DES. However, its usage is limited due to lacking the ability to describe time durations of the occurrence of events (activities) in a system. P/T net has constrained use in the fields of bottleneck identification, resource optimization, computation of the execution time of a given process, and so on. In this section, a kind of Petri net with time is introduced with the computation of its abstracted state space.

2.3.2 *Time Petri net*

An extension of a P/T net via associating time interval delays is presented in [46]. This approach uses transitions with time interval delays to address the idea of modeling the duration of activities of a system. This type of PN with time is called *Time Petri Net* (TPN).

Definition 2.11. *A* Time Petri Net *is a pair $\mathcal{N}_t = \langle \mathcal{N}, I \rangle$, where:*

- *$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ is a P/T net defined in Definition 2.1,*
- *$I : T \to \mathbb{Q}_{\geqslant 0} \times (\mathbb{Q}_{\geqslant 0}\{\infty\})$ is the time function associating a* time interval *with each transition and $\mathbb{Q}_{\geqslant 0}$ is the set of non-negative rational numbers.*

Definition 2.12. *A TPN system is a triple $\langle \mathcal{N}, I, \mathbf{m}_0 \rangle$, where:*

- *$\mathcal{N}$ and I is a TPN in Definition 2.11,*
- *$\mathbf{m}_0 \in \mathbb{N}^{|P|}$ is the initial marking.*

The enabling rule of transitions is the same in the TPN $\mathcal{N}_t = \langle \mathcal{N}, I \rangle$ as in the corresponding PN $\mathcal{N}$. Let the time interval associated with t be $I(t) = [l, u]$. The interval means that the duration of the activity represented by t is from l to u. When the activity can occur, it needs at least l time units to finish, and it will (must) finish within u time units. In the TPN, the corresponding firing rules are that when t is enabled in the P/T net:

- it cannot fire earlier than l time units;
- it can fire between l and u time units;

- it must fire if u time units have been reached.

This and all following chapters use single server semantics meaning that a transition cannot be enabled simultaneously more than once.

Definition 2.13. *A* timed PN *is a triple* $\langle \mathcal{N}, \theta, m_0 \rangle$, *where* $\mathcal{N}$ *is a PN and* $\theta \in \mathbb{R}_{\geqslant 0}^{|T|}$ *is the time vector that associates to each transition* t_j *a consistent time delay,* $\theta_j = \theta[t_j]$.

2.3.3 *TPN with unobservable transitions*

The set of transitions T is partitioned into two: $T = T_o \cup T_u, T_o \cap T_u = \emptyset$, where T_o is the set of *observable* transitions, whose firing can be detected by an external observer, and T_u is the set of *unobservable* transitions. The firing sequence σ_o is an observable one, if for all $t \in \sigma_o$, then $t \in T_o$; σ_u is an unobservable firing sequence, if for all $t \in \sigma_u$, then $t \in T_u$. An observation function $\lambda : \sigma \to T_o^*$ extracts a sequence of observable transitions $\lambda(\sigma)$ from σ. Let $\sigma = \sigma_{u1}\sigma_{o1}\sigma_{u2}\sigma_{o2} \cdots \sigma_{un}$, then $\lambda(\sigma) = \sigma_{o1}\sigma_{o2} \cdots \sigma_{on-1}$. In figures, observable transitions are represented by white rectangles, while unobservable ones are black rectangles. The length of σ is $|\sigma|$ meaning the number of transitions in σ.

Definition 2.14. *The unobservable subnet of a TPN* $\langle \mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle, I \rangle$ *is* $\langle \mathcal{N}_u = \langle P, T_u, \mathbf{Pre}_u, \mathbf{Post}_u \rangle, I_u \rangle$, *where:*

1. P *is the set of places;*
2. T_u *is the set of unobservable transitions of* $\mathcal{N}$;
3. $\mathbf{Pre}_u$ *and* $\mathbf{Post}_u$ *are pre and post incidence matrices restricted to* T_u;
4. $I_u : T_u \to \mathbb{Q}_0 \times \mathbb{Q}_0 \cup \{\infty\}$.

■

2.4 STATE ESTIMATION OF TIMED PETRI NETS

2.4.1 *Basis marking in timed Petri nets*

We present an on-line algorithm for state estimation of timed choice-free Petri nets. We assume that the net structure and initial marking are known, and that the transitions is composed by observable and unobservable transitions. The proposed algorithm works on-line as three steps. First, it waits for an observation, which is the firing of an observable transition, and computes possible markings, which construct the set of basis markings. Second, the set of time equations is updated from the set of basis markings and reduced by removing inconsistent equations. Third, the set of basis markings is reduced according to the reducing process of the set of time equations. The extension of the algorithm to general nets is discussed.

We make the following assumptions:

(A1) The initial marking and net structure are known.

(A2) The T_u-induced subnet is acyclic.

(A3) The time durations of observable transitions are known, while the time durations of unobservable transitions are unknown.

Even if the initial marking is known, because of the partial observation, the state of timed PN's cannot be determined by observation. To characterize the possible set of markings we use a subset of it, which is called the set of basis markings. Knowing this set of basis markings, the consistent markings, which are the possible markings in which the net system can be obtained by simply firing the unobservable transitions from the basis markings.

Definition 2.15. *([28]) Given a marking* $\mathbf{m}$ *and an observable transition* $t \in T_o$*, we define the set of* explanations *of* t *at* $\mathbf{m}$ *as*

$$\Sigma(\mathbf{m}, t) = \{\sigma \in T_u^* | \mathbf{m}[\sigma\rangle \mathbf{m}', \mathbf{m}' \geqslant \mathbf{Pre}[\cdot, t]\}.$$

The set of minimal explanations *of* t *at* $\mathbf{m}$ *as*

$$\Sigma_{min}(\mathbf{m}, t) = \{ \quad \sigma \in \Sigma(\mathbf{m}, t) | \nexists \sigma' \in \Sigma(\mathbf{m}, t) : \boldsymbol{\sigma}' \lneqq \boldsymbol{\sigma}\},$$

where $\boldsymbol{\sigma}' \lneqq \boldsymbol{\sigma}$ *means that for each transition* t*, there is* $\boldsymbol{\sigma}'[t] < \boldsymbol{\sigma}[t]$.

In the following, the set of basis markings without time is introduced. The set of basis markings of observation w is $\mathcal{M}_b(w)$ and denotes the possible markings according to w.

Definition 2.16. *The set of basis markings of observation* $w = \nu t$ *is defined as:*

$$\mathcal{M}_b(w) = \{\mathbf{m} \in \mathbb{N}_{\geqslant 0}^{|P|} | \forall \mathbf{m}' \in \mathcal{M}_b(\nu) : \forall \sigma \in \Sigma_{min}(\mathbf{m}', t), \\ \mathbf{m}'[\sigma t\rangle \mathbf{m}\},$$

while $\mathcal{M}_b(\epsilon) = \{\mathbf{m}_0\}$ *for empty word* ϵ.

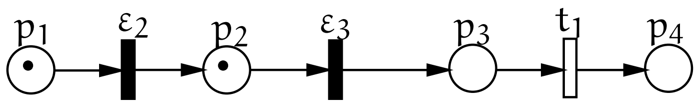


Figure 2.3: Example of the set of basis markings

Example 2.17. *Let us consider the PN's in Fig. 2.3 with* $\mathbf{m}_0 = [1, 1, 0, 0]^T$. *The unobservable transitions are* ε_2 *and* ε_3*, while the observable transition is* t_1*. Assume* t_1 *has been observed.*

The set of basis markings with no observation is $\mathcal{M}_b(\epsilon) = \{\mathbf{m}_0\}$*. When* t_1 *is observed, the set of explanations is* $\Sigma(\mathbf{m}_0, w) = \{\sigma_1 = \varepsilon_3, \sigma_2 = \varepsilon_2\varepsilon_3\}$ *and the set of minimal explanations is* $\Sigma_{min}(\mathbf{m}_0, w) = \{\sigma_1\}$*. By firing* $\sigma_1 t_1$*, the marking* $\mathbf{m}_1 = [1, 0, 0, 1]^T$ *is obtained and the new set of basis marking is* $\mathcal{M}_b(t_1) = \{\mathbf{m}_1\}$.

For a marking m in the set of basis markings, there is σ such that $m_0[\sigma\rangle m$. The sequence σ is composed by the observable transitions and unobservable firing sequences, which are minimal explanations. In order to represent the firing sequences that drive the marking from m_0 to m, based on the set of minimal explanation, we present the set of minimal firing sequences.

Definition 2.18. *Given a marking m and an observation word $w = t_{i1}\ t_{i2} \cdots t_{i,n-1}\ t_{in}$, we define the set of firing sequences to reach m, consistent with w as:*

$$\Gamma(m, w) = \{\sigma \in T^* | \sigma = \sigma_1^u t_{i1} \sigma_2^u t_{i2} \cdots t_{i,n-1} \sigma_n^u, \\ m_0[\sigma t_{in}\rangle m\}.$$

Based on $\Gamma(m, w)$, we define the set of minimal firing sequences Γ_{min} (m, w) as

$$\Gamma_{min}(m, w) = \{\sigma \in \Gamma(m, w) | \forall \sigma^u \in \sigma, \sigma^u \textit{ is a} \\ \textit{minimal explanation.}\}$$

Definition 2.19. *The set of basis markings at time τ of a timed Petri net can be easily defined as*

$$\mathcal{M}_b(w, \tau) = \{m \in \mathcal{M}_b(w) | \exists \sigma \in \Gamma_{min}(m, w), \sigma = \sigma' t, \\ \lambda(\sigma t) = w, t \textit{ is observed at } \tau.\}$$

2.4.2 *Time Duration of Firing Sequence*

In order to estimate the state of a timed PN, it is important to know the time duration of a firing sequence. In this section, we define and analyze such time duration.

Let us consider a firing sequence $\sigma = t_1 t_2 \cdots t_n$. The time duration of σ is denoted by $\iota(\sigma)$ and it is defined as the time duration from the enabling of t_1 to the firing of t_n:

$$\iota(\sigma) = \tau_n - (\tau_1 - \theta_1). \tag{2.1}$$

Proposition 2.20. *Let $\sigma = t_1 t_2 \cdots t_n$. The following two conditions are satisfied:*

- $$\max\{\theta_1, \ldots, \theta_n\} \leqslant \iota(\sigma) \leqslant \sum_{i=1}^{n} \theta_i. \tag{2.2}$$

- *If one and only one transition from σ is enabled at each time instant, then*

$$\iota(\sigma) = \sum_{i=1}^{n} \theta_i \tag{2.3}$$

Proof. *If there exists overlapping of time durations, the time duration of the firing sequence is less than the sum of the time durations of all transitions* (2.2)*. If there is no overlapping, then* (2.3) *holds.* □

The previous proposition can be generalized to sequences that can be partitioned into subsequences. For example, if $\sigma = \sigma_1\sigma_2\cdots\sigma_n$ and at each time moment, the enabled transitions belong to one and only one subsequence σ_i, then:

$$\iota(\sigma) = \iota(\sigma_1) + \iota(\sigma_2) + \cdots + \iota(\sigma_n). \tag{2.4}$$

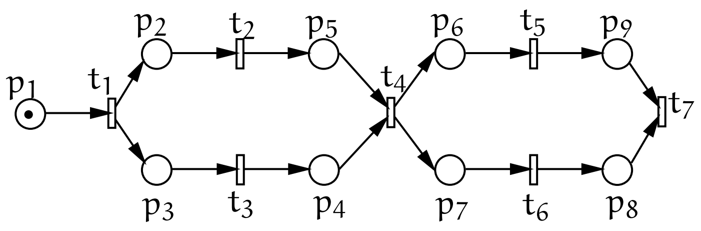


Figure 2.4: Example of $\iota(\sigma) = \iota(\sigma_1) + \iota(\sigma_2) + \cdots + \iota(\sigma_n)$

Example 2.21. *Let us consider the PN in Fig.* 2.4 *with* $\mathbf{m}_0 = p_1$ *and* $\theta = [1,2,3,1,2,3,1]^T$*. Since it is a deterministic PN, the following observed word is obtained* $w = t_1t_2t_3t_4t_5t_6t_7$ *at the following time instants* $1,3,4,5,7,8,9$*.*

Let us write w *as* $w = \sigma = \sigma_1\sigma_2\sigma_3\sigma_4\sigma_5$*, with* $\sigma_1 = t_1, \sigma_2 = t_2t_3, \sigma_3 = t_4, \sigma_4 = t_5t_6, \sigma_5 = t_7$*. According to* (2.1)*, the time durations are* $\iota(\sigma) = 9, \iota(\sigma_1) = 1, \iota(\sigma_2) = 3, \iota(\sigma_3) = 1, \iota(\sigma_4) = 3, \iota(\sigma_5) = 1$*. Since the condition in*(2.4) *is satisfied,*

$$\begin{aligned}\iota(\sigma) &= \iota(\sigma_1) + \iota(\sigma_2) + \iota(\sigma_3) + \iota(\sigma_4) + \iota(\sigma_5)\\ &= 1 + 3 + 1 + 3 + 1 = 9.\end{aligned}$$

2.4.3 *State estimation of choice-free nets*

The state estimation mainly includes three steps: 1. the set of basis markings is computed without considering time; 2. the set of time equations is obtained; 3. the set of basis markings is reduced based on the time information. Comparing with the basis marking, our state estimation algorithm takes time information into account. Moreover, it is appliable to choice-free nets, while the basis marking approach on untimed PN can be used on a wider range of PN.

2.4.3.1 *Compute* $\mathcal{M}_b(wt_j, \tau_j)$

The set of basis markings at time $\tau = 0$ is $\mathcal{M}_b(\epsilon, 0) = \{\mathbf{m}_0\}$. Let us assume that the current set of basis markings at time τ is $\mathcal{M}_b(w, \tau)$, where w is the actual observation. When the firing of a new transition t_j is observed at time τ_j, the following operations should be performed in order to compute $\mathcal{M}_b(wt_j, \tau_j)$.

1. Let $\mathcal{M}_b(wt_j, \tau_j) = \emptyset$,
2. For each $\mathbf{m} \in \mathcal{M}_b(w, \tau)$,
    a) compute $\Sigma_{min}(\mathbf{m}, t_j)$,
    b) let $\mathcal{M}' = \{\mathbf{m}' | \mathbf{m}[\sigma t_j\rangle \mathbf{m}', \sigma \in \Sigma_{min}(\mathbf{m}, t_j)\}$,
    c) let $\mathcal{M}_b(wt_j, \tau_j) = \mathcal{M}_b(wt_j, \tau_j) \cup \mathcal{M}'$.

For each basis marking $\mathbf{m}$ of the previous set, the set of minimal explanations is computed in $\Sigma_{min}(\mathbf{m}, t_j)$. Then, when t_j is observed after the firing of the minimal explanations of $\Sigma_{min}(\mathbf{m}, t_j)$ from $\mathbf{m}$, the new set of basic markings is obtained.

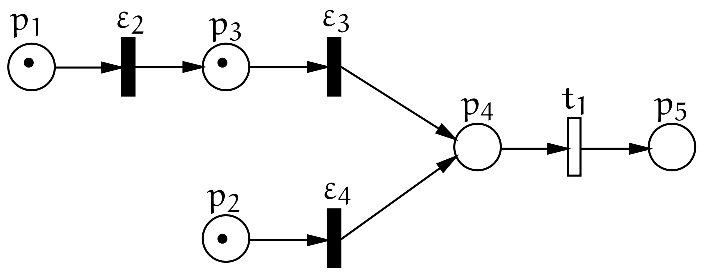


Figure 2.5: PN system used in Ex. 2.22

Example 2.22. *Let us consider the PN's in Fig. 2.5 with* $\theta_1 = 1$ *and* $\mathbf{m}_0 = [1, 1, 1, 0, 0]^T$. *The set of minimal firing sequences for the empty word is* $\Gamma_{min}(\mathbf{m}_0, \epsilon) = \emptyset$, *and the set of basis marking at time* 0 *is* $\mathcal{M}_b(\epsilon, 0) = \{\mathbf{m}_0\}$.

If $w = t_1$ *is observed at time* 4, $\mathcal{M}_b(t_1, 4)$ *is computed as follows.*

- $\mathcal{M}_b(t_1, 4) = \emptyset$.
- $\Sigma_{min}(\mathbf{m}_0, t_1) = \{\varepsilon_3, \varepsilon_4\}$.
- $\mathcal{M}' = \{\mathbf{m}_1 = [1, 0, 1, 0, 1]^T, \mathbf{m}_2 = [1, 1, 0, 0, 1]^T\}$, *where* $\mathbf{m}_0[\varepsilon_4 t_1\rangle \mathbf{m}_1$, $\mathbf{m}_0[\varepsilon_3 t_1\rangle \mathbf{m}_2$.
- $\mathcal{M}_b(t_1, 4) = \{\mathbf{m}_1, \mathbf{m}_2\}$.

The sets of minimal firing sequences are Γ_{min} $(\mathbf{m}_1, w) = \{\varepsilon_4\ t_1\}, \Gamma_{min}$ $(\mathbf{m}_2, w) = \{\varepsilon_3\ t_1\}$. *Observe that the set of basis markings is simply obtained from* $\mathbf{m}_0$ *firing* $w = t_1$ *and all those unobservable transitions that are* strictly *necessary to enable it.*

2.4.3.2 *Obtain the set of time equations*

The set of basis markings in the previous section is computed without considering any time notion. Assuming that the time durations associated to the unobservable transitions are not known, in this section we provide a procedure to obtain a set of equations to characterize all possible time durations associated to these unobservable transitions. It will be shown also how this set of time equations can be used to remove those time-inconsistent markings from the set of basis markings.

Let us assume that the time instant at which t_j was observed is τ_j, while the current set of basis markings is $\mathcal{M}_b(wt_j, \tau_j)$. To each set of basis markings we associate a *set of time equations*. These equations are obtained as the union of different equations. Let $\Gamma = \cup_{m \in \mathcal{M}_b(wt_j,\tau_j)} \Gamma_{min}\ (m, wt_j)$ be the set of all minimal firing sequences of all basis markings. The following time equation is obtained:

$$min\{\iota(\Gamma)\} = \tau_j$$

where, $\iota(\Gamma)$ is the set of time durations of each sequence in Γ.

Example 2.23. *In Example 2.22, the set of basis markings at time* 4 *has been computed. The set of minimal firing sequences is* $\Gamma_{min}\ (m_1, t_1) = \{\varepsilon_4\ t_1\}$ *and* $\Gamma_{min}\ (m_2, t_1) = \{\varepsilon_3\ t_1\}$*. Therefore,* $\Gamma = \{\varepsilon_3 t_1, \varepsilon_4 t_1\}$ *and the time equation is:*

$$o_4 = \min\{\iota(\varepsilon_3 t_1), \iota(\varepsilon_4 t_1)\} = 4$$

This has the following interpretation: because t_1 *has been fired at* 4 *and since for its firing,* ε_3 *or* ε_4 *should fire the firing delay of at least one of the following sequences* $\varepsilon_3 t_1$ *and* $\varepsilon_4 t_1$ *should be* 4*.*

If t_1 *is observed again at time* 6*, the sets of minimal explanations are*

$$\Sigma_{min}(m_1, t_1) = \{\varepsilon_3\}, \ \ \Sigma_{min}(m_2, t_1) = \{\varepsilon_4, \varepsilon_2\varepsilon_3\}.$$

implying

$$\Gamma_{min}(m_1, t_1) = \{\varepsilon_4 t_1 \varepsilon_3 t_1\}$$

and

$$\Gamma_{min}(m_2, t_1) = \{\varepsilon_3 t_1 \varepsilon_4 t_1, \varepsilon_3 t_1 \varepsilon_2 \varepsilon_3 t_1\}$$

corresponding to the set of basis markings:

$$\mathcal{M}_b(t_1 t_1, 6) = \{m_3 = [1, 0, 0, 0, 2]^T, m_4 = [0, 1, 0, 0, 2]^T\}.$$

while the corresponding time equation is

$$o_6 = \min\{\iota(\varepsilon_4 t_1 \varepsilon_3 t_1), \iota(\varepsilon_3 t_1 \varepsilon_4 t_1), \iota(\varepsilon_3 t_1 \varepsilon_2 \varepsilon_3 t_1)\} = 6.$$

Let us analyze the time durations of the sequences in o_6*. First of all, according to the definition of the time duration of a sequence,* $\iota(\varepsilon_4 t_1 \varepsilon_3 t_1)$ *and* $\iota(\varepsilon_3 t_1 \varepsilon_4 t_1)$ *provides the same information. Because both of* $\varepsilon_3 t_1$ *and* $\varepsilon_4 t_1$ *are enabled at time* 0 *(the initial time), the one with the shorter time duration is fired first, and then the another one is fired. The time durations of the two firing sequence are same, which is*

$$max\{min\{\theta_3, \theta_4\} + \theta_1, max\{\theta_3, \theta_4\}\} + \theta_1$$

Hence one of this sequence can be removed from o_6*. Removing for example the second one, we obtain:*

$$o_6 = \min\{\iota(\varepsilon_4 t_1 \varepsilon_3 t_1), \iota(\varepsilon_3 t_1 \varepsilon_2 \varepsilon_3 t_1)\} = 6.$$

According to o_4, $\theta_3 \geqslant 4 - \theta_1 = 3$. *We will show that in* o_6, $\iota(\varepsilon_3 t_1\ \varepsilon_2\ \varepsilon_3\ t_1) > 6$, *meaning that it is never the one that gives the minimum and can be removed:*

$$\iota(\varepsilon_3 t_1 \varepsilon_2 \varepsilon_3 t_1) \geqslant \theta_3 + \theta_3 + \theta_1 = 2\theta_3 + \theta_1 \geqslant 7$$

Therefore, $\varepsilon_3 t_1 \varepsilon_2 \varepsilon_3 t_1$ *is inconsistent with the time information. It can be deleted from* o_6, *so,*

$$o_6 = \iota(\varepsilon_4 t_1 \varepsilon_3 t_1) = 6.$$

and the corresponding basis marking should be removed, i.e.,

$$\mathcal{M}_b(t_1 t_1, 6) = \{\mathbf{m}_3 = [1, 0, 0, 0, 2]^T\}.$$

As it was illustrated by the previous example, some basis markings are time-inconsistent with the observation. On the other hand, some time equations that are obtained can be redundant.

The idea to reduce the time duration $\iota(\sigma_j)$ from o_j according to O is in three steps: First, divide σ_j into subsequences according to the condition of (2.4), that is, let σ_j' is a subsequence, there is $\iota(\sigma_j) = \sum \iota(\sigma_j')$; Second, find $\sigma_{o,j}$ from O that $\sigma_{o,j}$ is a subsequence of σ_j'. Based on (2.4), there is $\iota(\sigma_j') \geqslant \iota(\sigma_{o,j})$. Last, we can get that the time duration of σ_j is greater or equal than the sum of time durations of all $\sigma_{o,j}$. If the latter one is greater than τ_j, which is the time instant when o_j is computed, $\iota(\sigma_j)$ should be removed from o_j.

Proposition 2.24. *Let* O *be the current set of time equations that*

$$O = \left\{ \begin{array}{c} \min\{\iota(\sigma_{1,1}), \iota(\sigma_{1,2}), \ldots, \iota(\sigma_{1,k_1})\} = \tau_1, \\ \min\{\iota(\sigma_{2,1}), \iota(\sigma_{2,2}), \ldots, \iota(\sigma_{2,k_2})\} = \tau_2, \\ \vdots \\ \min\{\iota(\sigma_{q,1}), \iota(\sigma_{q,2}), \ldots, \iota(\sigma_{q,k_q})\} = \tau_q, \end{array} \right\}$$

while o_j *is the one obtained at time* τ_j *that*

$$o_j : \min\{\iota(\sigma_{j,1}), \iota(\sigma_{j,2}), \ldots, \iota(\sigma_{j,k_j})\} = \tau_j.$$

and $q, k_q, j \in \mathbb{N}_{>0}$.

Let $\iota(\sigma_j) \in \{\iota(\sigma_{j,1}), \iota(\sigma_{j,2}), \ldots, \iota(\sigma_{j,k_j})\}$. *If* $\iota(\sigma_j)$ *can be represented as*

$$\iota(\sigma_j) \geqslant \sum_{i=1}^{q} \sum_{r=1}^{k_i} a_{i,r} \iota(\sigma_{i,r}),$$

where $a_{i,r} > 0, i = 1, 2, \ldots, q, r = 1, \ldots, k_i$, *that* $\sum_{i=1}^{q} \sum_{r=1}^{k_i} a_{i,r} \iota(\sigma_{i,r}) \geqslant \tau_j - \theta_j$, *then* $\iota(\sigma_j)$ *should be removed from* o_j.

If the condition of Prop. 2.24 is satisfied, $\iota(\sigma_j)$is greater than $\tau_j - \theta_j$ and it is not consistent with the observation, i.e., if the observation is derived from σ_j, then the time of the observation should be greater than τ_j.

Proposition 2.25. *When the set of time equations keeps the same infinitely, it holds that*

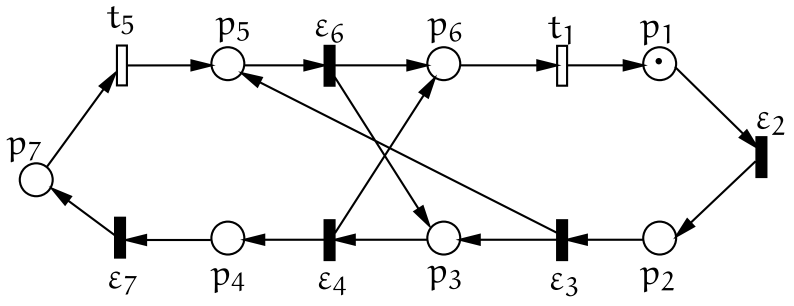


Figure 2.6: Example of the algorithm

1. *all observable transitions have been observed,*
2. *all firing sequences in the set of time equations are periodical.*

The proposition above gives two necessary conditions of the time that the set of time equations stops updating. The set of time equations stops updating means that there is no new information to update the set. For the first condition, if these exists observable transitions that have not been observed, then there exists new observation to update the set of time equations; For the second condition, if there exists transitions that are not periodical, then there may exists new firing sequences for updating the set of time equations.

2.4.4 *Algorithm for estimating the state*

In this section, we present an algorithm to estimate the state of timed PN's with unobservable transitions. After observing the firing of an observable transition, there are four steps to estimate the state:

1 Compute the set of basis markings $\mathcal{M}_b(wt_j, \tau_j)$ based on current observation t_j at τ_j.

2 Compute the time equation o_{τ_j} according to the set of basis markings $\mathcal{M}_b(wt_j, \tau_j)$.

3 Reduce the time equation o_{τ_j} based on Prop. 2.24 according to O.

4 Reduce the set of basis markings $\mathcal{M}_b(wt_j, \tau_j)$ according to the reduced set of time equations O.

Example 2.26. *For the PN in Fig. 2.6 with observable transitions* t_1 *and* t_5, $\theta_1 = \theta_5 = 1$, *and the initial marking* $\mathbf{m}_0 = [1, 0, 0, 0, 0, 0, 0]^T$.

The PN model in Fig. 2.6 can be reduced to the net in Fig. 2.7. The initial marking of the reduced net is $\mathbf{m}_0 = [p_{12}, p_3, p_4, p_5, p_6, p_7]^T = [1, 0, 0, 0, 0, 0]^T$.

We observe t_1 *at* $5, 9$ *and* t_5 *at* 10.

At time 0, *the set of basis markings* $\mathcal{M}_b(\epsilon, 0) = \{\mathbf{m}_0\}$ *and the set of time equations* $O = \emptyset$.

At time 6, t_1 *is observed, i.e.,* $w_6 = t_1$. *The set of minimal explanations* $\Sigma_{min} = (\mathbf{m}_0, t_1) = \{\sigma_1 = \varepsilon_{23}\varepsilon_6, \sigma_2 = \varepsilon_{23}\varepsilon_4\}$. *By firing* $\sigma_1 t_1$

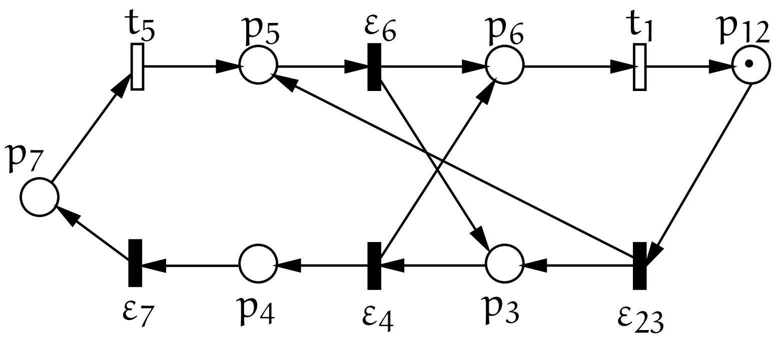


Figure 2.7: The reduced net of the PN in Fig. 2.6

and $\sigma_2 t_1$*, the set of basis markings is obtained,* $\mathcal{M}_b(w_6, 6) = \{\mathbf{m}_1 = [1,2,0,0,0,0]^T, \mathbf{m}_2 = [1,0,1,1,0,0]^T$*, and the sets of minimal firing sequences are* $\Gamma_{min}(\mathbf{m}_1, w_6) = \{\sigma_1 t_1\}, \Gamma_{min}(\mathbf{m}_2, w_6) = \{\sigma_2 t_1\}$*. The time equation at time 6 is* $\min\{\iota(\sigma_1 t_1), \iota(\sigma_2 t_1)\} = 6$*, and the set of time equations is obtained as* $O = \{\min\{\iota(\sigma_1 t_1), \iota(\sigma_2 t_1)\} = 6\}$*.*

At time 9, t_1 *is observed, i.e.,* $w_9 = t_1 t_1$*. The sets of minimal explanations are*

$$\Sigma_{min}(\mathbf{m}_1, t_1) = \{\sigma_1, \varepsilon_4\},$$
$$\Sigma_{min}(\mathbf{m}_2, t_1) = \{\sigma_2, \varepsilon_6\}.$$

Firing $\sigma_1 t_1$ *and* $\varepsilon_4 t_1$ *at* $\mathbf{m}_1$*, we get* $\mathbf{m}_3 = [1,4,0,0,0,0]^T$*,* $\mathbf{m}_4 = [2,1,1,0,0,0]^T$*, respectively; Firing* $\sigma_2 t_1$ *and* $\varepsilon_6 t_1$ *at* $\mathbf{m}_2$*, the markings are* $\mathbf{m}_4$ *and* $\mathbf{m}_5 = [1,0,2,2,0,0]^T$*, respectively. The set of basis markings at time 9 is obtained as* $\mathcal{M}_b(w_9, 9) = \{\mathbf{m}_3, \mathbf{m}_4, \mathbf{m}_5\}$ *and the sets of minimal firing sequences are*

$$\Gamma_{min}(\mathbf{m}_3, w_9) = \{\sigma_3 = \sigma_1 t_1 \sigma_1 t_1\},$$
$$\Gamma_{min}(\mathbf{m}_4, w_9) = \{\sigma_4 = \sigma_1 t_1 \varepsilon_4 t_1, \sigma_6 = \sigma_2 t_1 \varepsilon_6 t_1\},$$
$$\Gamma_{min}(\mathbf{m}_5, w_9) = \{\sigma_5 = \sigma_2 t_1 \sigma_2 t_1\}.$$

From the sets of minimal firing sequences, we obtain the time equation at time 9:

$$o_9 : \min\{\iota(\sigma_3), \iota(\sigma_4), \iota(\sigma_5), \iota(\sigma_6)\} = 9.$$

The firing sequence σ_3 *and* σ_5 *satisfy Prop. 2.24, and there is*

$$\iota(\sigma_3) > 2 \times \iota(\sigma_1) + \theta_1, \iota(\sigma_5) > 2 \times \iota(\sigma_2) + \theta 1.$$

From O, we have $\iota(\sigma_3) \geqslant 6 - 1 = 5$ *and* $\iota(\sigma_3) \geqslant 6 - 1 = 5$*. It means that* $\iota(\sigma_3) \geqslant 11, \iota(\sigma_5) \geqslant 11$ *(according to O), which are not consistent with the observation, then* o_9 *is reduced to* $o_9 : \min\{\iota(\sigma_4), \iota(\sigma_6)\} = 8$*. The set of time equations is*

$$O = \left\{ \begin{array}{l} \min\{\iota(\sigma_1 t_1), \iota(\sigma_2 t_1)\} = 6, \\ \min\{\iota(\sigma_4), \iota(\sigma_6)\} = 9. \end{array} \right\}$$

The set of basis markings is reduced to $\mathcal{M}_b(w_9, 9) = \{\mathbf{m}_4\}$*.*

At time 10, t_5 *is observed, i.e.,* $w_{10} = t_1 t_1 t_5$*. The set of minimal explanations* $\Sigma_{min} = (\mathbf{m}_4, t_5) = \{\varepsilon_7\}$*. Firing* $\varepsilon_7 t_5$*, the set of basis markings*

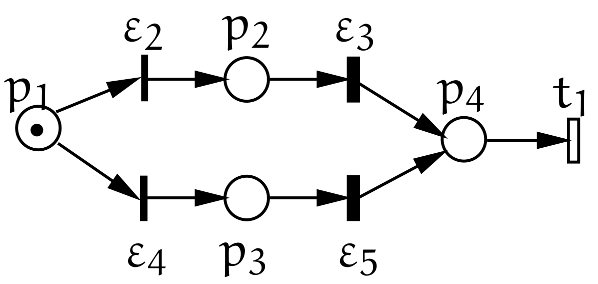


Figure 2.8: Example of PN's with choice

$\mathcal{M}_b(w_{10}, 10) = \{\mathbf{m}_6 = [2,1,0,1,0,0]^T\}$, *and the set of minimal firing sequences is* $\Gamma_{min}(\mathbf{m}_6, w_{10}) = \{\sigma_7 = \sigma_4\varepsilon_7 t_5, \sigma_8 = \sigma_6\varepsilon_7 t_5\}$*. The time equation obtained at time* 10 *is* $o_{10} : min\{\iota(\sigma_7), \iota(\sigma_8)\} = 10$*. The set of time equations is*

$$O = \left\{ \begin{array}{l} min\{\iota(\sigma_1 t_1), \iota(\sigma_2 t_1)\} = 6, \\ min\{\iota(\sigma_4), \iota(\sigma_6)\} = 9, \\ min\{\iota(\sigma_7), \iota(\sigma_8)\} = 10. \end{array} \right\}$$

2.4.5 *Discussion*

The min operation cannot be applied to PN's with choices. Let us consider the PN's in Fig. 2.8 with immediate transitions ε_2 and ε_4, $\theta_1 = 1, \theta_2 = \theta_4 = 0$, and initial marking $\mathbf{m}_0 = [1,0,0,0]^T$. Assume t_1 is observed at time 4. It means that one of $\iota(\varepsilon_2\varepsilon_3)$ and $\iota(\varepsilon_4\varepsilon_5)$ is 4, and another one can be greater or less than 4 or equal to 4. It means that we cannot say the minimal one of $\iota(\varepsilon_2\varepsilon_3)$ and $\iota(\varepsilon_4\varepsilon_5)$ is 4. Therefore, $min\{\iota(\varepsilon_2\varepsilon_3), \iota(\varepsilon_4\varepsilon_5)\} = 4$ cannot be applied.

To apply the algorithm to general nets with choices, we can treat each choice separately, i.e., enumerate all possible combinations of firing sequences. It is similar with the state estimation of the untimed PN's.

2.5 STATE CLASS GRAPH

2.5.1 *State class graph and its construction*

Because time is continuous, the reachability graph of a TPN may contain an infinite number of states. In order to cope with this problem, the *State Class Graph* (SCG) has been proposed [10]. The State Class Graph (SCG) is an abstracted reachability graph of TPN. In order to compute SCG, we recall the algorithms in [33], in which the reduced SCG (a compact representation of the SCG for model checking) is computed based on the SCG.

Because time in TPN is continuous, in general the reachability graph of a TPN is infinite. In order to cope with this problem, the states in the reachability graph of a TPN are partitioned into regions called state classes. A *state class* is a pair $\alpha = \langle \mathbf{m}, F \rangle$, where $\mathbf{m}$ is a reachable marking and F is a conjunction of inequalities representing the (time) firing domains, i.e., the possible firing delays of transitions. If t_j is an enabled transition at $\mathbf{m}$ and it can be fired after l time units and must be fired before u time units, then there exists an inequality of the form $l \leqslant x_j \leqslant u$ in F, where x_j is the time delay in which t_j can be fired at $\mathbf{m}$.

The state class $\alpha = \langle \mathbf{m}, F \rangle$ is a partition in the reachability set of the TPN. It represnets the states that:

- the markings of the states are $\mathbf{m}$,
- the transitions enabled at $\mathbf{m}$ can be fired in the firing domain described by F.

In this way, the infinite reachability set is partitioned into finite number of subsets/partitions.

For the computation of the SCG, the following two functions are required (for details, see [33]):

1. isFireable(α, t_j) (see Algorithm 2.1) returns true if t_j can be fired at α. In the algorithm, the set of enabled transitions at $\mathbf{m}$ is denoted by $\mathrm{En}(\mathbf{m}) = \{t_j | \mathbf{m} \geqslant \mathbf{Pre}[\cdot, t_j]\}$.
2. succ(α, t_j) (see Algorithm 2.2) computes the successor of α by firing t_j.

Algorithm 2.1 [33] isFireable$(\alpha = \langle \mathbf{m}, F \rangle, t_j)$

```
1: **if** $t_j \notin \mathrm{En}(\mathbf{m})$ **then**
2:   **return** false
3: **end if**
4: let $F' := F \wedge (\bigwedge_{t_k \in \mathrm{En}(\mathbf{m}) \setminus \{t_j\}} x_j \leqslant x_k)$
5: **if** $F'$ is consistent **then**                ▷ *exists a solution of* $F'$
6:   **return** true
7: **else**
8:   **return** false
9: **end if**
```

Algorithm 2.2 [33] succ$(\alpha = \langle \mathbf{m}, F \rangle, t_j)$

```
1: $\mathbf{m}' := \mathbf{m} - \mathbf{Pre}[\cdot, t_j] + \mathbf{Post}[\cdot, t_j]$
2: $F' := F \wedge (\bigwedge_{t_k \in \mathrm{En}(\mathbf{m}) \setminus \{t_j\}} x_j \leqslant x_k)$
3: replace in $F'$ each variable $x_k \neq x_j$ by $x_k + x_j$
4: eliminate by substitution in $F'$, $x_j$ and all variables associated with
   transitions conflicting with $t_j$ at $\mathbf{m}$
5: **for each** newly enabled transition $t_k$ at $\mathbf{m}'$ **do**
6:   add $x_k \in I(t_k)$ to $F'$
7: **end for**
8: **return** $\alpha' := \langle \mathbf{m}', F' \rangle$
```

Definition 2.27. *[33] A* state class graph *is a triple* $\mathrm{SCG} = \langle \Omega, \twoheadrightarrow, \alpha_0 \rangle$, *where:*

1. $\alpha_0 = \langle m_0, F_0 \rangle$ *is the initial state class;*
2. $\twoheadrightarrow = \{\langle \alpha, t, \alpha' \rangle |$ *isFireable*$(\alpha, t) =$ *true*$, \alpha' =$ *succ*$(\alpha, t)\}$ *is the set of edges;*
3. $\Omega = \{\alpha | \alpha_0 \twoheadrightarrow^* \alpha\}$ *is the set of nodes (reachable state classes), where* $\twoheadrightarrow^*$ *is the reflexive and transitive closure of* $\twoheadrightarrow$.

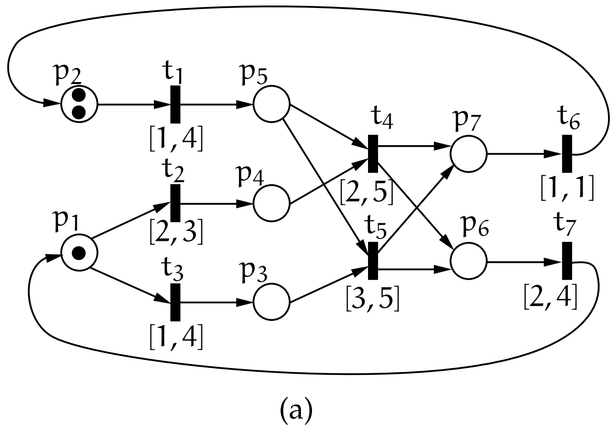
Algorithm 2.3 computes the state class graph. This algorithm implements two additional steps (steps 12 and 13) comparing with the original version in [33], because, without them, some edges could be omitted. The SCG is returned as a tuple $\langle \Omega, \twoheadrightarrow, \alpha_0 \rangle$, where Ω is the set of nodes (reachable state classes), $\twoheadrightarrow$ is the set of edges in SCG, and α_0 is the initial node (initial state class). An edge $\langle \alpha, t, \alpha' \rangle$ belongs to $\twoheadrightarrow$, if t is fireable at α and α' is obtained by firing t at α. The set W keeps state classes that are not explored yet. For a newly computed state class α', if it has not been explored, it is added to W (step 10) and the edge $\langle \alpha, t, \alpha' \rangle$ is inserted into $\twoheadrightarrow$. Considering the edge $\langle \alpha, t, \alpha' \rangle$, if α' exists in Ω, but the edge does not exist in $\twoheadrightarrow$, then the edge will be added into $\twoheadrightarrow$, while α' is not inserted in Ω or W (steps 12 and 13).

Algorithm 2.3 $\langle \Omega, \twoheadrightarrow, \alpha_0 \rangle = \text{SCG}(\mathcal{N}, m_0)$

```
1:  $\alpha_0 := \langle m_0, F_0 \rangle$
2:  $\Omega := \{\alpha_0\}$
3:  $\twoheadrightarrow := \emptyset$
4:  $W := \{\alpha_0\}$
5:  while $W \neq \emptyset$ do
6:    get and remove $\alpha := \langle m, F \rangle$ from $W$
7:    for each $t \in En(m)$ s.t. isFireable$(\alpha, t)$ do
8:      $\alpha' := succ(\alpha, t)$
9:      if $\alpha' \notin \Omega$ then
10:       add $\alpha'$ to $\Omega$ and to $W$
11:       add $\langle \alpha, t, \alpha' \rangle$ to $\twoheadrightarrow$
12:     else if $\langle \alpha, t, \alpha' \rangle \notin \twoheadrightarrow$ then
13:       add $\langle \alpha, t, \alpha' \rangle$ to $\twoheadrightarrow$
14:     end if
15:   end for
16: end while
17: return $\langle \Omega, \twoheadrightarrow, \alpha_0 \rangle$
```

Example 2.28. *Let us consider the TPN in Figure 2.9(a) with the initial marking* $m_0 = p_1 + 2p_2$[†]*. The initial state is* α_0 *whose marking is* m_0 *and its firing domain is* $F_0 = (1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_2 \leqslant 3) \wedge (1 \leqslant x_3 \leqslant 4)$. *Three transitions are enabled at* α_0 *and let us assume, for example, that* t_1 *will be fired first. In order to check if* t_1 *is fireable or not before all other enabled transitions, Algorithm 2.1 adds the following constraints to* F*:* $x_1 \leqslant x_2$ *and* $x_1 \leqslant x_3$ *obtaining* F'.

[†] We use $p_1 + 2p_2$ to represent the marking where there are one token in p_1 and two tokens in p_2.



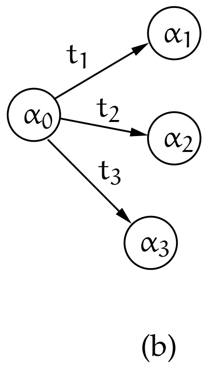


Figure 2.9: (a) A TPN where ε_4 is a fault transition [23]. (b) A part of the SCG starting from the inital class α_0, whose marking is $\mathbf{m}_0 = p_1 + 2p_2$. Details of nodes are shown in Table 2.1.

Table 2.1: State classes in Figure 2.9(b)

state class	marking	firing domain
α_0	$p_1 + 2p_2$	$(1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_2 \leqslant 3) \wedge (1 \leqslant x_3 \leqslant 4)$
α_1	$p_1 + p_2 + p_5$	$(1 \leqslant x_1 \leqslant 4) \wedge (0 \leqslant x_2 \leqslant 2) \wedge (0 \leqslant x_3 \leqslant 3) \wedge (x_3 - x_2 \leqslant 2)$
α_2	$2p_2 + p_4$	$0 \leqslant x_1 \leqslant 2$
α_3	$2p_2 + p_3$	$0 \leqslant x_1 \leqslant 3$

The successor of α_0 reached by firing t_1 is α_1, which is computed by using Algorithm 2.2. The marking of α_1 is obtained with the state equation of untimed PN. In order to compute the firing domain of α_1, the following steps are preformed:

1. *Let* $F_0 = (1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_2 \leqslant 3) \wedge (1 \leqslant x_3 \leqslant 4)$ *(the firing domain of* α_0*).*

2. *Add two constraints* $(x_1 \leqslant x_2) \wedge (x_1 \leqslant x_3)$ *(where the variable* x_i *represents time from the enabling of* t_i *or* ε_i *to its firing) to* F_0 *in order to represent that* t_1 *is fired not later than the other enabled transitions.*

3. *For the transition* ε_2 *(*ε_3*) whose local clock is* x_2 *(*x_3*), the residual time is* $x_2 - x_1$ *(*$x_3 - x_1$*) (because after* x_1 *time units* t_1 *is fired). Replace in the previous set of inequalities* x_2 *and* x_3 *with* $x_2 + x_1$ *and* $x_3 + x_1$*, respectively. The resulted firing domain is* $F_b = (1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_2 + x_1 \leqslant 3) \wedge (1 \leqslant x_3 + x_1 \leqslant 4) \wedge (0 \leqslant x_2) \wedge (0 \leqslant x_3)$.

4. *Eliminate* x_1 *from* F_b *by using Fourier-Motzkin technique. The resulted firing domain is* $F_c = (0 \leqslant x_2 \leqslant 2) \wedge (0 \leqslant x_3 \leqslant 3) \wedge (x_3 - x_2 \leqslant 2)$.

5. *Add the inequalities corresponding to newly enabled transitions after the firing of* t_1 *into* F_c *(*t_1 *in this case). The inequality* $1 \leqslant x_1 \leqslant 4$ *is inserted into* F_c *and the firing domain of* α_1 *is obtained as* $F_1 = (1 \leqslant x_1 \leqslant 4) \wedge (0 \leqslant x_2 \leqslant 2) \wedge (0 \leqslant x_3 \leqslant 3) \wedge (x_3 - x_2 \leqslant 2)$.

2.5.2 *Reduction rules of TPN*

Nevertheless, the number of state classes (nodes) in the SCG could be enormous. An approach to speed up the computation of the SCG of a TPN is to reduce the net, by removing some transitions, before constructing its SCG. Structural reduction rules are powerful tools in the analysis of untimed PN [9, 56]. Most of the reduction rules are designed to preserve properties of the PN system, such as boundedness and liveness, and they are applied to state estimation in [13]. However, most of these reduction rules are not applicable to TPN, due to the time intervals.

Consider the TPN in Figure 2.10(a) whose initial marking is $m_0 = 2p_1$. The reduction rule of untimed net is applied and the net in Figure 2.10(b) is obtained by merging t_1 and t_2. In the reduced net, the transition t'_{12} replaces t_1 and t_2 and its time interval is $[x, y]$. We will illustrate that in this example there is no feasible solution for x and y. In the original TPN, the global time interval at which t_3 is fired for the first time is $I_1 = [3, 7]$ and in the reduced TPN the global time interval is $I'_1 = [x + 1, y + 1]$. To have equivalent behaviors, $x = 2$ and $y = 6$. The time intervals at which t_3 can be fired for the second time in the original TPN is $I_2 = [4, 10]$ while in the reduced TPN is $I'_2 = [2x + 1, 2y + 1]$. In this case, $x = 1.5$ and $y = 4.5$. Obviously, this is not consistent with the first values of x and y. Therefore, this reduction rule is not preserving the time behavior of the original net.

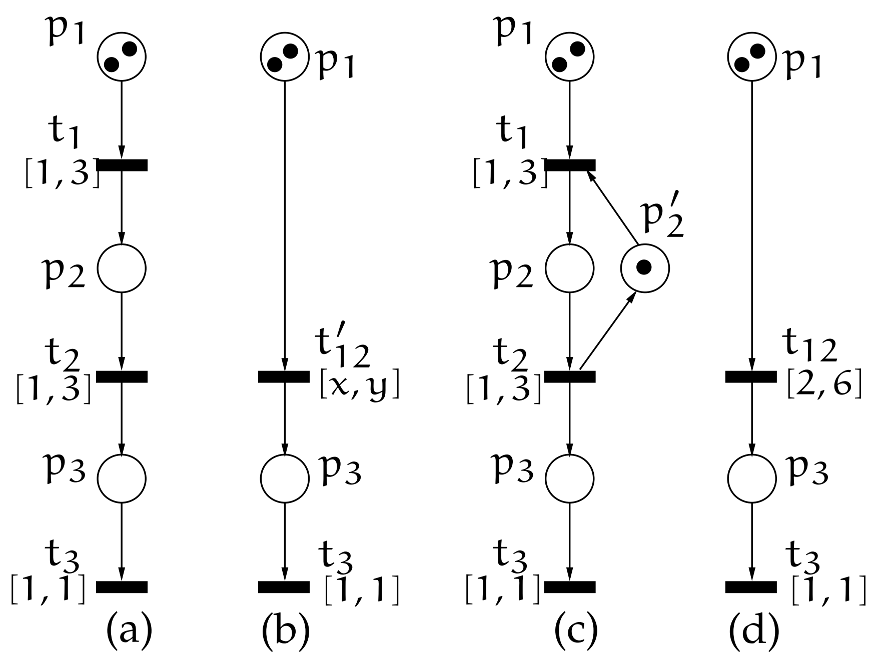


Figure 2.10: A reduction rule that cannot be applied if the TPN is not safe (from PN in (a) to the one in (b)). It can be applied when p_2 is safe (shown in (c) and (d)).

This happens because in the original TPN t_1 and t_2 can be fired concurrently, while in the reduced TPN t'_{12} cannot have different time intervals for the intermediate marking. In TPN, due to the addition of time intervals, structural reduction rules can only be applied under certain conditions. For example, some reduction rules have been proposed in [60] for parts of TPN in which the upper bound of the number of tokens in each place is 1. Let us consider again the PN in Figure 2.10(a), but assume now that the capacity of p_2 is 1. The equivalent PN is shown in Figure 2.10(c), where the new place p'_2 ensures that the maximum number of tokens in p_2 is 1 at every reachable marking. In that subnet system, the previous reduction rule can merge t_1 and t_2 and remove the places p_2 and p'_2. The time interval of t_{12} is $[2,6]$ computed by summing the lower and upper bounds of the time intervals of t_1 and t_2, respectively (see in Figure 2.10(d)). This is a rule considered in [13] for safe TPN[‡].

2.6 PN IN RESOURCE ALLOCATION SYSTEMS

A *resource allocation system* represents the allocation and deallocation of resources in a physical system. It is widely used in deadlock analysis, collision avoidance and system performance investigation on resource critic systems.

‡ In a safe TPN, the maximum number of tokens in every place is 1 at every reachable marking.

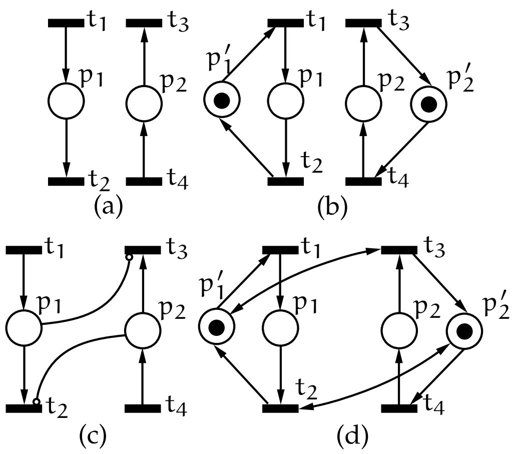


Figure 2.11: Assuming the net system in (a) is binary, (a) and (b) have the same behavior. The inhibitor arcs in (c) are transformed as p'_1 and p'_2 with their input and output arcs shown in the net in (d).

2.6.1 *Alternative notations of PN*

In resource allocation systems and deadlock analysis, two subclasses of PN, S^3PR and S^4PR, are used.

Example 2.29. *Let us consider the PN in Figure 2.11(a) with the assumption that the markings of places are upper bounded by one. Their capacities can be represented by two additional places as in Figure 2.11(b). The inhibitor arcs (p_1, t_3) and (p_2, t_2) (Figure 2.11(c)) could be replaced by two* reading arcs[§] *as shown in Figure 2.11(d).*

2.6.2 *The Class of S^3PR*

In the sequel, we introduce S^3PR (System of Simple Sequential Processes with Resource)[24].

Definition 2.30. *Let $I_\mathcal{N}$ be a finite set of indices, an S^3PR is a Petri net $\mathcal{N} = \langle P, T, F\rangle$[¶] where:*

1. $P = P_S \cup P_R \cup P_0$ *is a partition such that:*
   a) $P_0 = \{p_0^1, \dots, p_0^k\}, k > 0$ *($p \in P_0$ is called an* idle *place);*
   b) $P_S = \bigcup_{i=1}^k P_S^i$*, where* $\forall i \neq j, P_S^i \cap P_S^j = \emptyset$ *($p \in P_S$ is called an* operation *or* process *place);*
   c) $P_R = \{r_1, \dots, r_n\}, n > 0$ *($r \in P_R$ is called a* resource *place);*
2. $T = \bigcup_{i=1}^k T^i$*, where* $\forall i \neq j, T^i \cap T^j = \emptyset$*;*
3. $\forall i \in I_\mathcal{N}$ *the subnet $\mathcal{N}^i$ generated by* $\{p_0^i\} \cup P_S^i \cup T^i$ *is a strongly connected state machine such that every circle contains* $\{p_0^i\}$ *($\mathcal{N}^i$ is called a* simple sequential process *(S^2P));*

§ A reading arc (p'_1, t_3) is a self-loop and it means two arcs (p'_1, t_3) and (t_3, p'_1). A reading arc can also be called a *bidirectional arc*.

¶ In order to be consistent to the widely used notation in S^3PR, we use F instead of **Pre** and **Post** in deadlock prevention on S^3PR.

4. *$\mathcal{N}$ is strongly connected;*

5. $\forall i \in I_{\mathcal{N}}, \forall p \in P_S^i$, ${}^{\bullet\bullet}p \cap P_R = p^{\bullet\bullet} \cap P_R$ *and* $|{}^{\bullet\bullet}p \cap P_R| = 1$.

From 2.30, we can see that an S^3PR is a well-defined system meaning. The firing of a transition will allocate and/or release one resource. If a resource is allocated in $p \in P_S$ by firing $t \in {}^{\bullet}p$, it will be released by the firing of $t' \in p^{\bullet}$. The notation $I_{\mathcal{N}}$ is the set of indices of processes in an S^3PR $\mathcal{N}$.

Definition 2.31. *Let $\mathcal{N} = \langle P_S \cup P_R \cup P_0, T, F\rangle$ be an S^3PR. An initial marking $\mathbf{m}_0$ is called* acceptable *if:*

1. $\mathbf{m}_0[p] \geqslant 1, \forall p \in P_R$;

2. $\mathbf{m}_0[p] = 0, \forall p \in P_S$;

3. $\mathbf{m}_0[p] \geqslant 1, \forall p \in P_0$.

The couple $\langle \mathcal{N}, \mathbf{m}_0\rangle$ is called an (acceptably) marked S^3PR if:

1. at least one token is assigned to each idle place and resource place;

2. all process places are empty.

In the S^3PR in Figure 2.12:

- $P_0 = \{p_0^1, p_0^2\}$,
- $P_S = \{p_i, i = 1, \ldots, 8\}$,
- $P_r = \{r_i, i = 1, \ldots, 5\}$.

Definition 2.32. *Let $I_{\mathcal{N}}$ be a finite set of indices. An S^4PR net is a connected self-loop free PN $\mathcal{N} = \langle P, T, F\rangle$, where:*

1. $P = P_0 \cup P_S \cup P_R$ *is a partition such that:*
   a) $P_s = \bigcup_{i \in I_{\mathcal{N}}} P_S^i$, $P_S^i \neq \emptyset$ *and* $\forall i \neq j, P_S^i \cap P_S^j = \emptyset$;
   b) $P_0 = \bigcup_{i \in I_{\mathcal{N}}} \{p_0^i\}$;
   c) $P_R = \{r_1, \ldots, r_n\}, n > 0$;

2. $T = \bigcup_{i \in I_{\mathcal{N}}} T_i, T_i \neq \emptyset, \forall i \neq j, T_i \cap T_j = \emptyset$;

3. $\forall i \in I_{\mathcal{N}}$, *the subnet $\mathcal{N}_i$ generated by $P_S^i \cup \{p_0^i\}$ and T_i is a strongly connected state machine, such that every cycle contains p_0^i;*

4. *for each $r \in P_R$, there exists a minimal P-semiflow, $\mathbf{y}_r$ such that $\{r\} = \|\mathbf{y}_r\| \cap P_R$, $\mathbf{y}_r[r] = 1$, $P_0 \cap \|\mathbf{y}_r\| = \emptyset$ and $P_S \cap \|\mathbf{y}_r\| \neq \emptyset$.*

The constraint in S^3PR that each process can only allocate one resource is relaxed in S^4PR. As shown in Figure 2.13, the process place p_3 can allocate resources from r_3 and r_6. Therefore, S^4PR can represent a wider range of systems than S^3PR.

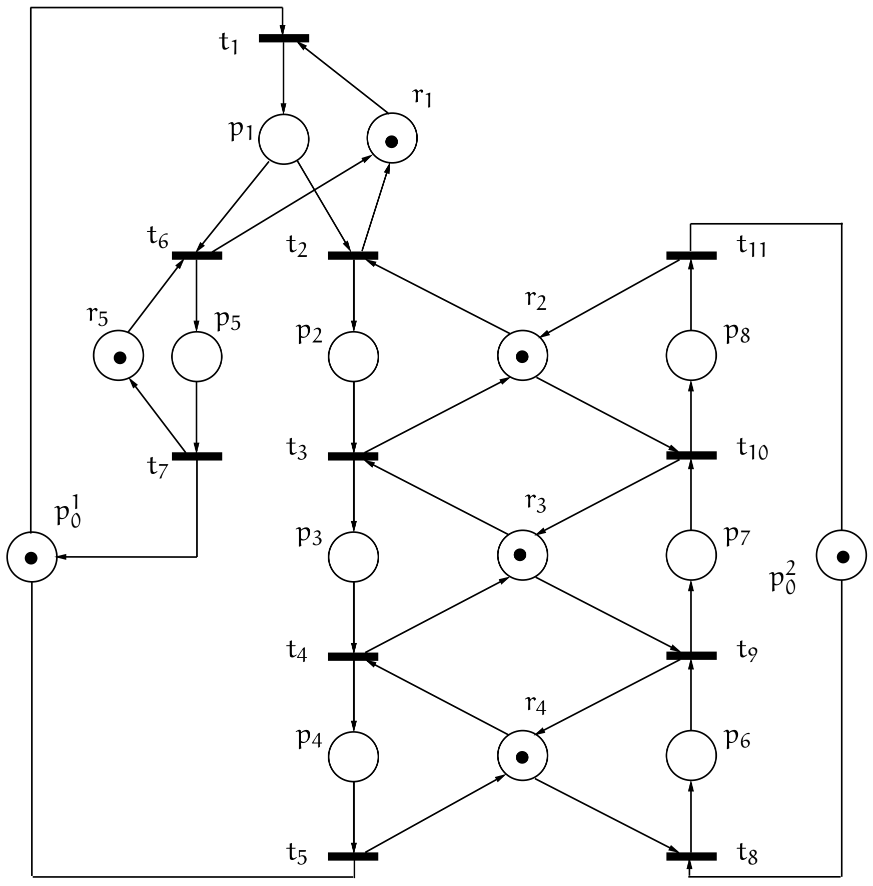


Figure 2.12: An S^3PR

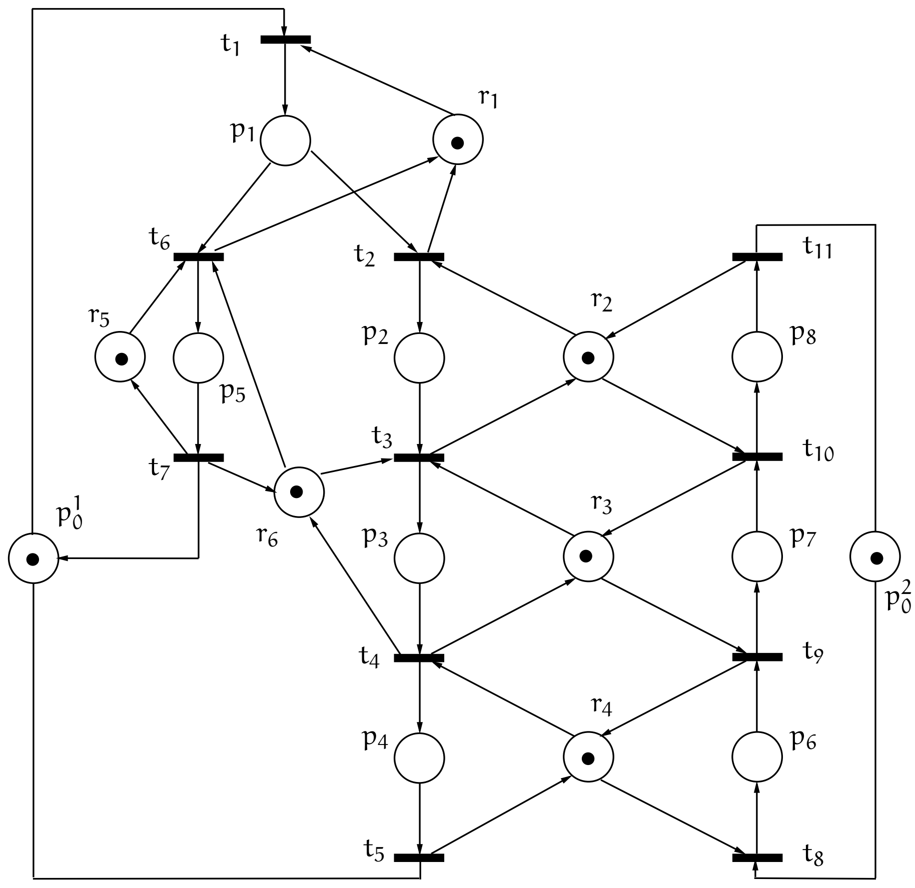


Figure 2.13: An S^4PR

Part II

FAULT DIAGNOSIS ON TIME PETRI NETS

3 INTRODUCTION TO FAULT DIAGNOSIS ON PETRI NET

3.1 INTRODUCTION

In real world, most systems are real time systems, e.g., power plant control system, aircraft navigation system, and so on. Inside a real time system, each components interact with others continuously, while the whole system must react to the environment (inputs) constantly. The correctness of the outputs and timeliness [63] ensure the logical correctness of real time systems. *Time Petri nets* (TPN) are proposed [46] by associating time interval delays with *transitions*, and they have the ability to associate time with events. Thus, TPN is one widely used tool for real time system specification and verification. However, the *state space* of a TPN system is infinite. In order to represent the state space in a finite way, some approaches are proposed. One of them is *State Class Graph* (SCG) [10], which partitions the state space of a TPN system into regions (called *state classes*). The states belonging to the same state class satisfy two conditions. First, they have the same marking, and second, their firing domains of transitions (defining when each transitions can be fired) can be represented using linear inequalities.

Fault diagnosis on TPN considers various problem configurations on which parts of the TPN can be observed and how to define faults:

- The observable elements of TPN may contain observable places, transitions or both. The number of tokens in an observable place can be measured by an external observer. It may be observed continuously, where the latter means marking of the place is observable at some moments, but not always. The firing of an observable transition can be detected by a sensor. The information of which transitions are fired and when they are fired is used in diagnosis.

- Types of faults contains fault markings or transitions. In systems with fault markings, a fault occurs when a fault marking is reached or the marking of the system satisfies a pattern, e.g., an inequality. The firing of a transitions can be considered as a fault and the transition is a *fault transition*. In TPN, some fault transition describe faults with time, while others do not. For example, the firing of a fault transition in the latter type means a fault event occurs irrespective of when it is fired. If a fault is associated with a given time interval, then the firing of the corresponding transition outside the time interval does not represent the occurrence of the fault.

We consider also the case in which some transitions are observable, while markings are not. In the representation of faults, we assume some (unobservable) transitions are associated with faults (without time) and no marking is fault marking. Based on the problem configuration, we propose the definition of *diagnoser* in Section 3.3.2.

In order to deal with the problem, we propose FDG. In brief, FDG contains necessary information for diagnosis on TPN such that at each time, the diagnosis algorithm can take information from FDG without using a TPN model. In detail, an FDG is a projection of SCG, and in the FDG we remove nodes and edges if they are not used in diagnosis. Therefore, FDG has less nodes than the corresponding SCG. Moreover, some labels are associated with both nodes and edges in FDG to improve the diagnosis procedure, in particular, the time complexity. FDG is constructed incrementally, meaning that a part of an FDG will not be constructed until it is used. We also propose some reduction rules on FDG so that it can be more compact without loss of information for diagnosis. Then, FDG is applied to both centralized and decentralized problems. In centralized diagnosis, a single *diagnoser* manages all tasks of diagnosis, including observing firings of transitions, constructing FDG and computing diagnosis states. In decentralized diagnosis, several local diagnosers collaborate through a coordinator. Each local diagnoser observes its local observable transitions and constructs a local FDG. The coordinator update the (global) diagnosis states using messages sent by local diagnosers.

In Chapter 4, besides the details of algorithms on construction of FDG, we also illustrate the details of the algorithm computing the *firing domain of a given firing sequence*. A firing domain describes the earliest and latest time in which all transitions in the firing sequence can be fired sequentially. It checks whether a firing sequence is consistent with an observed transition and the observed time or not. Using the firing domain, false diagnosis states or uncertain diagnosis states are reduced. The case study in Chapter 7 shows that, by incrementally constructing the FDG and using labels in the FDG, the diagnosis on FDG can be as fast as an untimed approach, which does not use timing information. The time and space complexities are discusses in Chapter 5, in which we prove that for a finite observation the number of consistent states is polynomial according to the length of the observation.

The problem to be considered in the first part of the thesis relate to fault detection, and they are presented in Chapter 4, Chapter 5, Chapter 6 and Chapter 7:

- In order to deal with the problem, we propose *Fault Diagnosis Graph* (FDG). The definition and detailed algorithms of constructing FDG are proposed in Chapter 4. In order to construct FDG, *firing domain of a firing sequence* must be computed. We give an algorithm to complete this task. After that, we propose an algorithm to construct FDG incrementally according to observations. We also define some rules to reduce FDG to save storage space and improve time complexity.

- In Chapter 5, FDG is applied to centralized diagnosis and detailed algorithms are given with an analysis on computational complexity. We first give general steps on centralized diagnosis and explain how it works with an example. Second, the centralized diagnosis algorithm using FDG is introduced. Last, we show results on computational complexity analysis of centralized diagnosis using FDG.

- Chapter 6 discusses decentralized diagnosis using FDG. The decentralized system architecture consists of several local systems and a coordinator. We propose how each local diagnoser (associated with each local systems) collaborate with others through the coordinator.

- A case study containing two systems are proposed in Chapter 7. The FDG based approach is compared with an approach where timing information is not considered. As expected, our approach can reduce uncertainty to states of fault diagnosis.

3.2 LITERATURE REVIEW

An interpreted diagnoser is proposed in [4] for untimed PN using *Integer Linear programming Problem* (ILP) with the assumption that all unobservable events (transitions) can be detected. It means that the observed information is sufficient to detect whether a fault transition has been fired or not. In order to manage the diagnosis, the so called *g-marking* (short for *generalized marking*) approach is developed. When an observable transition is observed and it was not enabled in the net system, a firing sequence containing only unobservable transitions must have been fired to enable the observed transition. By using the PN state equation, a marking having negative elements is obtained from the firing of the observed transition and this marking is called a g-marking. The g-marking corresponding to the firing of an observable transition is not unique in general, because the firings of various unobservable firing sequence may be able to explain the enabling of the observable transition. Hence, a set of possible g-markings can be obtained according to an observation. If the set of g-markings is a singleton, then the g-marking is used in diagnosis; otherwise, more observations are necessary.

For the diagnosis on untimed PN, *basis marking* is introduced in [15] with the assumption that the unobservable subnet is acyclic. When an observable transitions is fired, a basis marking is reached by firing only a minimal sequence of unobservable transition followed by the observed transition. The *Basis Reachability Graph* (BRG) is constructed, whose nodes are basis markings, and used to summary information required for diagnosis. After constructing the BRG, when an observable transition is fired, the diagnosis procedure can look at the BRG for paths to update the diagnosis states.

An on-line fault detection strategy is proposed [23] avoiding the redefinition and redesign of dianogser when the structure of the untimed PN varies. They assume that the PN structure and the initial marking are known. The set of transitions is partitioned into observable and unobservable subsets, while fault transitions are unobservable ones. A set of constraints is maintained during the diagnosis process. When an observable transition is fired, some inequalities are added into the set of constraints. The set of constraints is used in an ILP. The firing of fault transitions is potentially detected by maximizing the firing counts of fault transitions. The proposed algorithms do not need off-line calculation. However, the number of constraints may become colossal. In order to reduce the computational complexity, a sufficient condition is given: if the PN system is a bounded state machine, the continuous relaxation of the ILP can be applied.

In [25], this approach is extended to the diagnosis of timed systems modeled by TPN. The constraints of the ILP problems characterize consistent firing sequences according to the observation and the time. In order to integrate the representation of time information into ILP problems, they compute the time bounds of a firing sequence by summing the lower and upper bounds of time delays associated with transitions in the firing sequence. The approach is complete, which means that if a new observation comes, the computation of diagnosis states is done starting from the initial time of the system. Our approach, which is incremental, is enumeration-based and ILP problems are not involved in diagnosis. When a new observation comes, we compute (if necessary) only the part corresponding with the observed transition. The time bounds of a firing sequence are represented by linear constraints and computed by solving LPPs.

In timed PN, a (non-interval) time delay is associated with each transition representing the time that must elapse from the enabling until the firing of the transition. Basile et al. [3] present *Timed Explanation Tree* (TET) for fault diagnosis on timed PN. If an unobservable firing sequence containing an fault transition is enabled, then a timer is associated with the fault transition. If no firing sequence containing the fault transition is remaining being enabled after the firing of an observable transition, then the timer is reset. A candidate g-marking can be discarded if the timer does not match the possible firing time of the corresponding observable transition.

In [5], it is assumed that observations contain only the firings of some (observable) transitions, which are associated with sensors and the sensors generate output when the observable transitions are fired. The firing of other transitions are not observable. The same sensor can be associated with more than one transition so that the same output may mean the firings of various transitions that share the same sensor. These transitions are *indistinguishable*. The SCG is extended to *Modified State Class Graph* (MSCG) by attaching labels to edges. The labels contains timing variables of transitions and constraints. The number of nodes in an MSCG is larger than the one of the corresponding SCG, because of the modifications in the MSCG. In order to determine which states are consistent with a given observation with a given time, a procedure is proposed that explores the MSCG and solve Linear Programming Problems (LPP). In order to provide a trade-off between computation time and storage space, MSCG is constructed off-line while LPP is solved on-line. In our approach, FDG has less nodes than the corresponding SCG and it is constructed incrementally meaning the SCG is not necessary to be computed in the beginning.

For on-line distributed asynchronous diagnosis, a *net unfolding* approach is proposed in [8]. A so called *true concurrency* is considered where no global state or global time is available. As a result, partial order model of time is used. Using net unfolding, the *state explosion* is under control by paying the cost on on-line diagnosis performance, which is typical in diagnosis of complex asynchronous systems in a true concurrency environment. The disadvantage of net unfolding is that it constrains the approach to *safe* nets, whose places can have most one token each.

Genc and Lafortune [27] propose a distributed diagnosis approach for on-line fault detection and isolation of modular dynamic systems represented by *place-bordered* PN. A place-bordered PN consists of submodels sharing places on their borders. Their solution contains two steps: first, diagnose fault occurrence in each module in an on-line manner, and second, use the diagnosis results in all module to recover the diagnosis states of the global system. The second step uses the pattern of the construction of the global system. Each diagnoser can use its local information about the corresponding module and the shared place. One diagnoser sends messages to another diagnoser if they share places with each other on their border. The messages contain only changes of markings in the shared places and they are sent when a change occurs. When a diagnoser receives a message, it updates its diagnosis states accordingly.

Distributed diagnosis in large scaled systems is considered in [35], in which a distributed protocol is defined for fault detection and isolation. The global system contains local processes and each local processes is modeled as a TPN system. The local processes interacts with each other using *guarded transitions*, which are associated with conditions and can be enabled only when the corresponding conditions are satisfied. A condition on a guarded transition is a predicate over the marking of some places in other processes. Two types of faults associated with transitions are considered, where the firing of some transitions means faults occur and for some other fault transitions, if they are fired in some given intervals, then faults occur. An agent (local diagnoser) is associated with each local process, while they exchange limited information between each other. It is proved that with the communication, the diagnosis performance is same as in the centralized case.

Cabasino et al., address decentralized diagnosis using PN in [14], where the global system consists of *sites*. Each site has the knowledge of the structure and the initial marking of the global system, but can only observe the firing of a subset of observable transitions (via different masks). A site performs local diagnosis based on its local knowledge and the approach used in local diagnosis is the one proposed in [15]. After that, following a protocol, the local observations and some other information are sent to a coordinator, which is in charge of calculating global diagnosis states.

3.3 PROBLEM STATEMENT

Fault diagnosis has been studied for many years. In DES, various diagnosis approaches are developed in automata, PN, etc. Because most systems are real time systems, we consider diagnosis on TPN (defined in Chapter 2). In our problem, a TPN system has observable and unobservable transitions, while faults are the firings of some unobservable transitions. Our target is to estimate the firings of fault transitions using observations, which contains the firings of observable transitions with the time when they are fired. The diagnoses without using timing information lead to many uncertain states.

Definition 3.1. *The* unobservable SCG *of a TPN system* $\langle \mathcal{N}, I, \mathbf{m}_0 \rangle$ *is the SCG of the TPN system* $\langle \mathcal{N}_u, I_u \mathbf{m}_0 \rangle$, *i.e., the SCG of the TPN obtained by firing only unobservable transitions.* ■

Definition 3.2. *An* observed word *is a sequence of ordered pairs* $w = \langle t_1, \tau_1 \rangle \cdots \langle t_k, \tau_k \rangle \in (T_o \times \mathbb{Q}_0)^*$, *in which* t_1 *is the first observed transition and it is observed at* τ_1, *while* t_k *is the last observed transition at* τ_k. *Let* $\langle \mathcal{N}, I, \mathbf{m}_0 \rangle$ *be a TPN system and* $w = \langle t_1, \tau_1 \rangle \ldots \langle t_k, \tau_k \rangle$ *be an observed word. We define the* set of firing sequences consistent with w *by* $\mathcal{L}_\lambda(w) = \{\sigma | \mathbf{m}_0[\sigma\rangle, w = \lambda(\sigma) = t_1 \ldots t_k,$ *such that* t_i *is fireable at* $\tau_i, i = 1, \ldots, k\}$. ■

3.3.1 *Fault classes*

We model faults using unobservable transitions. It means that some unobservable transitions and all observable transitions represent normal behaviors. To denote the fault and regular behaviors, we use T_f to represent the set of fault transitions and T_{reg} for all regular unobservable transitions. That is $T_u = T_f \cup T_{reg}$ and $T_f \cap T_{reg} = \emptyset$. Because there are several types of faults, the set of fault transitions is further partitioned according to these types such that

$$T_f = T_f^1 \cup T_f^2 \cup \cdots \cup T_f^r.$$

All transitions in the same subset represent one type of fault (called a *fault class*). Hence, we concentrate on the firing of transitions in each fault class rather than the firing of each individual fault transition.

3.3.2 *Diagnoser*

Now, we formally define the diagnoser and diagnosis states. Diagnosis states describe whether each faulty event occurred or not, while fault transitions are grouped into fault classes. The diagnoser is used to characterize diagnosis states corresponding to observations.

There are three diagnosis states namely *normal*, *faulty* and *uncertain*. A diagnoser maps each observed word to one of these diagnosis states.

Definition 3.3. *A* diagnoser *is a function* $\Delta : (T_o \times \mathbb{Q}_0)^* \times \{T_f^1, T_f^2, \ldots, T_f^r\} \to \{N, F, U\}$, *where* $(T_o \times \mathbb{Q}_o)^*$ *is the set of observed words* $w =$

$\langle t_1, \tau_1 \rangle \ldots \langle t_j, \tau_j \rangle$ *and N, F and U represent* Normal, Faulty *and* Uncertain *states, respectively. The diagnoser associates with each observed word* w *and with each fault class* $T_f^i, i = 1, \ldots, r$, *a diagnosis state.*

- $\Delta(w, T_f^i) = N$ *if for all* $\sigma \in \mathcal{L}_\lambda(w)$ *and* $\forall t_f \in T_f^i$ *it holds* $t_f \notin \sigma$.

  In this case, none *of the firing sequences consistent with the observation contains a fault transition of class* i, *i.e., the* i*-th fault cannot have occurred.*

- $\Delta(w, T_f^i) = U$ *if*

  1. $\exists \sigma \in \mathcal{L}_\lambda(w)$ *and* $t_f \in T_f^i$ *such that* $t_f \in \sigma$, *but*
  2. $\exists \sigma' \in \mathcal{L}_\lambda(w)$ *such that* $t_f \notin \sigma', \forall t_f \in T_f^i$.

  In this case, a fault transition of class i *may have occurred or not, i.e., it is uncertain.*

- $\Delta(w, T_f^i) = F$ *if* $\forall \sigma \in \mathcal{L}_\lambda(w)$ *and* $\exists t_f \in T_f^i$ *it holds* $t_f \in \sigma$.

  In such a case, because every fireable sequence consistent with the observation contains at least one fault transition of class i, *it is certain that the* i*-th fault have occurred.*

Let us use $\Delta(w)$ *to denote the set of all diagnosis states corresponding to the observation* w, *i.e.,*

$$\Delta(w) = \{\Delta(w, T_f^i), i = 1, \ldots, r\}.$$

■

Definition 3.4. *Considering an observation word* $w = \langle t_{o1}, \tau_{o1} \rangle \langle t_{o2}, \tau_{o2} \rangle \ldots \langle t_{on}, \tau_{on} \rangle$, *a* consistent state class α *is a state class reached by firing a firing sequence* $\sigma = \sigma_{u1} t_{o1} \sigma_{u2} t_{o2} \ldots \sigma_{un} t_{on}, \sigma_{ui} \in T_u^*$ *with* $i = 1, \ldots, n, w = \lambda(\sigma) = t_{o1} \ldots t_{on}$. ■

4 FAULT DIAGNOSIS GRAPH AND ALGORITHMS

Fault diagnosis graph is proposed for diagnosis on TPN in this chapter with algorithms on the construction of fault diagnosis graph in an incrementally way. Two algorithms are introduced: one computes the firing domain of a given firing sequence, and the other one constructs fault diagnosis graph incrementally. In diagnosis, firing domains characterize whether firing sequences are consistent with observations or not. We propose a linear programming problem (LPP) based algorithm to compute firing domains. The construction of fault diagnosis graph starts from computing an state class graph. After that, the nodes and edges in the state class graph are removed if they will not be used in diagnosis. Hence, the number of nodes of a fault diagnosis graph is smaller than the one of the corresponding state class graph. Fault diagnosis graph will be used in centralized and decentralized in Chapter 5 and Chapter 6, respectively.

4.1 INTRODUCTION

In this chapter, we propose Fault Diagnosis Graph (FDG) and detailed algorithms to construct FDG and illustrate it with an example. Next two chapters concentrate on applications of FDG to centralized (Chapter 5) and decentralized diagnoses (Chapter 6).

In order to construct FDG, firing domain of a given firing sequence is necessary (Section 4.2). It is used to check whether an unobservable firing is consistent with an observation or not and will be a part of labels of edges in FDG. We construct a Linear Programming Problem (LPP), whose constraints represent the firings of transition in a firing sequence, to compute the firing domain of the firing sequence. The LPP represents the tight firing domain of a firing sequence. Solving the LPP is a simple problem with widely used solvers. However, the number of constraints of the LPP grows with the length of the firing sequence. Therefore, it is appliable to finite firing sequences.

Having firing domains, we can construct FDG from an SCG incrementally. Nodes and edges are removed from the SCG, if they will not be used in diagnosis forever. In order to speed up diagnosis on FDG, we associate labels to edges, which contains firing domains. An example is used to illustrate the construction of FDG.

4.2 FIRING DOMAIN OF A GIVEN FIRING SEQUENCE

In order to reduce the SCG keeping only information that is relevant for fault diagnosis, the *firing domain of a given firing sequence*[∗] (or simply the *firing domain*) is computed as the time interval $[g_l, g_u]$ describing the earliest and latest instants to fire all transitions in the firing sequence according to the order of transitions in the firing sequence. It will be used in diagnosis to verify whether a state class is consistent with an observation or not. Moreover, the firing domain of $\sigma_u t$ is the time interval at which t can be observed.

The *firing domain of a firing sequence* is a time interval defining the earliest and latest instants in which all transitions belonging to the firing sequence can be fired sequentially. This firing domain is used to detect if a firing sequence is consistent with an observation or not, i.e., whether the observed transition can be fired at the observed time or not. Also, this information is shown on the edges in the FDG.

In order to compute the firing domain $[g_l, g_u]$ of a sequence σ, we define $k = |\sigma|$ variables to represent the time when each transition is fired. For each transition t in σ, the following steps are applied:

1. Find when the transition becomes enabled.
2. Construct the linear inequalities of the firing duration of t representing when it is enabled and when it is fired.
3. Find the transitions that are disabled by the firing of t and add constraints to represent that they are disabled before reaching

∗ The firing domain verifies a given firing sequence is consistent with the observation or not.

their upper bounds of time intervals (otherwise, they must be fired).

4. For each transition enabled before the firing of the last transition of σ, we add a constraint to describe that the last transition must be fired before them.

Finally, minimize and maximize the last variable, which represents the time moment when all transitions in σ have been fired, subject to the linear constraints, to calculate g_l and g_u.

Algorithm 4.1 $[g_l, g_u] := \text{domain}(\sigma, \mathcal{G}_{scg}, \alpha_0)$

```
1:  $\bar{\sigma} := \sigma = t_1 t_2 \cdots t_k$
2:  get $\alpha_0 = \langle m_0, F_0 \rangle \xrightarrow{t_1} \alpha_1 = \langle m_1, F_1 \rangle \xrightarrow{t_2} \cdots \xrightarrow{t_k} \alpha_k = \langle m_k, F_k \rangle$ from $\mathcal{G}_{scg}$
3:  define k variables $y_1, \dots, y_k$
4:  $cons := 0 \leqslant y_1 \leqslant \cdots \leqslant y_k$
5:  **while** $|\bar{\sigma}| > 0$ **do**
6:    let $\sigma'$ s.t. $\bar{\sigma} = \sigma' t$
7:    find max j s.t. $\sigma' := \sigma'' \sigma'''$, $j = |\sigma''|$, t is enabled after $\sigma''$ and remains enabled during $\sigma'''$
8:    get $\alpha_j$ and $\alpha_s$ s.t. $\alpha_0 \xrightarrow{\sigma''} \alpha_j \xrightarrow{\sigma'''} \alpha_s$
9:    $[l, u] :=$ minimize and maximize $x_t$ s.t. $F_j$
                ▷ *$l$ and $u$ are the earliest and latest time at which $t$ can be fired*
10:   $cons := cons \wedge (l \leqslant y_{|\bar{\sigma}|} - y_j \leqslant u)$
11:   **for each** $t_q \in En(m_s) \setminus (En(m_s - \mathbf{Pre}[\cdot, t]) \cup t)$ **do**
                ▷ *the clock of $t_q$ is reset after the firing of $t$*
12:     find max q s.t. $\sigma' := \sigma_1 \sigma_2$, $q = |\sigma_1|$, $t_q$ is enabled after $\sigma_1$ and remains enabled during $\sigma_2$
13:     get $\alpha_q$ s.t. $\alpha_0 \xrightarrow{\sigma_1} \alpha_q$
14:     $u_q :=$ maximize $x_q$ subject to $F_q$
                ▷ *$u_q$ is the latest time at which $t_q$ must be fired*
15:     $cons := cons \wedge (y_{|\bar{\sigma}|} - y_q \leqslant u_q)$
16:   **end for**
17:   $\bar{\sigma} = \sigma'$
18: **end while**
19: **for each** $t \in En(m_{k-1}) \setminus \{t_k\}$ **do**
20:   max j s.t. $\sigma = \sigma'' \sigma'''$, $j = |\sigma''|$, t is enabled after $\sigma''$
21:   $u :=$ maximize $x_t$ s.t. $F_{k-1}$
22:   $cons := cons \wedge (y_k - y_j \leqslant u)$
                ▷ *$t_k$ must not be forced to fire before the latest firing time of $t$*
23: **end for**
24: $[g_l, g_u] :=$ minimize and maximize $y_k$ subject to $cons$
25: **return** $[g_l, g_u]$
```

Algorithm 4.1 computes the firing domain of a given firing sequence σ. It constructs linear constraints of the transitions in σ from the last transition to the first one. Before proposing the algorithm, we first define two variables x_i and y_i associated with the transition t_i. In the linear constraints, both x_i and y_i are time instants when t_i could be fired, but:

Table 4.1: State classes in Figure 4.1(b)

state class	marking	firing domain
α_0	$p_1 + 2p_2$	$(1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_2 \leqslant 3) \wedge (1 \leqslant x_3 \leqslant 4)$
α_1	$p_1 + p_2 + p_5$	$(1 \leqslant x_1 \leqslant 4) \wedge (0 \leqslant x_2 \leqslant 2) \wedge (0 \leqslant x_3 \leqslant 3) \wedge (x_3 - x_2 \leqslant 2)$
α_2	$2p_2 + p_4$	$0 \leqslant x_1 \leqslant 2$
α_3	$2p_2 + p_3$	$0 \leqslant x_1 \leqslant 3$
α_7	$p_2 + p_4 + p_5$	$(1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_4 \leqslant 5)$
α_8	$p_2 + p_3 + p_5$	$(1 \leqslant x_1 \leqslant 4) \wedge (3 \leqslant x_5 \leqslant 5)$

1. x_i is the *local time* counting the time units from the moment when t_i becomes enabled;

2. y_i is the *global time* counting the time units from the initial time of the whole system.

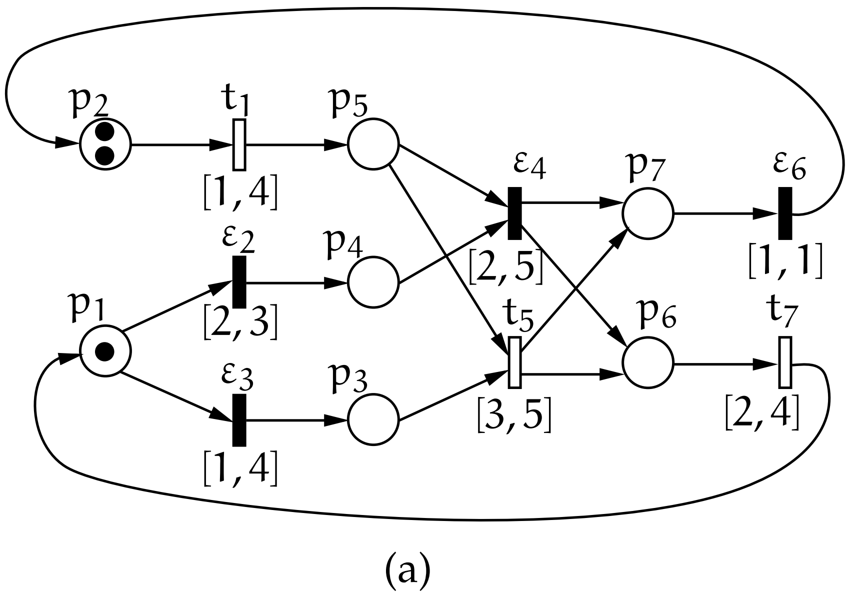
Assume the time interval of t_i is $[a, b]$ and, at time τ, t_i becomes enabled. The time at which t_i could be fired is $x_i \in [a, b]$ and $y_i \in [a + \tau, b + \tau]$. The variable x_i appears only in the firing domains associated with state classes, while the constraints obtained in Algorithm 4.1 contain only y_i.

The sequence of state classes corresponding to σ is found in $\mathcal{G}_{scg}$ (step 2). We associate a variable y_i with the i-th transition t_i in σ (step 3). Because the transitions in σ are sequentially fired, the time moment (y_i) should be in the same order (step 4). Every transition in σ is enabled either from the beginning or after a subsequence. Let the transition t be the last transition of σ and assume that t is enabled after the firing of the first j transitions of σ, i.e., t is enabled at y_j, and then the time between the enabling of t until it is fired is given by $y_{|\sigma|} - y_j$. Let α_j be the state class at which t starts to be enabled. Hence, the firing delay of t is constrained by the firing domain F_j of α_j (step 8). The time interval in which t can be fired is obtained in step 9.

Next, we consider transitions that are disabled due to the firing of t (steps 11 to 15). For each disabled transition, obtain the upper bound of the time domain and add an inequality describing that t must be fired earlier than the disabled transition (step 15).

If a transition $t \notin \sigma$ is enabled but not fired in σ, then the firings of all transitions in σ must be earlier than t (steps 19 to 23). Finally, the firing domain of the firing sequence σ is obtained by minimizing and maximizing the last variable y_k subject to $cons$ (step 24).

Example 4.1. *Let us consider again the TPN in Figure 4.1 and the firing sequence $\sigma = t_1\varepsilon_2\varepsilon_4$, which is enabled at $\alpha_0 = \langle \mathbf{m}_0, F_0 \rangle$ with $\mathbf{m}_0 = p_1 + 2p_2$ and $F_0 = (1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_2 \leqslant 3) \wedge (1 \leqslant x_3 \leqslant 4)$. The computation of the firing domain of σ is as follows. First, the path $\alpha_0 \xrightarrow{t_1} \alpha_1 \xrightarrow{\varepsilon_2} \alpha_5 \xrightarrow{\varepsilon_4} \alpha_{12}$ is found in the SCG and it is shown as follows (step 2 of Algorithm 4.1).*



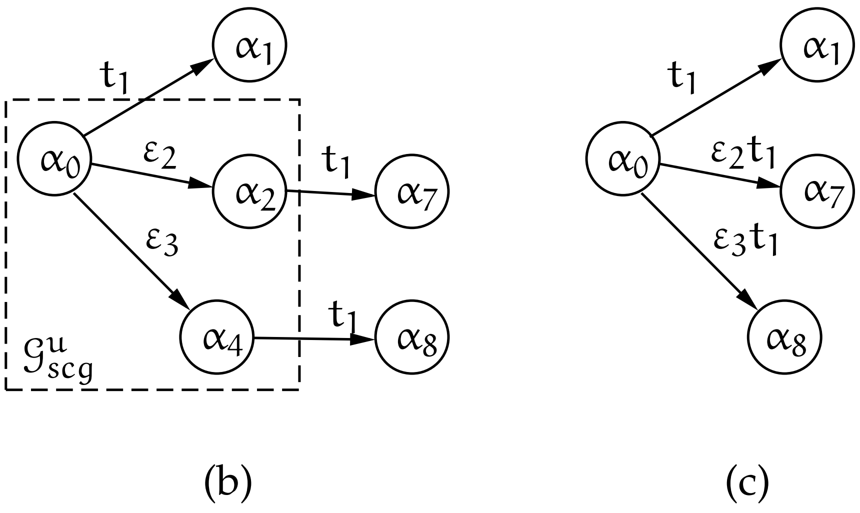


Figure 4.1: (a) A TPN where ε_4 is a fault transition [23]. (b) A part of the SCG starting from the initial class α_0, whose marking is $m_0 = p_1 + 2p_2$. (c) The corresponding FDG. Details of nodes are shown in Table 4.1.

$$\begin{tikzcd}
\alpha_0 : \left|\begin{array}{cc} & p_1 + p_2 \\ \hline & (1 \leqslant x_1 \leqslant 4) \\ \wedge & (2 \leqslant x_2 \leqslant 3) \\ \wedge & (1 \leqslant x_3 \leqslant 4) \end{array}\right. \arrow[r, "t_1"] & \alpha_1 : \left|\begin{array}{cc} & p_1 + p_2 + p_5 \\ \hline & (1 \leqslant x_1 \leqslant 4) \\ \wedge & (0 \leqslant x_2 \leqslant 2) \\ \wedge & (0 \leqslant x_3 \leqslant 3) \\ \wedge & (x_3 - x_2 \leqslant 2) \end{array}\right. \arrow[d, "\varepsilon_3"] \\
\alpha_{12} : \left|\begin{array}{cc} & p_2 + p_6 + p_7 \\ \hline & (0 \leqslant x_1 \leqslant 2) \\ \wedge & (1 \leqslant x_6 \leqslant 1) \\ \wedge & (2 \leqslant x_7 \leqslant 4) \end{array}\right. & \alpha_5 : \left|\begin{array}{cc} & p_2 + p_4 + p_5 \\ \hline & (0 \leqslant x_1 \leqslant 4) \\ \wedge & (2 \leqslant x_4 \leqslant 5) \end{array}\right. \arrow[l, "\varepsilon_4"']
\end{tikzcd}$$

Second, three variables y_1*,* y_2 *and* y_4 *are assigned to* t_1*,* ε_2 *and* ε_4*, respectively. From the order of transitions in* σ*, the first inequalities are* $0 \leqslant y_1 \leqslant y_2 \leqslant y_4$ *(step 4). Let* $\sigma' = t_1\varepsilon_2$ *(step 6) and find the last time when* ε_4 *turns from being disabled to enabled (step 7). The state class when* ε_4 *starts to be enabled is* α_5 *and it is reached after the firing of* $t_1\varepsilon_2$ *(step 8). The firing domain of* α_5 *is* $F_5 = (0 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_4 \leqslant 5)$ *and it means that two transitions* t_1 *and* ε_4 *are enabled at* α_5*. Note that* t_1 *is enabled at* α_0 *and also at* α_1*. By minimizing and maximizing* x_4 *subject to* F_5*, the time interval in which* ε_4 *can be fired is* $[2, 4]$*. The inequality describing the time delay to fire* ε_4 *is* $2 \leqslant y_4 - y_2 \leqslant 4$ *and it is inserted into the constraints* cons *(step 9 to 10). There is not any transition disabled by the firing of* ε_4 *(step 11). Therefore, no other inequality is added to* cons*. The sequence* σ *is set to* $\bar{\sigma} = \sigma' = t_2\varepsilon_2$ *(step 17) and the constraints are*

$$\mathrm{cons} = \begin{cases} \ 2 \leqslant y_4 - y_2 \leqslant 4, \\ 0 \leqslant y_1 \leqslant y_2 \leqslant y_4. \end{cases}$$

After that, ε_2 *is considered, which is enabled at* α_0 *and there is* $\sigma' = t_2$*,* $\sigma'' = \epsilon$ *and* $\sigma''' = t_2$ *(step 7). The constraint* $2 \leqslant y_2 \leqslant 3$ *is obtained by minimizing and maximizing* x_2 *subject to* F_0 *and it is added to* cons *(steps 9 to 10). The transition* ε_2 *is fired at* α_1 *leading to* α_5 *(step 11). Its firing disables* ε_3 *(step 11), which is enabled at* α_0 *(steps 12 and 13). Maximize* x_3 *subject to the firing domain of* α_0*, the upper bound of time delay of* ε_3 *is* 4 *(step 14). The constraint* $y_2 \leqslant 4$ *is attached to* cons *(step 15). After constructing the inequalities of* t_1 *and* ε_2*, the constraints are*

$$\mathrm{cons} = \begin{cases} \quad 2 \leqslant y_2 \leqslant 3, \\ \qquad y_2 \leqslant 4, \\ \ 2 \leqslant y_4 - y_2 \leqslant 4, \\ 0 \leqslant y_1 \leqslant y_2 \leqslant y_4. \end{cases}$$

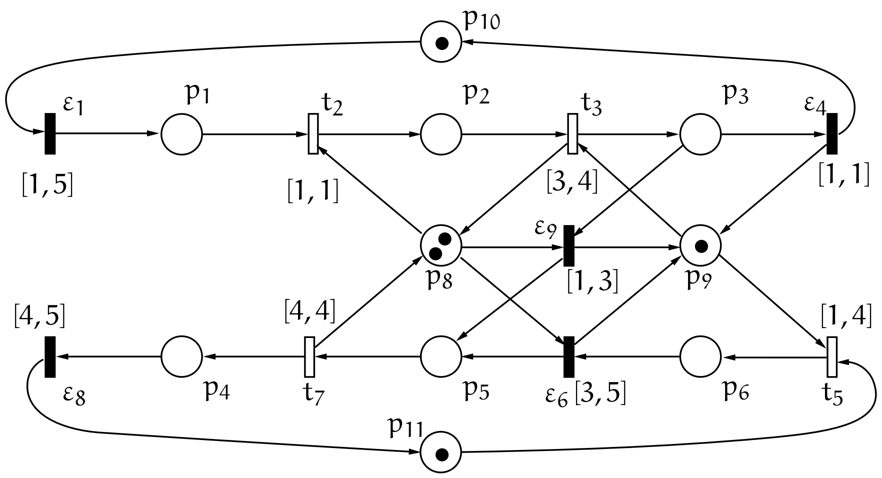


Figure 4.2: A manufacturing system having two sequential processes. One process is represented by p_1, p_2 and p_3; the other one contains p_4, p_5 and p_6. There are two robots p_8 and p_9 moving materials and can be used by both processes. We assume that ε_9 is a fault transition representing that by mistake a material is moved from p_3 to p_5†.

The constraints corresponding to the firing of t_1 *is constructed in the same way and they are attached to* cons*:*

$$\mathrm{cons} = \begin{cases} 1 \leqslant y_1 \leqslant 4, \\ 2 \leqslant y_2 \leqslant 3, \\ y_2 \leqslant 4, \\ 2 \leqslant y_4 - y_2 \leqslant 4, \\ 0 \leqslant y_1 \leqslant y_2 \leqslant y_4. \end{cases}$$

At α_{12}*, transition* t_1 *is enabled but it is not fired, then the inequalities to represent that the last transition* ε_4 *is fired earlier than the upper bound of the time interval of* t_1 *must be attached to* cons *(step 19). The state class at which* t_1 *starts to be enabled is* α_1 *and an inequality* $y_4 - y_1 \leqslant 4$ *is included to* cons *(steps 20 to 23). Minimizing and maximizing* y_4 *subject to*

$$\mathrm{cons} = \begin{cases} y_4 - y_1 \leqslant 4, \\ 1 \leqslant y_1 \leqslant 4, \\ 2 \leqslant y_2 \leqslant 3, \\ y_2 \leqslant 4, \\ 2 \leqslant y_4 - y_2 \leqslant 4, \\ 0 \leqslant y_1 \leqslant y_2 \leqslant y_4. \end{cases}$$

The firing domain of $\sigma = t_1\varepsilon_2\varepsilon_4$ *is obtained as* $[4,7]$.

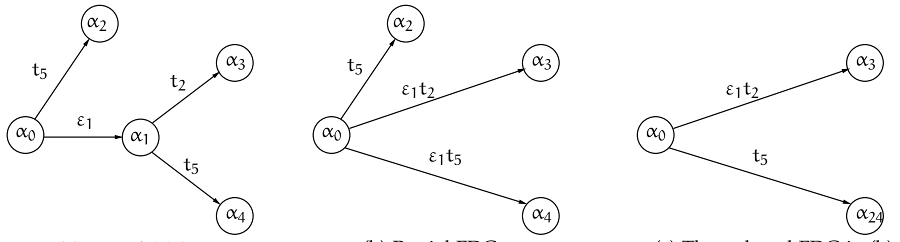


Figure 4.3: A partial SCG and FDG of the TPN in Figure 4.2. The markings of $\alpha_i, i = 0, 1, \ldots, 4$ are m_i such that $m_0 = 2p_8 + p_9 + p_{10} + p_{11}$, $m_1 = p_1 + 2p_8 + p_9 + p_{11}$, $m_2 = p_6 + 2p_8 + p_{10}$, $m_3 = p_2 + p_8 + p_9 + p_{11}$ and $m_4 = p_1 + p_6 + 2p_8$, respectively. The domains of $\alpha_i, i = 0, 1, \ldots, 4$ are F_i such that $F_0 = (1 \leqslant x_1 \leqslant 5, 1 \leqslant x_5 \leqslant 4)$, $F_1 = (1 \leqslant x_2 \leqslant 1, 0 \leqslant x_5 \leqslant 3)$, $F_2 = (0 \leqslant x_1 \leqslant 4, 3 \leqslant x_6 \leqslant 5)$, $F_3 = (3 \leqslant x_3 \leqslant 4, 0 \leqslant x_5 \leqslant 3)$ and $F_4 = (0 \leqslant x_1 \leqslant 1, 3 \leqslant x_6 \leqslant 5)$, respectively.

4.3 FAULT DIAGNOSIS GRAPH

4.3.1 *Motivation*

A *Fault Diagnosis Graph* (FDG) is a projection of SCG such that the information needed for fault diagnosis is kept.

Because of the huge number of reachable state classes, it is time consuming to compute the full SCG. For fault diagnosis, only the parts of the SCG corresponding to the observation and the *consistent states*, which are reached by firing observable transitions are used. Let us assume that no transition has been observed in the PN system in Figure 4.2, where t_2, t_3, t_5 and t_7 are observable and the initial marking is $m_0 = 2p_8 + p_9 + p_{10} + p_{11}$. Assume one fault class $T_f^1 = \{\varepsilon_9\}$. We compute the part of SCG starting from m_0 that contains only two state classes, α_0 and α_1 (see Figure 4.3(a)). Then, by firing observable transitions at α_0 and α_1, the SCG in Figure 4.3(a) is obtained, where more state classes are computed: α_2, α_3 and α_4. The FDG is obtained from the SCG in Figure 4.3(a) by removing the node α_1, because it is reached by firing the unobservable transition ε_1. The information of paths $\varepsilon_1 t_2$ and $\varepsilon_1 t_5$ is associated with the corresponding edges $\alpha_0 \to \alpha_3$ and $\alpha_0 \to \alpha_5$ (see Figure 4.3(b)). The graph used to update the FDG is shown in Figure 4.3(b), where the labels associated with edges are listed in Table 4.2.

In diagnosis we care about whether fault transitions are fired or not, instead of which transitions are fired earlier than the others. It is sufficient to record the firing count vector, not the firing sequence. By means of this vector, the representation of the relevant information becomes more compact, since the firing sequence grows to infinite when the time elapses, but such a vector has a fixed length.

Definition 4.2. *A* fault diagnosis graph *(FDG) is a 4-tuple* $\mathcal{G} = \langle \Omega, \twoheadrightarrow, \alpha_0, \mathcal{L} \rangle$*, where:*

- $\alpha_0 = \langle m_0, F_0 \rangle$ *is the initial state class,*

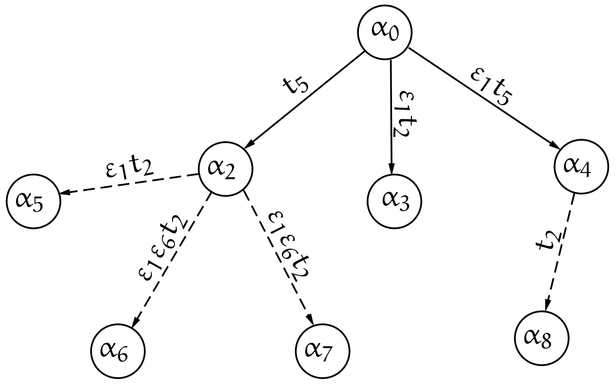


Figure 4.4: The FDG corresponding to $w = t_5$ is observed at time 4 in the PN system of in Figure 4.2.

Table 4.2: Details of edges in the FDG shown in Figure 4.4

edge	t_o	I	D
$\alpha_0 \to \alpha_2$	t_5	$[1,4]$	(N)
$\alpha_0 \to \alpha_3$	t_2	$[2,4]$	(N)
$\alpha_0 \to \alpha_4$	t_5	$[1,4]$	(N)
$\alpha_2 \to \alpha_5$	t_2	$[1,5]$	(N)
$\alpha_2 \to \alpha_6$	t_2	$[3,5]$	(N)
$\alpha_2 \to \alpha_7$	t_2	$[4,5]$	(N)
$\alpha_4 \to \alpha_8$	t_2	$[0,1]$	(N)

- $\twoheadrightarrow$ *is the set of edges, such that* $\alpha \twoheadrightarrow \alpha'$ *means* $\exists \sigma_u \in T_u^*, t_o \in T_o$ *and* α' *is reachable from* α *by firing* $\sigma_u t_o$*,*
- $\Omega = \{\alpha | \alpha_0 \twoheadrightarrow^* \alpha\}$*, where* $\twoheadrightarrow^*$ *is the reflexive and transitive closure of* $\twoheadrightarrow$*, is the set of reachable state classes,*
- $\mathcal{L} : \twoheadrightarrow \to T_o \times \{\mathbb{Q}_0 \times (\mathbb{Q}_0 \cup \{\infty\})\} \times \{N, F\}^r$ *is a labeling function of edges, i.e., for an edge* e *representing* $\alpha \xrightarrow{\sigma_u t_o} \alpha'$*, the label of* e *is* $\mathcal{L}(e) = \langle t_o, I_{\sigma_u t_o}, D_{\sigma_u t_o} \rangle$*, where* t_o *is the observed transition,* $I_{\sigma_u t_o}$ *is the firing domain of* $\sigma_u t_o$ *and* $D_{\sigma_u t_o}$ *records the firing of fault transitions in* $\sigma_u t_o$*.* ■

An FDG is a directed multi graph. Each node (excepting the initial one) corresponds to a state class, which can be reached by the firing of an observable transition preceded eventually by the firings of some unobservable transitions. A label is associated with each edge containing:

1. the observable transition,
2. the firing domain of the corresponding firing sequence,
3. the diagnosis information.

The diagnosis information is a vector representing the firing of fault transitions in the corresponding sequence. We denote the edge from node α to α' with the observable transition t, the firing domain I and the diagnosis information D as $\langle \alpha, \langle t, I, D \rangle, \alpha' \rangle$. For example, in Table 4.2, the label of $\alpha_0 \to \alpha_3$ is $\langle t_5, [2,4], (N) \rangle$, where t_5 is the observed transition, $[2,4]$ is the firing domain of the corresponding firing sequence and (N) indicates that the fault transition is not fired in the unobservable firing sequence.

4.3.2 *Construction of an FDG*

The approach to construct an FDG is straightforward. Given a consistent state class (node) α in the FDG, the subgraph of the FDG starting from α is computed from the unobservable SCG, in which α is the initial state class. The unobservable SCG is obtained by firing only unobservable transitions.

In the construction of FDG, first, we find in the unobservable SCG all state classes enabling observable transitions. All the paths are computed from the initial state class of the SCG (i.e., α_0) to those ones. As we assumed that the unobservable subnet is acyclic, the number of these paths is finite.

Notice that the state classes, in which the TPN can be before the firing of the first observable transition (i.e., α_1 in Figure 4.3(a)), are not explicitly given in the FDG. However, the diagnosis states can be computed by considering only the edges obtained in the FDG since they contain the same information.

The computation of the FDG is incremental according to the observation. Algorithm 4.2 constructs the FDG $\mathcal{G}$ by using the unobservable

SCG $\mathcal{G}_{scg}$. The initial state class of $\mathcal{G}_{scg}$ is (any) one of the consistent state classes of the observation. The unobservable SCG is computed by firing only unobservable transitions. Algorithm 4.2 also updates the set of unexplored state classes W, which is both an input and an output parameter of Algorithm 4.2.

Algorithm 4.2 $[\mathcal{G}, W] = \text{FDG}(\mathcal{N}, \mathcal{G}, \mathcal{G}_{scg}, W)$

```
 1: let $\alpha_0$ be the initial state class of $\mathcal{G}_{scg} = \langle \Omega_{scg}, \twoheadrightarrow_{scg}, \alpha_0 \rangle$
 2: for each $\alpha_u = \langle m_u, F_u \rangle \in \mathcal{G}_{scg}$ do
 3:   for each $t \in En(m_u)$ s.t. $t \in T_o \wedge$
          isFireable$(\alpha_u, t)$ do
 4:     $\alpha := succ(\alpha_u, t)$                          ▷ compute the successor
 5:     if $\alpha \notin \mathcal{G}$ then                               ▷ if $\alpha$ is not explored
 6:       $W = W \cup \{\alpha\}$                                    ▷ update $W$
 7:     end if
 8:     let $\Sigma$ be the set of paths from $\alpha_0$ to $\alpha_u$
          in $\mathcal{G}_{scg}$
 9:     for each $\sigma \in \Sigma$ do
10:       $I := domain(\mathcal{G}_{scg}, \sigma t, \alpha_0)$                ▷ the firing domain of $\sigma t$
11:       add $\alpha$ and $\langle \alpha_0, \langle \overrightarrow{\sigma t}, I \rangle, \alpha \rangle$ to $\mathcal{G}$
12:     end for
13:   end for
14: end for
15: return $\mathcal{G}, W$
```

For every state class α_u in the unobservable SCG $\mathcal{G}_{scg}$, which enables an observable transition (step 3), the successor state class α (of α_u) is computed by firing t (step 4). If α is a newly obtained state class such that it has not been explored, then α is added into the set of unexplored state classes W (step 6). All paths from the initial state class α_0 of the $\mathcal{G}_{scg}$ to α_u are computed in $\mathcal{G}_{scg}$ (step 8), and for each of them, its firing domain I is computed (step 10). The FDG is updated with the state class α (as a new node) and an edge $\langle \alpha_0, \langle \overrightarrow{\sigma t}, I \rangle, \alpha \rangle$ (step 11).

Example 4.3. *Let us consider the net system of Figure 4.1(a) with a fault class $T_f^1 = \{\varepsilon_4\}$. The third input parameter of Algorithm 4.2 is the unobservable SCG $\mathcal{G}_{scg}$ starting from the initial state class α_0, and it contains three state classes α_0, α_2 and α_4 (shown in Figure 4.1(b)). The last input parameter is the set of unexplored state classes is $W = \{\alpha_0\}$. The observable transition t_1 is fireable at α_4, and then by firing t_1 its successor is α_8 (step 4). Because α_8 is not in $\mathcal{G}$, it is added into W and $W = \{\alpha_8\}$ (step 6). The path from α_0 to α_8 is $\alpha_0 \xrightarrow{\varepsilon_3} \alpha_4 \xrightarrow{t_1} \alpha_8$ (step 8). By applying 10, the firing domain of $\varepsilon_3 t_1$ is obtained equal to $[1,4]$ (step 10). Then the node α_8 and the edge $\langle \alpha_0, \langle \overrightarrow{\varepsilon_3 t_1}, [1,4] \rangle, \alpha_8 \rangle$ are included in $\mathcal{G}$ (step 11). After the execution of the algorithm, the FDG in Figure 4.1(c) is constructed and the labels of edges in the FDG are:*

- *$\alpha_0 \to \alpha_1$ is labeled with $\langle \overrightarrow{t_1}, [1,3] \rangle$,*
- *$\alpha_0 \to \alpha_7$ is labeled with $\langle \overrightarrow{\varepsilon_2 t_1}, [2,4] \rangle$,*

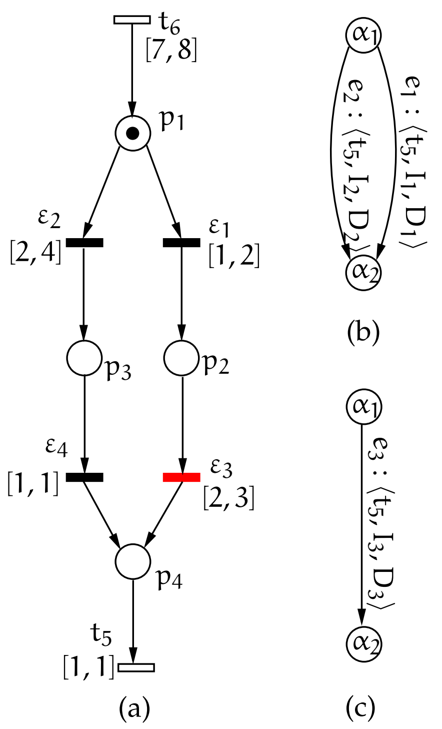


Figure 4.5: Reduction rule 1: A TPN (a) and a part of its FDG (b). The FDG is reduced to the one in (c) by merging two edges in a single one. In (b) and (c), $I_1 = I_2 = I_3 = [4,6]$, $D_1 = \langle F \rangle$, $D_2 = \langle N \rangle$, $D_3 = \langle U \rangle$, $T_F^1 = \{\varepsilon_3\}$

- $\alpha_0 \to \alpha_8$ *is labeled with* $\langle \overrightarrow{\varepsilon_3 t_1}, [1,4] \rangle$.

Three state classes are reachable from the initial state class by firing only unobservable transitions followed by an observable one. When an observable transition is fired, the state classes consistent with the observation can be found by looking at the output edges of α_0*. After updating the FDG,* α_0 *is removed from W, because it has been explored and the three new state classes are added into W. Finally, the set of unexplored state classes is* $W = \{\alpha_1, \alpha_7, \alpha_8\}$.

4.3.3 *Reduction of an FDG*

As already pointed out, the number of nodes in an FDG is smaller than the one of the corresponding SCG, since the state classes obtained by the firing of the unobservable transitions are not kept (e.g., α_1 in Figure 4.3(a)). Moreover, in order to save storage space, three reduction rules can be used to eliminate edges and nodes from an FDG, while important information for diagnosis is kept. The important information includes observable transitions, firing domains and the firing of fault transitions.

The first reduction rule is illustrated using the PN in Figure 4.5(a). Assume that the initial marking is $\mathbf{m}_0 = p_1$ and the only fault class is $T_f^1 = \{\varepsilon_3\}$. Its FDG is shown in Figure 4.5(b) in which the initial state is α_1 and the two edges are:

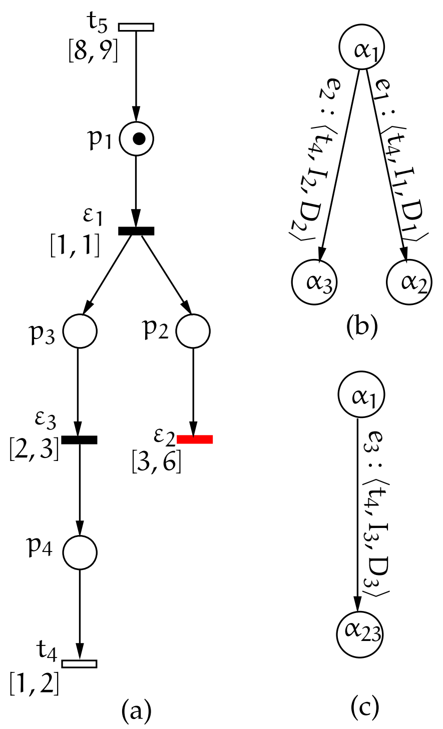


Figure 4.6: Reduction rule 2: A TPN (a) and a part of its FDG (b). The FDG in (b) is reduced to the one in (c) by merging nodes α_3 and α_2 in α_{23}. In (b) and (c), $I_1 = I_2 = I_3 = [4, 6]$, $D_1 = \langle N \rangle$, $D_2 = \langle F \rangle$, $D_3 = \langle U \rangle$, $T_F^1 = \langle \varepsilon_2 \rangle$

1. $e_1 : \langle t_5, I_1 = [4, 6], D_1 = \langle F \rangle \rangle$ corresponding to the firing sequence $\varepsilon_1 \varepsilon_3 t_5$,

2. $e_2 : \langle t_5, I_2 = [4, 6], D_2 = \langle N \rangle \rangle$ corresponding to $\varepsilon_2 \varepsilon_4 t_5$.

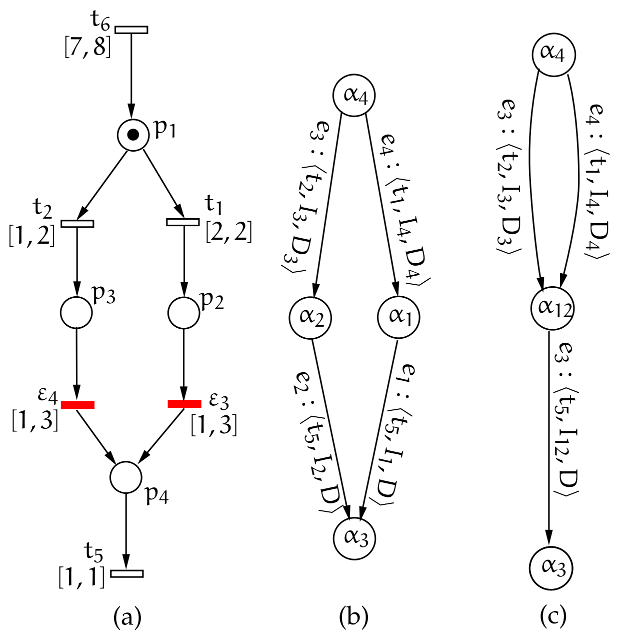
The observable transition and firing domains of e_1 and e_2 are the same. The transition t_5 will be fired in an interval $I_1 = I_2 = [4, 6]$. When t_5 is observed, the consistent state is α_2 and the labels of two edges (i.e., e_1 and e_2) are used to compute the diagnosis states. Since the time intervals in e_1 and e_2 are the same, both edges can be merged in $e_3 = \langle t_5, I_3 = [4, 6], D_3 = \langle U \rangle \rangle$ (Figure 4.5(c)). Therefore, the number of edges of the FDG is reduced by one in this situation.

The second reduction rule for the FDG is shown by using the TPN in Figure 4.6(a), whose initial marking is $m_0 = p_1$ and assume the fault class $T_f^1 = \{\varepsilon_2\}$ and the only observable transition is t_4. A part of its FDG (not firing ε_5) is shown in Figure 4.6(b). The initial state is α_1 having two output edges:

1. $e_1 : \langle t_4, I_1 = [4, 6], D_1 = \langle N \rangle \rangle$ corresponding to the firing sequence $\varepsilon_1 \varepsilon_3 t_4$ and

2. $e_2 : \langle t_4, I_2 = [4, 6], D_2 = \langle F \rangle \rangle$ corresponding to $\varepsilon_1 \varepsilon_3 \varepsilon_2 t_4$.

The observable transition t_4 can be observed in time interval $[4, 6]$. If it is observed, two state classes α_2 and α_3 are consistent with the observation, and both e_1 and e_2 will be considered in diagnosis. Obviously, α_2 and α_3 can be merged into one node α_{23} and the edges



t_6 [7,8]
p_1
t_2 [1,2]
t_1 [2,2]
p_3
p_2
ε_4 [1,3]
ε_3 [1,3]
p_4
t_5 [1,1]
(a)
α_4
$e_3 : \langle t_2, I_3, D_3 \rangle$
$e_4 : \langle t_1, I_4, D_4 \rangle$
α_2
α_1
$e_2 : \langle t_5, I_2, D \rangle$
$e_1 : \langle t_5, I_1, D \rangle$
α_3
(b)
α_4
$e_3 : \langle t_2, I_3, D_3 \rangle$
$e_4 : \langle t_1, I_4, D_4 \rangle$
α_{12}
$e_3 : \langle t_5, I_{12}, D \rangle$
α_3
(c)

Figure 4.7: Reduction rule 3: A TPN (a) and its FDG (b). The FDG in (b) is reduced to the one in (c) by merging nodes α_1 and α_2 into α_{12}. In (b) and (c), $I_1 = I_2 = I_{12} = [1,3]$, $I_3 = [1,2]$, $I_4 = [2,2]$, $D = \langle F \rangle$, $D_3 = D_4 = \langle N \rangle$, $T_F^1 = \{\varepsilon_3, \varepsilon_4\}$

reduced to $e_3 : \langle t_4, I_3 = [4,6], D_3 = \langle U \rangle \rangle$. The resulted FDG is shown (see Figure 4.6(c)).

We can adapt the previous rule to merge the predecessors of a node. Let us consider the TPN in Figure 4.7(a) with the initial marking $m_0 = p_1$ corresponding to the state class α_4. Assume that the fault class is $T_f^1 = \{\varepsilon_3, \varepsilon_4\}$ and the observable transitions are t_1, t_2 and t_5. At α_4, t_1 is enabled and by firing it, state α_1 is obtained (corresponding to the marking $m_1 = p_2$). Moreover, t_2 is also enabled at α_4 and by firing it state class α_2 is obtained (corresponding to the marking $m_2 = p_3$). The node α_3 (corresponding to $m_3 = \vec{0}$) is reachable by the firing sequences $\varepsilon_3 t_5$ from α_1 or $\varepsilon_4 t_5$ from α_2 (Figure 4.7(b)). When α_3 is reached, the edges $\alpha_1 \rightarrow \alpha_3$ and $\alpha_2 \rightarrow \alpha_3$ provide the same information for fault diagnosis, since both correspond to firing sequences that include fault transitions (ε_3 and ε_4) finishing with the firing of the same observable transition. For this reason, the nodes α_1 and α_2 can be merged into one node α_{12} (Figure 4.7(c)).

For example, by applying the previous rule to merge α_2 and α_4 in the FDG shown in Figure 4.3(b), node α_{24} in Figure 4.3(c) is obtained[‡].

‡ In order to explain more clearly how diagnosis algorithms works, in the sequel we use the original FDG, not the reduced one.

5 CENTRALIZED DIAGNOSIS ALGORITHMS

In this chapter, we apply *fault diagnosis graph* (Chapter 4) to centralized diagnosis. In centralized diagnosis, a single (unique) diagnoser conducts the diagnosis task. It monitors the firings of all observable transitions, constructs the fault diagnosis graph and computes diagnosis states of the target system. As will be illustrated in this chapter, only a part of the full diagnosis graph is used in the diagnosis. In order to analyze the complexity of centralized diagnosis, a section of discussion addresses both time and space complexities. In the space complexity analysis, we propose an approach to compute an upper bound of the number of consistent state classes, which is critical to the use of computer memory (space) and the time needed to construct fault diagnosis graph and diagnosis.

5.1 INTRODUCTION

In this chapter, we propose centralized diagnosis based on FDG. General steps of the centralized diagnosis are presented in Section 5.2. For the fault diagnosis, the construction of FDG is performed incrementally, when a new observation is available. Therefore, the diagnosis contains mainly two steps: update the FDG and diagnosis on the FDG. In centralized diagnosis, a (unique) centralized diagnoser (monitor) collects observations and computes diagnosis states. After initializing the FDG, the diagnoser waits for an observation. With an observation, the diagnoser update the FDG, if the corresponding parts are not in the FDG. After that, the diagnosis algorithm is applied to compute diagnosis states. The diagnosis algorithm is given in Section 5.3. Diagnosis states, corresponding to an observed word, describe:

1. whether any fault transition has been fired when the current observed word appears;
2. whether there exist any fault transition that can be fired before next possible observed words.

Hence, when an observable transition is fired, we first look for all *consistent state classes* in the FDG, and second explore the *subgraphs* starting at the consistent state classes for the firing of fault transitions before next observation. Because it uses labels associated with edges in the FDG, the diagnosis process is efficient. An example is given to illustrate the steps in the centralized diagnosis.

We also discuss the time and space complexity of constructing FDG and diagnosing on it. We show that the number of consistent state classes is polynomial according to the length of the observation. The results on complexity analysis can also be used in decentralized diagnosis in Chapter 6.

5.2 GENERAL ALGORITHM

The fault diagnosis algorithm is presented in Algorithm 5.1. The algorithm first initializes the FDG and conducts the diagnosis corresponding to the empty observation. After that, it waits for an observation. When a transition is observed, the algorithm updates the FDG if it is necessary, and then computes diagnosis states.

Before the system starts to evolve, i.e., the observation word is $w = \epsilon$, the SCG of the unobservable subnet with the initial state class α_0 is computed (step 2). Based on it, the FDG corresponding to the empty word is obtained (step 3). In the same step, W is initialized with the set of nodes in the actual FDG minus α_0. The state classes in W will not be explored at this observation, because it is enough to check the diagnosis state based only on the labels of the output edges of α_0. If a transition of a fault class T_f^i is fired in all output edges of α_0, the diagnosis state is set to F; if no fault transition is fired in any output edge of α_0, the diagnosis state is set to N; otherwise, the diagnosis

Algorithm 5.1 General fault diagnosis algorithm

```
1: $\nu := \epsilon$
2: $\mathcal{G}_{scg} := \mathrm{SCG}(\mathcal{N}_u, \alpha_0)$
                              ▷ Compute the unobservable SCG starting from $\alpha_0$
3: $[\mathcal{G}, W] := \mathrm{FDG}(\mathcal{N}, \emptyset, \mathcal{G}_{scg}, \{\alpha_0\})$
                              ▷ Construct FDG consistent with the empty word
4: compute the diagnosis state based on the labels of the output
   edges of $\alpha_0$ in $\mathcal{G}$
5: $\mathcal{S} := \{\langle \alpha_0, \vec{0} \rangle\}$
                 ▷ Next observation is obtained by exploring the output edges of
   state classes in $\mathcal{S}$
6: let $t_j$ be a new observation and $w = \nu t_j$ at $\tau_w$
7: if $W \neq \emptyset$ then
8:   for each $\alpha \in \mathcal{S}$ s.t. $\exists \langle \alpha, \langle \overrightarrow{\sigma t_j}, I \rangle, \alpha' \rangle \in \mathcal{G} \wedge \alpha' \in W$ do
9:     $\mathcal{G}_{scg} := \mathrm{SCG}(\mathcal{N}_u, \alpha')$
                              ▷ compute the unobservable SCG starting from $\alpha'$
10:    $[\mathcal{G}, W] := \mathrm{FDG}(\mathcal{N}, \mathcal{G}, \mathcal{G}_{scg}, W)$
                                        ▷ update the FDG by adding $\mathcal{G}_{scg}$
11:    $W := W \setminus \{\alpha'\}$                ▷ Remove explored $\alpha'$ from W
12:  end for
13: end if
14: $[\Delta(w), \mathcal{S}] := \mathrm{diag}(\mathcal{G}, \mathcal{S}, t_j, \delta_\tau)$
                              ▷ $\delta_\tau$ is the time from last observation to this one
15: go to 6
```

state is set to U (step 4). The set $\mathcal{S}$ contains nodes of the FDG, which are consistent with the observation, together with the firing count vectors of the paths from the initial state class to them.

When a new observation comes (step 6), the FDG is updated if there exists unexplored nodes (step 7). For each α in $\mathcal{S}$, if it has an output edge that, in the firing count vector, the observed transition is fired and the output state class is not explored (step 8), then the unobservable SCG starting from the output state class is computed (step 9). In order to update the FDG, we first fire observable transitions at all state classes in $\mathcal{G}_{scg}$ (if possible), and then the state classes obtained by firing unobservable transitions are removed and the resulted graph is added to the FDG (step 10). After the computation of the FDG corresponding to the current observation, the diagnosis states are calculated, while $\mathcal{S}$ is updated (step 14).

5.3 CENTRALIZED DIAGNOSIS ON FDG

After updating the FDG, diagnosis states are computed. The diagnosis starts from the state classes consistent with the previous observation, i.e., the state classes in $\mathcal{S}$. When a transition is observed, we determine at which state classes it can be fired by using the firing count vectors of the output edges of the state classes in $\mathcal{S}$. Algorithm 5.2 constructs the set of consistent state classes corresponding to the current observation. It also checks if fault transitions are fired or not by using the firing counts of fault transitions in the path vectors, from the initial state class of the FDG to the consistent state classes, and the firing count vectors labeled on the output edges of consistent state classes.

Algorithm 5.2 $[\Delta(w), \mathcal{S}] := \text{diag}(\mathcal{G}, \mathcal{S}, t_j, \delta_\tau)$

```
1: $\mathcal{S}_f := \emptyset$
2: for each $\langle \alpha, \overrightarrow{\sigma}_\alpha \rangle \in \mathcal{S}$ do
3:   for each $\langle \alpha, \langle \overrightarrow{\sigma_u t_j}, I \rangle, \alpha_1 \rangle \in \mathcal{G}$ s.t. $l_I \leqslant \delta_\tau \leqslant r_I$ do ▷ $\delta_\tau$ is the time between the pervious observation and the current one, $I = [l_I, r_I]$
4:     $\overrightarrow{\sigma}_{\alpha_1} := \overrightarrow{\sigma}_\alpha + \overrightarrow{\sigma_u t_j}$
5:     $\mathcal{S}_f := \mathcal{S}_f \cup \{\langle \alpha_1, \overrightarrow{\sigma}_{\alpha_1} \rangle\}$
6:     update $\Delta(w)$ by $\overrightarrow{\sigma}_{\alpha_1}$ and the output edges of $\alpha_1$
7:   end for
8: end for
9: $\mathcal{S} := \mathcal{S}_f$
10: return $[\Delta(w), \mathcal{S}]$
```

In Algorithm 5.1, for each element $\langle \alpha, \overrightarrow{\sigma}_\alpha \rangle$ of $\mathcal{S}$, the algorithm checks if the transition t_j is fireable at α by using the firing count vectors associated with the output edges of α in the FDG. Let α_1 be a state class reachable from α such that t_j is fired according to the firing count vector of the edge from α to α_1. If α_1 is consistent with the observation (step 3), the firing count vector of the path from the initial state class of the FDG to α_1 is computed ($\overrightarrow{\sigma}_{\alpha_1}$) and $\langle \alpha_1, \overrightarrow{\sigma}_{\alpha_1} \rangle$ is inserted into $\mathcal{S}_f$ (steps 4 and 5). Using both $\overrightarrow{\sigma}_{\alpha_1}$ and the output edges of α_1, the diagnosis states in $\Delta(w)$ are updated (step 6). The following steps are performed to update diagnosis states:

1. Considering T_f^i, if $\exists t_f \in T_f^i$ and the firing count of t_f in $\overrightarrow{\sigma}_{\alpha_1}$ is not zero, then it has been fired and the diagnosis state of T_f^i corresponding to α_1 is set to F. If t_f is not fired in the path, then check the firing counts of t_f in output edges of α_1. If all the firing counts are greater than zero, then the diagnosis state of T_f^i is set to F; if they are zero, then the diagnosis state is set to N; otherwise, the diagnosis state is set to U.

2. If the diagnosis state of T_f^i is empty, i.e., $\Delta(w, T_f^i) = \epsilon$, $\Delta(w, T_f^i)$ is set to the diagnosis state of α_1. If $\Delta(w, T_f^i) = N$ and the diagnosis state in the first step is N too, then the updated diagnosis state of T_f^i is N. If both of them are F, then the updated diagnosis state is $\Delta(w, T_f^i) = F$. Otherwise, the updated diagnosis state is U.

5.4 EXAMPLE

Continuing Example 4.3, let us assume t_1 is observed at time 4. The FDG in Figure 4.1(c) is updated using Algorithm 4.2 and it is shown in Figure 5.1. Nodes and the labels of edges are given in Table 5.1 and Table 5.2, respectively. The input parameters of Algorithm 5.2 are:

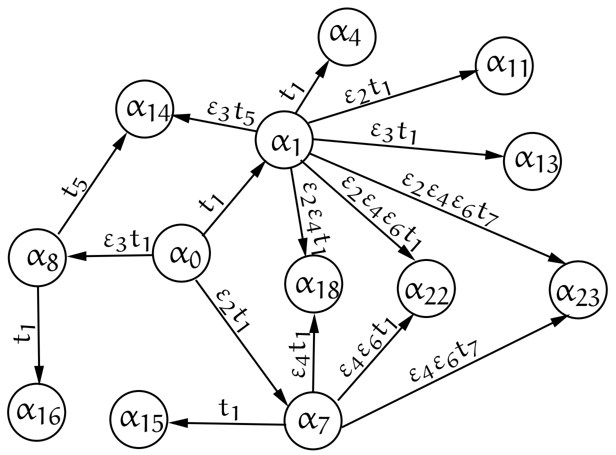


Figure 5.1: The FDG corresponding to $w = t_1$. For diagnosis, we need to consider all consistent firing sequences of the form: $\sigma = \sigma'_u t_1 \sigma''_u$ where σ'_u and σ''_u are firing sequences of unobservable transitions. Since in the FDG we keep only the nodes corresponding to the firing of observable transitions, we will fire after σ''_u the enabled observable transitions. Notice that we assume that the observable transitions are normal transitions which implies that the diagnosis state is not affected by firing of an observable transition after σ.

1. $\mathcal{G}$ is the updated FDG;
2. $\mathcal{S} = \{\langle \alpha_0, \overrightarrow{0} \rangle\}$ and it will be updated during the execution of the algorithm;
3. t is t_1;
4. $\delta_\tau = 4$, because t_1 is the first observed transition and it is observed at 4.

For $\langle \alpha_0, \overrightarrow{0} \rangle \in \mathcal{S}$, there are three output edges from α_0 in $\mathcal{G}$ to α_1, α_7 and α_8, respectively. The label of the edge $\alpha_0 \to \alpha_1$ is $\langle \overrightarrow{t}_1, [1,3] \rangle$. Although t_1 is fired, it cannot be fired at time 4, because $\delta_\tau = 4$ is not in the firing domain associated with the edge (step 3).

The edge $\alpha_0 \to \alpha_7$ is considered and its label is $\langle \varepsilon_2 t_1, [2,4] \rangle$. The observed transition t_1 is fired according to the firing count vector and δ_τ is in the firing domain $[2,4]$ (step 3). The path vector of α_7 is computed as $\sigma_{\alpha_7} = \overrightarrow{0} + \overrightarrow{\varepsilon_2 t_1} = \overrightarrow{\varepsilon_2 t_1}$ (step 4). Corresponding to the current observation, the set of consistent state classes $\mathcal{S}_f$ is updated to $\mathcal{S}_f = \{\langle \alpha_7, \overrightarrow{\varepsilon_2 t_1} \rangle\}$ (step 5).

In order to update $\Delta(w)$, the path vector of α_7, which is $\overrightarrow{\varepsilon_2 t_1}$, and the labels of four output edges of α_7 are considered (step 6). There is one fault class in the net system $T_f^1 = \{\varepsilon_4\}$. The fault transition ε_4 is not fired in the path vector of α_7. In the label of the edge $\alpha_7 \to \alpha_{15}$, which is $\langle \overrightarrow{t}_1, [1,4] \rangle$, ε_4 is not fired. In the labels of other three edges, ε_4 is fired, and then the diagnosis state corresponding to α_7 is U. Because, in $\Delta(w)$, $\Delta(w, T_f^1) = N$, the diagnosis state of T_f^1 is updated to $\Delta(w, T_f^1) = U$.

Table 5.1: State classes of the FDG in Figure 5.1

state class	marking	firing domain
α_0	$p_1 + 2p_2$	$(1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_2 \leqslant 3) \wedge (1 \leqslant x_3 \leqslant 4)$
α_1	$p_1 + p_2 + p_5$	$(1 \leqslant x_1 \leqslant 4) \wedge (0 \leqslant x_2 \leqslant 2) \wedge (0 \leqslant x_3 \leqslant 3) \wedge (x_3 - x_2 \leqslant 2)$
α_4	$p_1 + 2p_5$	$(0 \leqslant x_2 \leqslant 1) \wedge (0 \leqslant x_3 \leqslant 2)$
α_7	$p_2 + p_4 + p_5$	$(1 \leqslant x_1 \leqslant 4) \wedge (2 \leqslant x_4 \leqslant 5)$
α_8	$p_2 + p_3 + p_5$	$(1 \leqslant x_1 \leqslant 4) \wedge (3 \leqslant x_5 \leqslant 5)$
α_{11}	$p_4 + 2p_5$	$0 \leqslant x_4 \leqslant 5$
α_{13}	$p_3 + 2p_5$	$0 \leqslant x_5 \leqslant 5$
α_{14}	$p_2 + p_6 + p_7$	$(0 \leqslant x_1 \leqslant 1) \wedge (1 \leqslant x_6 \leqslant 1) \wedge (2 \leqslant x_7 \leqslant 4)$
α_{15}	$p_4 + 2p_5$	$0 \leqslant x_4 \leqslant 4$
α_{16}	$p_3 + 2p_5$	$0 \leqslant x_5 \leqslant 4$
α_{18}	$p_5 + p_6 + p_7$	$(0 \leqslant x_6 \leqslant 1) \wedge (1 \leqslant x_7 \leqslant 4) \wedge (1 \leqslant x_7 - x_6 \leqslant 3)$
α_{22}	$p_2 + p_5 + p_6$	$(1 \leqslant x_1 \leqslant 4) \wedge (0 \leqslant x_7 \leqslant 3)$
α_{23}	$p_1 + 2p_2$	$(0 \leqslant x_1 \leqslant 0) \wedge (2 \leqslant x_2 \leqslant 3) \wedge (1 \leqslant x_3 \leqslant 4)$

Table 5.2: Edges of the FDG in Figure 5.1

edge	firing vector	firing domain
$\alpha_0 \to \alpha_1$	$\overrightarrow{t_1}$	$[1, 3]$
$\alpha_0 \to \alpha_7$	$\overrightarrow{\varepsilon_2 t_1}$	$[2, 4]$
$\alpha_0 \to \alpha_8$	$\overrightarrow{\varepsilon_3 t_1}$	$[1, 4]$
$\alpha_1 \to \alpha_4$	$\overrightarrow{t_1}$	$[1, 2]$
$\alpha_1 \to \alpha_{11}$	$\overrightarrow{\varepsilon_2 t_1}$	$[1, 4]$
$\alpha_1 \to \alpha_{13}$	$\overrightarrow{\varepsilon_3 t_1}$	$[1, 4]$
$\alpha_1 \to \alpha_{14}$	$\overrightarrow{\varepsilon_3 t_5}$	$[3, 6]$
$\alpha_1 \to \alpha_{18}$	$\overrightarrow{\varepsilon_2 \varepsilon_4 t_1}$	$[2, 4]$
$\alpha_1 \to \alpha_{22}$	$\overrightarrow{\varepsilon_2 \varepsilon_4 \varepsilon_6 t_1}$	$[3, 4]$
$\alpha_1 \to \alpha_{23}$	$\overrightarrow{\varepsilon_2 \varepsilon_4 \varepsilon_6 t_7}$	$[3, 4]$
$\alpha_7 \to \alpha_{15}$	$\overrightarrow{t_1}$	$[1, 4]$
$\alpha_7 \to \alpha_{18}$	$\overrightarrow{\varepsilon_4 t_1}$	$[2, 4]$
$\alpha_7 \to \alpha_{22}$	$\overrightarrow{\varepsilon_4 \varepsilon_6 t_1}$	$[3, 4]$
$\alpha_7 \to \alpha_{23}$	$\overrightarrow{\varepsilon_4 \varepsilon_6 t_7}$	$[4, 4]$
$\alpha_8 \to \alpha_{14}$	$\overrightarrow{t_5}$	$[3, 4]$
$\alpha_8 \to \alpha_{16}$	$\overrightarrow{t_1}$	$[1, 4]$

After considering the edge $\alpha_0 \to \alpha_7$, the algorithm updates $\Delta(w)$ with the other edge $\alpha_0 \to \alpha_8$. Finally, the diagnosis state is $\Delta(w) = \{\Delta(w, T_f^1) = U\}$ and the set of consistent state classes corresponding to the observation is $\mathcal{S} = \{\langle \alpha_7, \overrightarrow{\varepsilon_2 t_1} \rangle, \langle \alpha_8, \overrightarrow{\varepsilon_3 t_1} \rangle\}$.

5.5 BOUNDEDNESS

In general, the boundedness of the number of state classes of TPNs is an undecidable problem [11]. However, a TPN has a bounded number of state classes if it is bounded and the bounds of time durations are rational for all transitions [10]. The FDG is completely constructed when the set of unexplored state classes (W) is emptied (all state classes have been explored). In the worst case, the number of state classes in the FDG equals the one in the SCG.

5.6 TIME COMPLEXITY

The LPP obtained from F the firing domain of a state class $\langle \mathfrak{m}, F \rangle$) is used in both Algorithm 2.1 and Algorithm 4.1. We first discuss the construction of the constraints of the LPP. In a state class $\langle \alpha, F \rangle$, F is the conjunction of inequalities representing the firing domain of enabled transitions. In the LPP, the number of variables is the number of enabled transitions, and, in the worst case equals $|T|$. Assuming there are n enabled transitions, then $2n$ inequalities are used for the time bounds of transitions in F. The order of the firings of two transitions in F needs two inequalities to be described. In the worst case, the number of inequalities of each two transitions in F is $n(n-1)$. Therefore, in the worst case the LPP includes $|T|$ variables and $|T|(|T|+1)$ inequalities.

The time complexity of Algorithm 4.1 (the algorithm of computing the firing domain of a given firing sequence) depends on the length of the firing sequence. Considering a firing sequence of length k, in order to find when the i-th transition is enabled, the algorithm checks at most $i-1$ state classes in step 7. Therefore, the algorithm checks $k(k-1)/2$ state classes for all transitions in the firing sequence. After that, for each transition, its firing domain is calculated by using the domain F of the state class $\langle \alpha, F \rangle$, at which the transition is enabled. If the firing of a transition disables h transitions, h inequalities are attached to the set of constraints. The number h depends on the number of choices at a place and in the worst case, the firing of a transition can disable $|T|-1$ transitions. In Algorithm 4.1 step 24, the $cons$ contains k inequalities obtained in step 4, $2k$ inequalities representing the time delays of all transitions and $k(|T|-1)$ inequalities represent the conflict between transitions.

Algorithm 4.2 finds paths from the initial state class of the unobservable SCG ($\langle \Omega, \twoheadrightarrow, \alpha_0 \rangle$) to each state class where observable transitions can be fired. The time complexity of finding paths between two state classes is $\mathcal{O}(|\twoheadrightarrow| + |\Omega|)$, when the breadth-first search is applied. Therefore, the complexity of computing paths from the initial

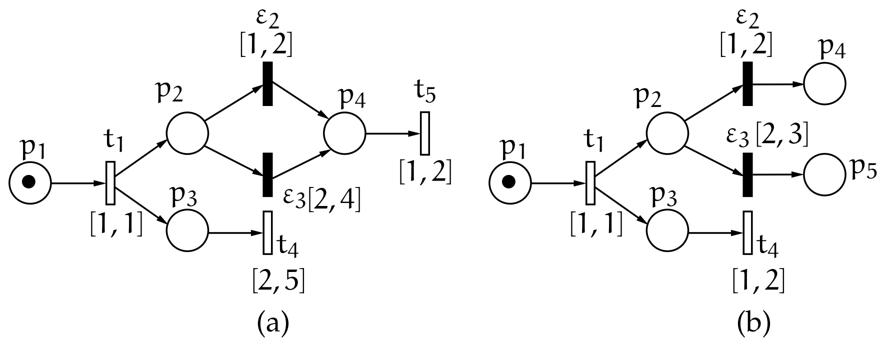


Figure 5.2: Examples that the numbers of consistent states in TPN and untimed PN are not comparable in general.

state class to k state classes is $\mathcal{O}(k \times (|\twoheadrightarrow| + |\Omega|))$. Assume in the unobservable SCG that every observable transition can be fired from every state class. The steps 3 to 12 will be executed $|T_o|$ times for each state class and the complexity is $\mathcal{O}(|T_o| \times |\twoheadrightarrow| \times (|\twoheadrightarrow| + |\Omega|))$. In this case, the algorithm can be adapted to that it computes paths from the initial state class to all other state classes and it can be done within one breadth-first search at the beginning (between steps 1 and 2). Although this adaptation is faster in this case, it may use more space to store all paths.

The algorithm of diagnosis, Algorithm 5.2, checks all output edges of every consistent state classes in the FDG (set $\mathcal{S}$). Assuming each state class has at most n output edges, the complexity of Algorithm 5.2 is $\mathcal{O}(n \times |\mathcal{S}|)$.

5.7 UPPER BOUND ON THE NUMBER OF CONSISTENT STATES

As discussed in the previous section, the complexity of the diagnosis algorithm depends on the number of consistent states. In this section, we compute an upper bound of the number of consistent states corresponding to the observation. In [53], the computation of an upper bound of the number of consistent markings in untimed PN is proposed. However, it cannot apply directly to TPN.

In the following example, we show the number of consistent states in TPN is not comparable to the one in untimed PN. Let us consider the nets in Figure 5.2 and assume ε_3 is the fault transition. In the net in Figure 5.2(a), if the observation is $w_a = \langle t_1, 1\rangle\langle t_5, 4\rangle$, there are two consistent states in the TPN and one in the untimed PN (shown in Table 5.3). In the other net (Figure 5.2(b)), if $w_b = \langle t_1, 1\rangle\langle t_4, 2\rangle$, in the untimed PN there are more consistent states than the ones in TPN; and if $w_c = \langle t_1, 1\rangle\langle t_4, 3\rangle$, both of TPN and untimed PN have the same number of consistent states.

In [53], it is proved that in untimed PN the number of consistent markings corresponding to an observation is polynomial in the length of the observation. We are going to compute an upper bound on the number of state classes with the same marking, i.e., the maximal k

Table 5.3: Consistent states and markings in TPN and untimed PN of examples in Figure 5.2

net	w	TPN	untimed PN	more states
a	$\langle t_1, 1\rangle\langle t_5, 4\rangle$	$\alpha_a = \langle \mathbf{m}_1 = p_3, 0 \leqslant x_4 \leqslant 2\rangle$, $\alpha'_a = \langle \mathbf{m}_1 = p_3, 0 \leqslant x_4 \leqslant 3\rangle$	$\mathbf{m}_1 = p_3$	TPN
b	$\langle t_1, 1\rangle\langle t_4, 2\rangle$	$\alpha_b = \langle \mathbf{m}_2 = p_4, \emptyset\rangle$	$\mathbf{m}_2 = p_4, \mathbf{m}_3 = p_5$	untimed PN
	$\langle t_1, 1\rangle\langle t_4, 3\rangle$	α_b, $\alpha'_b = \langle \mathbf{m}_3, \emptyset\rangle$	$\mathbf{m}_2$, $\mathbf{m}_3$	equal

such that $\forall i,j \in \{1,\dots,k\}$, $i \neq j$, $\alpha_i = \langle \mathbf{m}, F_i\rangle$, $\alpha_j = \langle \mathbf{m}, F_j\rangle$, $F_i \neq F_j$. Using this upper bound, the result in [53] can be adapted to TPN. We assume that the unobservable subnet $\mathcal{N}_u$ of $\mathcal{N}$ is bounded and the bounds of time intervals are non-negative integer numbers[*].

We define the following notations:

1. $\mathbf{C}_o^u$ is the submatrix of $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$ such that its rows correspond to P_u and columns correspond to T_o;
2. $\mathbf{C}_u^u = \mathbf{Post}_u - \mathbf{Pre}_u$;
3. $\mathbf{y}_u$ is a $|P_u|$-dimensional vector with strictly positive integer entries such that $\mathbf{y}_u^T \mathbf{C}_u^u \leqslant \mathbf{0}$;
4. $I(t) = [I_-(t), I_+(t)]$ such that $I_-(t)$ and $I_+(t)$ are the lower and upper bounds of the time interval associated to t.

Proposition 5.1 ([53]). *In an untimed PN, in which the set of unobservable transitions is T_u, if the length of observation is k, the number of consistent markings is upper bounded by*

$$\binom{|T_u|}{c_1 + c_2 k + |T_u|} \tag{5.5}$$

where $c_1 = \mathbf{y}_u^T \mathbf{m}_0^u$, $\mathbf{m}_0^u$ is the marking of places $p \in P, \exists t \in T_u, t \in {}^\bullet p \cup p^\bullet$ and c_2 is the maximal entry (element) of $\mathbf{y}_u^T \mathbf{C}_o^u$.

Proposition 5.2 ([10]). *Let $\mathcal{N}$ be a TPN, the inequalities in every state class are in the following forms:*

1. *$a_t \leqslant x_t \leqslant b_t$, for all enabled t, indicates that the transition t could be fired in a_t time units and must be fired in no more than b_t time units;*
2. *$x_t - x_{t'} \leqslant c_{tt'}$, for all enabled t and t' such that $t \neq t'$, implies that the time between the firing of t and t' is not larger than $c_{tt'}$ time units.*

Remark 5.3 ([10]). *It is proved that a_t, b_t and $c_{tt'}$ are linear combinations with integer coefficients of $I_-(t)$ and $I_+(t)$. If $\forall t \in T$, $I_-(t), I_+(t) \in \mathbb{N}_{\geqslant 0}$, the constants a_t, b_t and $c_{tt'}$ in every domain are non-negative integer.*

In the sequel, we will show first that the number of state classes, which have the same marking, has an upper bound, and the upper bound is independent of the length of observation (shown in Proposition 5.4). Second, using Proposition 5.1 and Proposition 5.4, we will indicate that the number of consistent state classes is finite corresponding to the length of observation (Corollary 5.5).

Proposition 5.4. *Let $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, I\rangle$ be a TPN and $\mathbf{m}$ be a marking. If the time bounds of the time intervals associated with transitions are non-negative rational numbers, an upper bound of the number of state classes corresponding to $\mathbf{m}$ is $\theta^{|T|^2+|T|}$, where $\theta = \max(\{\delta | t \in En(\mathbf{m}), \delta = I_+(t)\})$*[†].

[*] It can also be applied to the case that the bounds are rational numbers. In this case, the bounds can be multiplied by a number to convert them into integers.

[†] Here, θ is the maximal upper bound of the time intervals of enabled transitions.

Proof. The number of state classes corresponding to one marking equals the number of possible firing domains that could be obtained by combining the inequalities in Proposition 5.2:

1. The form "$a_t \leqslant x_t \leqslant b_t$": Initially, $a_t = I_-(t)$ and $b_t = I_+(t)$. If a transition $t' \neq t$ is fired and t remains enabled in the new state class, a new firing interval is selected for t in the new state class $a'_t \leqslant x_t \leqslant b'_t$ such that $0 \leqslant a'_t \leqslant a_t$ and $0 \leqslant b'_t \leqslant b_t$. Because a_t (a'_t) and b_t (b'_t) are non-negative integers, the number of possible values of them are $I_-(t)+1$ and $I_+(t)+1$, respectively. Therefore, for x_t, the maximal number of inequalities in the form $a_t \leqslant x_t \leqslant b_t$ is $(I_-(t)+1)(I_+(t)+1)$. In a state class whose marking is m, the maximal number of inequalities of this form is:
$$\prod_{t \in En(m)} (I_-(t)+1)(I_+(t)+1).$$
In the worst case, if all transitions are enabled, an upper bound is $\theta^{2|T|}$.

2. The form "$x_t - x_{t'} \leqslant c_{tt'}$": The variable x_t and $x_{t'}$ satisfy $x_t \leqslant I_+(t)$ and $x_{t'} \geqslant 0$. There is $x_t - x_{t'} \leqslant I_+(t) - x_{t'} \leqslant I_+(t)$ and the maximum number of inequalities of this form is:
$$\prod_{t,t' \in En(m), t \neq t'} I_+(t),$$
and an upper bound is $\theta^{|T|^2-|T|}$.

Finally, the maximal number of state classes corresponding to one marking is $\theta^{2|T|}\theta^{|T|^2-|T|} = \theta^{|T|^2+|T|}$. □

The number of consistent state classes is upper bounded by the product of the upper bound of consistent markings, which is polynomial in the length of observation, and the one of state classes that have the same marking.

Corollary 5.5. *Let $\langle \mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle, I \rangle$ be a TPN and m be a marking. If the time bounds of the time intervals associated with transitions are non-negative integer numbers, the upper bound of consistent state classes corresponding to each observation is finite.*

Proof. The proof is trivial. An upper bound can be obtained by multiplying the number of consistent markings Equation 5.5 with the maximal number of state classes corresponding to one marking $\theta^{|T|^2+|T|}$:
$$\binom{|T_u|}{c_1 + c_2 k + |T_u|} \theta^{|T|^2+|T|}.$$
□

6 DECENTRALIZED DIAGNOSIS ALGORITHMS

A system can be large and distributed in several (local) subsystems. In this case, a centralized diagnoser (Chapter 5) is not practical and a decentralized one is convenient. In this chapter, based on fault diagnosis graph (Chapter 4), we propose algorithms of decentralized diagnosis using a coordinated decentralized architecture. According to subsystems, each local diagnoser can observe only a subnet of the observable transitions. They construct local fault diagnosis graphs using local observations and communicate local states to a coordinator. The coordinator is constrained by limited memory and processing capability. It is in charge of computing diagnosis states of the global system using information provided by local diagnosers.

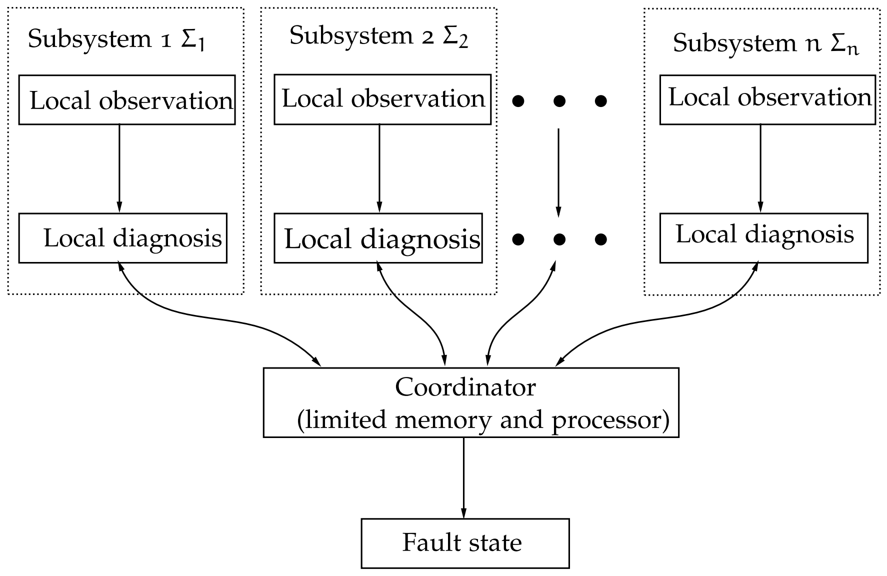


Figure 6.1: A coordinated decentralized architecture

6.1 INTRODUCTION

In this chapter, we propose algorithms and examples on decentralized diagnosis using FDG (Chapter 4).

The decentralized diagnosis uses a coordinator and local diagnosers. The general algorithm for decentralized diagnosis (Section 6.2) is different from the one for centralized diagnosis (Section 5.2). In decentralized diagnosis, first, each local diagnoser initialize its local FDG. Second, they wait for an observation. Third, when a local diagnoser observes the firing of an observable transition, it updates its local FDG if necessary (Section 6.3). Forth, the local diagnoser send information as a message to the coordinator. Last, the coordinator computes diagnosis states (Section 6.4).

6.1.1 *Decentralized diagnosis architecture*

In real world, sensors may be distributed in various locations being impractical to have a centralized diagnoser. Therefore, a decentralized diagnosis approach, in which computation is distributed into local diagnosers, is convenient.

We consider that the global system consists of several local diagnosers. Each one can observe a subset of the observable events (transitions) and computes local diagnosis states. Moreover, there is a global coordinator, which receives messages from local diagnosers. The message contains only brief information about local diagnosis states, which can be transmitted efficiently. The coordinator has only limited computation capacity, and the global diagnosis states must be computed easily using the received messages. The decentralized diagnosis architecture is illustrated in Figure 6.1.

Assuming there are k local diagnosers, we denote the set of observable transitions of the i-th local diagnoser as T_{oi} such that $T_o = \bigcup_{i=1}^{k} T_{oi}$.

Definition 6.1. *For the TPN of a global system* $\langle \mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle, I \rangle$, *the* unobservable subnet of a subsystem Σ_i, *whose subset of observable transition is* T_{oi}, *is* $\langle \mathcal{N}_{ui} = \langle P, T_{ui}, \mathbf{Pre}_{ui}, \mathbf{Post}_{ui} \rangle, I_{ui} \rangle$, *where:*

1. P *is the set of places;*
2. $T_{ui} = T \setminus T_{oi}$ *is the set of transitions which cannot be observed in* Σ_i*;*
3. $\mathbf{Pre}_{ui}$ *and* $\mathbf{Post}_{ui}$ *are pre and post incidence matrices restricted to* T_{ui}*;*
4. $I_{ui} : T_{ui} \rightarrow \mathbb{Q}_0 \times (\mathbb{Q}_0 \cup \{\infty\})$.

■

A subsystem Σ_i is the projection of the global system corresponding to the observable transition of the i-th local diagnoser. The local observation function of Σ_i is $\lambda_i : \sigma \rightarrow T_{oi}^*$. It extracts from σ the sequence of observable transitions that can be observed by Σ_i. The global observable function is $\lambda : \sigma \rightarrow T_o^*$.

The considered decentralized diagnosis architecture follows the assumptions in the sequel:

A1) The unobservable subnets in subsystems are *acyclic*. Does not exist arbitrary long unobservable firing sequence in subsystems;

A2) All messages sent between the coordinator and subsystems are received in the same order in which they are sent;

A3) Every subsystem knows which are observable transitions in the whole system. (Nevertheless, it does not mean that every subsystem can observe all observable transitions.)

A4) The coordinator does not know the structure and the initial state of the system.

6.1.2 *Adaptation of FDG to decentralized diagnosis*

The FDG is optimized for diagnosis on timed systems, and in general it has fewer state classes than the corresponding SCG. Hence, it will be suitable for local diagnosers to store FDG rather than SCG, and then we adapt FDG for decentralized diagnosis.

In decentralized diagnosis, each subsystem contains a subset of observable transitions of the global system, and then the number of state classes in the local FDG is less than the one of the global FDG. Remember that in the FDG the state classes obtained by the firing of unobservable transitions are removed. Let us consider the PN in Figure 6.2(a) under the assumption that $T_o = \{t_1, t_3, t_4, t_5\}$ and $T_f^1 = \{t_2\}$. The full FDG is computed and shown in Figure 6.3(a). It contains three state classes, while in the SCG (given in Figure 6.2(b)) there are

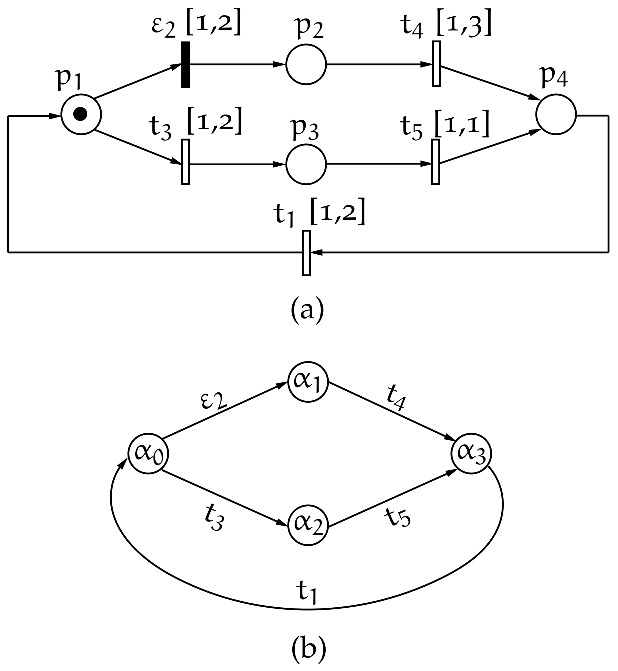
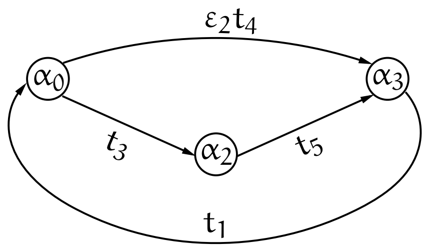


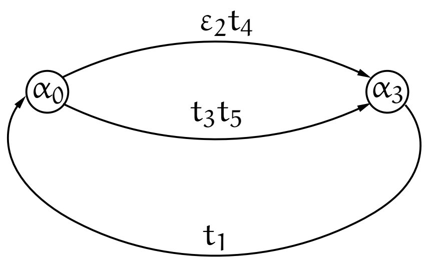
Figure 6.2: (a) A TPN model, with the initial marking $m_0 = p_1$, used to illustrate the computation of the firing domain. (b) The SCG of the TPN system in (a).

Table 6.1: State classes in the SCG in Figure 6.2(b)

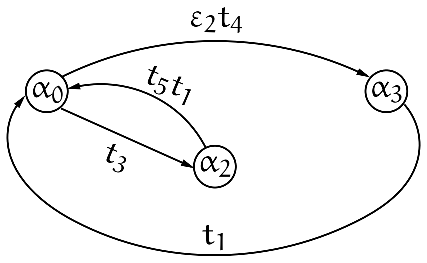
state class	marking	domain
α_0	p_1	$(1 \leqslant x_2 \leqslant 2) \wedge (1 \leqslant x_3 \leqslant 2)$
α_1	p_2	$1 \leqslant x_4 \leqslant 3$
α_2	p_3	$1 \leqslant x_5 \leqslant 1$
α_3	p_4	$1 \leqslant x_1 \leqslant 2$



(a) FDG $\mathcal{G}_1$



(b) FDG $\mathcal{G}_2$



(c) FDG $\mathcal{G}_3$

Figure 6.3: (a) The FDG $\mathcal{G}$ of the net system in Figure 6.2(a) where $T_o = \{t_1, t_3, t_4, t_5\}$. (b) The FDG $\mathcal{G}_1$ in the subsystem Σ_1 where $T_{o1} = \{t_1, t_4, t_5\}$. (c) The FDG $\mathcal{G}_2$ in Σ_2 where $T_{o2} = \{t_1, t_3, t_4\}$.

four. The three state classes are reached by firing the observable transitions in T_o. Assume there are two subsystems Σ_1 and Σ_2, such that in Σ_1, the observable transitions are $T_{o1} = \{t_1, t_4, t_5\}$, while in Σ_2, the observable transitions are $T_{o2} = \{t_1, t_3, t_4\}$. The corresponding FDG are computed using the same approach as in centralized case. They are illustrated in Figure 6.3(b) and (c).

Having now only partial information, some information is lost. One part of such information is the firing orders of the observable transitions. Let us consider a firing sequence of the global system $\sigma = \sigma_{u1}t_1\sigma_{u2}t_2\sigma_{u3}t_3$. Assume a subsystem Σ_i in which only t_3 is observable, while t_1 and t_2 are observable but by other subsystems. In the local FDG of Σ_i, there is an edge $\alpha_0 \to \alpha_1$ with the label $\langle t_3, I_\sigma, D_\sigma \rangle$, where:

1. t_3 is the observed transition,
2. I_σ is the firing domain of σ, and
3. D_σ records the firing of fault transitions in σ.

Because Σ_i knows t_1 and t_2 are observable by other subsystems, this knowledge can be used in the construction of local FDG and diagnosis. In order to preserve this information, we replace the first element in the label, the observed transition, with a sequence of observable transitions. In the case of σ, the associated sequence is $\lambda(\sigma) = t_1t_2t_3$. The last transition in the sequence is the one that can be observed in the subsystem Σ_3. The length of $\lambda(\sigma)$ depends on σ. Because we assume there is no arbitrary long unobservable firing sequence in any subsystem, the number of transitions in $\lambda(\sigma)$ is finite.

The other part of information, which is lost, consists of the firing of unobservable transitions at a state class. Let us consider again the firing sequence $\sigma = \sigma_{u1}t_1\sigma_{u2}t_2\sigma_{u3}t_3$, but assume now that the local observable transitions are t_1 and t_3. In the global system, where t_1, t_2 and t_3 are observable, there exists a path in the global FDG $\alpha_0 \xrightarrow{\sigma_{u1}t_1} \alpha_1 \xrightarrow{\sigma_{u2}t_2} \alpha_2 \xrightarrow{\sigma_{u3}t_3} \alpha_3$. Assuming t_1 is observed, α_1 is a consistent state class, and the label associated with the edge from α_1 to α_2 will be used in diagnosis. However, in the local FDG, because t_2 cannot be observed, the path becomes $\alpha_0 \xrightarrow{\sigma_{u1}t_1} \alpha_1 \xrightarrow{\sigma_{u2}t_2\sigma_{u3}t_3} \alpha_3$, where $\sigma_{u2}t_2\sigma_{u3}$ becomes an unobservable firing sequence. It is helpful to keep the information that t_2 can be observed by other subsystems. Using this information, we can partition the firing sequence $\sigma_{u2}t_2\sigma_{u3}t_3$ into $\sigma_{u2}t_2$ and $\sigma_{u3}t_3$. Since we are interested in the firing of fault transitions, we store the diagnosis information of $\sigma_{u2}t_2$ in the label of the edge $\alpha_1 \to \alpha_3$.

Since more information is necessary to be kept in the decentralized case, we extend the definition of the FDG to the following one.

Definition 6.2. *A* fault diagnosis graph of a subsystem Σ_i *is a triple* $\mathcal{G}_i = \langle \Omega, \twoheadrightarrow, \alpha_0, \mathcal{L}_v, \mathcal{L}_e \rangle$, *where:*

1. $\alpha_0 = \langle \mathbf{m}_0, F_0 \rangle$ *is the initial state,*

2. *$\twoheadrightarrow$ is the set of edges, such that $\alpha \twoheadrightarrow \alpha'$ means $\exists \sigma_u \in T_u^*, t_o \in T_{oi}$ and α' is reachable from α by firing $\sigma_u t_o$,*

3. *$\Omega = \{\alpha | \alpha_0 \twoheadrightarrow^* \alpha\}$, where $\twoheadrightarrow^*$ is the reflexive and transitive closure of $\twoheadrightarrow$,*

4. *$\mathcal{L}_v : \Omega \to \{N, F, U\}^r$ is a labeling function that associates diagnosis labels to nodes, where r is the number of fault classes,*

5. *$\mathcal{L}_e : \twoheadrightarrow \to T_o^* \times \{\mathbb{Q}_{\geqslant 0} \times (\mathbb{Q}_{\geqslant 0} \cup \{\infty\})\} \times \{N, F\}^*$ is a labeling function of edges, i.e., for an edge e corresponding to $\alpha \xrightarrow{\sigma_u t_o} \alpha'$, the label of e is $\mathcal{L}_e(e) = \langle \lambda(\sigma_u t_o), I_{\sigma_u t_o}, D_{\sigma_u t_o} \rangle$.*

■

6.2 GENERAL ALGORITHM

The general Algorithm 6.1 briefly shows how the diagnosis works. The idea is that, when a transition is observed in a subsystem, first, the subsystem updates its FDG and the set of consistent states (by using Algorithm 6.2); second, it sends the observed transition and consistent states to the coordinator; third, the coordinator computes the diagnosis state using the observed transition and consistent states sent by subsystems (by applying Algorithm 6.3).

In Algorithm 6.1, when the system starts evolving, the coordinator is initialized (step 1). Assuming there are k subsystems, the coordinator keeps for each one the partial observed words $w_i, i = 1, \ldots, k$ from the last communication from the subsystem to the present time. The local FDGs and sets of consistent states of subsystems are also initialized at the same time (steps 2 to 6). The set W_i contains unexplored states in the local FDG and initially it contains output states of the initial state (step 4). The set of consistent states $\mathcal{S}$ contains elements as $\langle \alpha, \overrightarrow{\sigma_\alpha}, w_\alpha \rangle$, where:

1. α is a reachable state, e.g., assuming there is a path $\alpha_0 \xrightarrow{\sigma_{u1} t_1} \alpha_1 \xrightarrow{\sigma_{u2} t_2} \alpha_2 \xrightarrow{\sigma_{u3} t_3} \alpha_3$ and $t_1 t_2 t_3$ is observed, then a reachable state is α_3;

2. $\overrightarrow{\sigma_\alpha}$ is the path vector from the initial state to α, e.g., in the above mentioned path, the path vector to α_3 is $\overrightarrow{\sigma_{u1} t_1} + \overrightarrow{\sigma_{u2} t_2} + \overrightarrow{\sigma_{u3} t_3}$;

3. w_α is the list of observable transitions, e.g., considering the path above, since the input edge of α_3 is $\sigma_{u3} t_3$, then w_{α_3} is t_3.

When a new observation comes at subsystem Σ_i, the local FDG is updated if there exists unexplored vertices in it (step 8). For each reachable state α in $\mathcal{S}$, if in the local FDG it has an output edge, denoted as $\langle \alpha, \langle \overrightarrow{\sigma t_j}, I, w_e \rangle, \alpha' \rangle$, from α to α' and labeled with $\langle \overrightarrow{\sigma t_j}, I, w_e \rangle$ that, in the firing count vector $\boldsymbol{\sigma}$, the firing count of the observed transition is 1 and the output state α' is not explored (step 9), then the local FDG and W are updated (steps 10 to 13). Considering an element $\langle \alpha, \overrightarrow{\sigma_\alpha}, w_\alpha \rangle \in \mathcal{S}_i$, if α has an output edge $\langle \alpha, \langle \overrightarrow{\sigma t_j}, I, w_e \rangle, \alpha' \rangle$ and t_j can be fired at τ_j according to I , then a new element $\langle \alpha', \overrightarrow{\sigma_\alpha} + \overrightarrow{\sigma t_j}, w_e \rangle$ is inserted into the updated set of consistent states corresponding to the observation that t_j is observed at τ_j (step 15). After updating, the subsystem sends the observed transition t_j and the following information V_i to the coordinator (step 16). The information V_i contains elements in the form $\langle \overrightarrow{\sigma_\alpha}, w_\alpha, D_\alpha \rangle$:

1. path vector $\overrightarrow{\sigma_\alpha}$ of a consistent state $\langle \alpha, \overrightarrow{\sigma_\alpha}, w_e \rangle$, which is used for detecting fault occurrence before t_j is observed,

2. the list of observable transitions w_α for the verification if α is reachable or not based on global observation,

3. the diagnosis label D_α of α.

The coordinator uses V_i and t_j for diagnosis and send the partial observed word w_{li} to Σ_i (step 17). Elements satisfying $\langle \alpha, \overrightarrow{\sigma_\alpha}, w_\alpha \rangle$ that $w_\alpha \neq w_{li}$ are removed from $\mathcal{S}_i$ (step 18).

Algorithm 6.1 General Fault Diagnosis Algorithm

```
1:  $W_{co} := \{w_i | w_i = \epsilon, i = 1, \dots, k\}$                ▷ the coordinator
2:  for each $\Sigma_i, i = 1, \dots, k$ do
3:    compute the unobservable $\mathcal{G}_{scg}$ based on $T_{oi}$
                          ▷ $T_{oi}$ is the set of observable transitions in $\Sigma_i$
4:    $[\mathcal{G}_i, W_i] := FDG_d(\mathcal{N}, \emptyset, \mathcal{G}_{scg}, \emptyset, T_{oi}, T_o)$
                                    ▷ initialize the FDG of each subsystem
5:    $\mathcal{S}_i := \{\langle \alpha_0, 0, \epsilon \rangle\}$
                                    ▷ initialize the set of consistent states
6:  end for
7:  let $t_j$ be a new observation at $\tau_j$ at $\Sigma_i$
8:  if $W_i \neq \emptyset$ then
9:    for each $\alpha \in \mathcal{S}_i$ s.t. $\exists \langle \alpha, \langle \overrightarrow{\sigma t_j}, I, w_e \rangle, \alpha' \rangle \in \mathcal{G}_i$
          $\wedge \alpha' \in W_i$ do
10:     compute the SCG $\mathcal{G}_{scg}$ of the unobservable subnet based on
          $T_{oi}$ with the initial state $\alpha'$
11:     $[\mathcal{G}_i, W_i] := FDG_d(\mathcal{N}, \mathcal{G}_i, \mathcal{G}_{scg}, W_i, T_{oi}, T_o)$
                                          ▷ update $\mathcal{G}_i$ by using $\mathcal{G}_{scg}$
12:     $W_i := W_i \setminus \{\alpha'\}$                  ▷ remove explored $\alpha'$ from $W_i$
13:   end for
14: end if
15: update $\mathcal{S}_i$ according to $t_j$ and $\tau$
16: compute $V_i$ from $\mathcal{S}_i$ and $\mathcal{G}_i$
                     ▷ compute the information for sending to the coordinator
17: $[w_{li}, W_{co}, D] := coordinator(W_{co}, V_i, t_j)$
18: remove states inconsistent with $w_{li}$ from $\mathcal{S}_i$
19: go to 7
```

6.3 UPDATE THE FDG IN THE SUBSYSTEMS

In a subsystem, the local FDG is updated using an SCG, where its initial state is a state in the FDG. The SCG is obtained by firing only transitions which cannot be observed in the subsystem. If a state in the SCG enables a transition that can be observed in the subsystem, then the successor state is computed by firing the observable transition at the state and a new node representing the successor state is inserted into the FDG. Paths in the SCG from the initial state of the SCG to the successor state are found and edges labeled with the paths are added into the FDG.

The local FDG of a subsystem is constructed using Algorithm 6.2. For every state α_u in the SCG $\mathcal{G}_{scg}$, such that there exists an observable transition t_j that can be fired at α_u (step 4), the successor state α (of α_u) is computed by firing t_j (step 5). If α does not belong to the FDG, i.e., α is a newly obtained state and it has not been explored, then α is added to the set of unexplored states W (step 7). All paths from the initial state α_0 of $\mathcal{G}_{scg}$ to α_u are found in $\mathcal{G}_{scg}$ (step 9), and then for each path its firing domain I is computed (step 11). The list of observable transitions of the edge is computed as w (step 12), and the FDG is updated with the state α as a node and an edge labeled with $\langle \alpha_0, \langle \overrightarrow{\sigma t}, I, w \rangle, \alpha \rangle$ (step 13).

The fault occurrence in the unobservable firing sequences starting at α_0 is detected in two steps and stored as the label of α_0. The first step consists in finding the unobservable firing sequences from α_0 to every node (step 14). In the second step, the unobservable firing sequences are used to update the label of the node of α_0 (step 15). Considering an unobservable firing sequence σ_u and a fault class T_f^i, if $\exists t \in T_f^i$ and t is fired in σ_u, then we label σ_u with $\langle i, F \rangle$; otherwise, the label is $\langle i, N \rangle$. If D_{α_0} is emptied, then the label of σ_u is added into D_{α_0}. If the diagnosis state of T_f^i in D_{α_0} is the same as the label associated to σ_u, then D_{α_0} remains unchanged; otherwise, the diagnosis state of T_f^i in D_{α_0} will be changed to U.

6.4 THE COORDINATOR

6.4.1 *Coordinator design*

In decentralized diagnosis, the diagnosis states are computed by the coordinator using *messages* sent from local diagnosers. The tasks of local diagnosers consist of:

1. observing the firings of observable transitions,
2. constructing local FDG, and
3. reporting local states to the coordinator.

Only local diagnosers that observe firings of observable transitions perform the second and third tasks. The tasks of the coordinator include:

1. computing the diagnosis states, and
2. sending to each diagnoser, which reported its local states, messages so that they can update local consistent states.

Algorithm 6.2 $[\mathcal{G}, W] = \mathrm{FDG}_d(\mathcal{N}, \mathcal{G}, \mathcal{G}_{scg}, W, T_o^i, T_o)$

```
1: $\alpha_0$ := the initial state in $\mathcal{G}_{scg}$
2: $D_{\alpha_0} := \emptyset$                                    ▷ initialize the label of $\alpha_0$
3: for each $\alpha_u = \langle m_u, F_u \rangle \in \mathcal{G}_{scg}$ do
4:   for each $t \in En(m_u) \cap T_o$ s.t. isFireable$(\alpha_u, t)$ do
5:     $\alpha = succ(\alpha_u, t)$                                ▷ compute the successor
6:     if $\alpha \notin \mathcal{G}$ then                                  ▷ if $\alpha$ is not explored
7:       $W = W \cup \{\alpha\}$                                       ▷ update $W$
8:     end if
9:     $\mathcal{P}$ := the paths from $\alpha_0$ to $\alpha_u$ in $\mathcal{G}_{scg}$
10:    for each $\sigma \in \mathcal{P}$ do
11:      $I := domain(\mathcal{G}_{scg}, \sigma t, \alpha_0)$                  ▷ the firing domain of $\sigma t$
12:      $w = \lambda_i(\sigma t)$ based on $T_o$
13:      add $\alpha$ and $\langle \alpha_0, \langle \overrightarrow{\sigma t}, I, w \rangle, \alpha \rangle$ to $\mathcal{G}$
14:      divide $\sigma$ into $\sigma = \sigma_u t'_o \sigma'$
15:      update $D_{\alpha_0}$ using $\sigma_u$
16:    end for
17:  end for
18: end for
19: associate $D_{\alpha_0}$ with $\alpha_0$ that $\mathcal{L}_v(\alpha_0) = D_{\alpha_0}$
20: return $\mathcal{G}, W$
```

We assume a limited computing capacity for the coordinator including processors, memory and so on. Therefore, we must ensure that:

1. the coordinator does not save FDG,
2. the computation of diagnosis states should be simple.

The coordinator receives messages from local diagnosers. Let us discuss first the information available in local diagnosers:

1. the observed transition;
2. the diagnosis information of the paths from the initial state class to the consistent ones, and the information is represented using a vector whose length equals to the number of fault classes;
3. the partial observed word that is a list representing the observed transitions in their observed order since the last communication with the coordinator;
4. the consistent state classes corresponding to the observed transition;
5. the consistent classes corresponding to the observed transition;
6. the diagnosis labels associated with these consistent state classes;
7. the local FDG.

Among these information, 1), 4), 6) and 7) are needed in local diagnoers. The coordinator needs 2), 3) and 5). The reasons are in the sequel. The observed transition is the last transition in the partial observed word, so it is not necessary to be sent. Because the computation of diagnosis states focuses on paths (not in which state classes the system is), the consistent state classes are not used in diagnosis and will not be sent. With these information, the coordinator does not need local FDG for diagnosis. Moreover, it is too large to be sent to the coordinator.

The coordinator keeps in its memory a partial observed word for each diagnoser. The observed word starts from the last communication with the diagnoser until the current observation.

6.4.2 *Algorithms*

The coordinator keeps, for each subsystem, a partial observed word to check if the consistent states reported by a subsystem are consistent with the global observation or not. Because a subsystem does not observe all observable transitions, a consistent state in a subsystem may not be consistent in the global system. In the diagnosis, the coordinator first removes inconsistent states and second finds the occurrence of fault events corresponding to each consistent state, i.e., every fault transition is fired or not in the path from the initial state to the consistent state and the unobservable firing sequences starting at the consistent state. Last, the diagnosis state is computed using the occurrence of fault events in the first step.

In Algorithm 6.3, the partial observed words $w_i, i = 1, \dots, n$ of all subsystems are updated using t_o in the received information (steps 1 to 3). The observed word of the subsystem which reports an observation will be set to the empty word after the diagnosis in the coordinator. The coordinator uses the partial observation w_i and the list of observed transitions to check the consistence of each element in the received information V_i (steps 4 to 8). After removing from V_i the inconsistent elements, the system is diagnosed using V_i (steps 9 to 24). For the i-th fault class, firstly, each element in the received message is checked (steps 10 to 16). If there is a transition in the fault class fired in σ, then this element is labeled as fault of the corresponding fault class (step 12). Otherwise, the element is labeled as D_l, which represents the fault transitions are fired or not in the unobservable firing sequences after the observation (step 14). After all elements have been processed, the coordinator computes the diagnosis state of the global system. If all elements are labeled as N, then the fault has not occurred (step 17) and in the global diagnosis state D, the diagnosis state of T_f^i is N (step 18). The global diagnosis state of T_f^i is F, if every element is labeled as F (step 19). Otherwise, fault may occur, and the diagnosis state is U (step 22). The variable w_i will be sent to the subsystem Σ_i as w_{li} (step 25), and w_i is set to the empty word (step 26).

Algorithm 6.3 $[w_{li}, W_{co}, D] = \text{coordinator}(W_{co}, V_i, t_o)$

```
1: for each $w_i \in W_{co}, i = 1, \ldots, k$ do
                                        ▷ assume there are $k$ subsystems
2:   $w_i := w_i t_o$
3: end for
4: for each $\langle \boldsymbol{\sigma}, w_l, D_l \rangle \in V_i$ do
5:   if $w_l \neq w_i$ then             ▷ $w_i$ is the $i$-th element in $W_{co}$
6:     $V_i := V_i \setminus \{\langle \boldsymbol{\sigma}, w_l, D_l \rangle\}$
7:   end if
8: end for
9: for each $T_f^i, i = 1, \ldots, r$ do
10:   for each $\langle \boldsymbol{\sigma}, w_l, D_l \rangle \in V_i$ do
11:     if $\exists t_f \in T_f^i, t_f \in \sigma$ then
12:       $T_f^i$ is labeled as fault of $\langle \boldsymbol{\sigma}, w_l, D_l \rangle$
13:     else
14:       $T_f^i$ is labeled as the corresponding
            diagnosis state in $D_l$
15:     end if
16:   end for
17:   if $\forall \langle \boldsymbol{\sigma}, w_l, D_l \rangle \in V_i$, the labels of $T_f^i$ are normal then
18:     The diagnosis state of $T_f^i$ in $D$ is normal.
19:   else if $\forall \langle \boldsymbol{\sigma}, w_l, D_l \rangle \in V_i$, the labels of $T_f^i$ are fault then
20:     The diagnosis state of $T_f^i$ in $D$ is fault.
21:   else
22:     The diagnosis state of $T_f^i$ in $D$ is uncertain.
23:   end if
24: end for
25: $w_{li} := w_i$
26: $w_i := \epsilon$
```

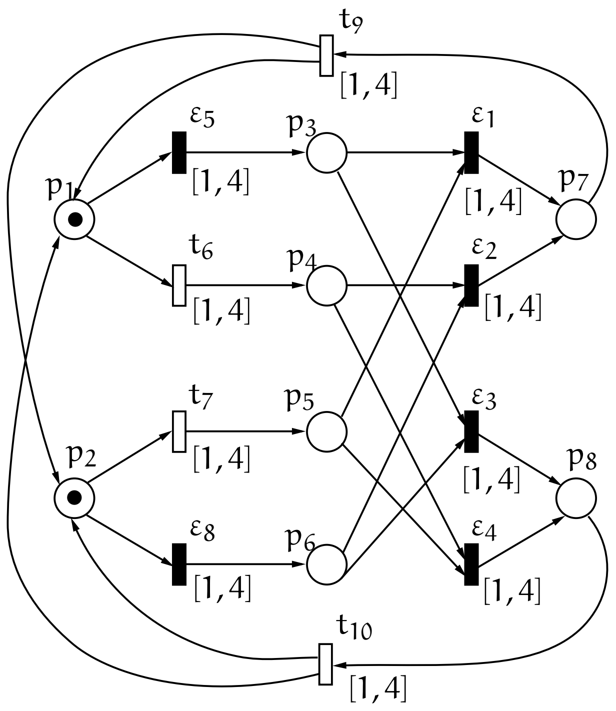


Figure 6.4: A TPN containing two subsystems

6.5 EXAMPLE

Let us consider the TPN system in Figure 6.4 with $m_0 = [1\ 1\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ 0]^T$. The set of observable transitions is $T_o = \{t_6, t_7, t_9, t_{10}\}$. There are two subsystems Σ_1 and Σ_2. The sets of observable transitions of Σ_1 and Σ_2 are $T_{o1} = \{t_6, t_9, t_{10}\}$ and $T_{o2} = \{t_7, t_9, t_{10}\}$, respectively. Time duration of every transition is $[1, 4]$. There is one fault class T_f^1 containing transition ε_2. The initial state is $\alpha_1 = \langle m_0, (1 \leqslant x_5 \leqslant 4) \wedge (1 \leqslant x_6 \leqslant 4) \wedge (1 \leqslant x_7 \leqslant 4) \wedge (1 \leqslant x_8 \leqslant 4)\rangle$. Let us assume the observed transition is $t_6(2)$, which means t_6 is observed at time 2.

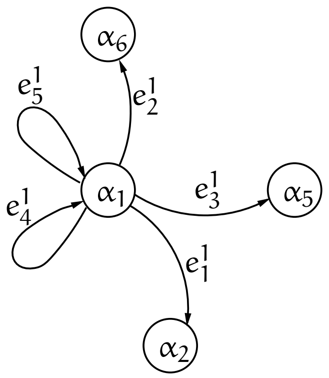
In the diagnosis corresponding to the empty observation, the coordinator and two subsystems initialize their variables. Since there are

Table 6.2: States in local FDGs in Figure 6.5

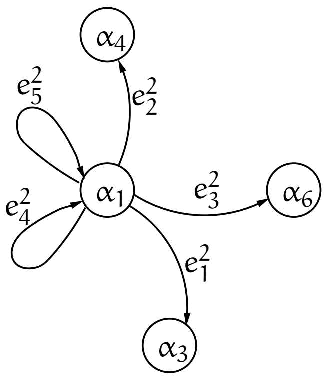
	m	F	D_α
α_1	$[1\ 1\ 0\ 0\ 0\ 0\ 0\ 0]^T$	$1 \leqslant x_5 \leqslant 4 \wedge 1 \leqslant x_6 \leqslant 4 \wedge 1 \leqslant x_7 \leqslant 4 \wedge 1 \leqslant x_8 \leqslant 4$	N
α_2	$[0\ 1\ 0\ 1\ 0\ 0\ 0\ 0]^T$	$0 \leqslant x_7 \leqslant 3 \wedge 0 \leqslant x_8 \leqslant 3 \wedge -3 \leqslant x_7 - x_8 \leqslant 3$	U
α_3	$[1\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$	$0 \leqslant x_5 \leqslant 3 \wedge 0 \leqslant x_6 \leqslant 3 \wedge -3 \leqslant x_5 - x_6 \leqslant 3$	N
α_4	$[0\ 0\ 1\ 0\ 1\ 0\ 0\ 0]^T$	$1 \leqslant x_1 \leqslant 4$	N
α_5	$[0\ 0\ 0\ 1\ 0\ 1\ 0\ 0]^T$	$1 \leqslant x_2 \leqslant 4$	N
α_6	$[0\ 0\ 0\ 1\ 1\ 0\ 0\ 0]^T$	$1 \leqslant x_4 \leqslant 4$	N

Table 6.3: Edges in local FDGs Figure 6.5

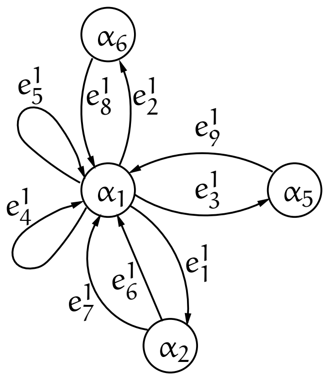
	σ	I	w_e
e_1^1	[0 0 0 0 0 1 0 0 0 0]	[1, 4]	t_6
e_2^1	[0 0 0 0 0 1 1 0 0 0]	[1, 4]	t_7t_6
e_3^1	[0 0 0 0 0 1 0 1 0 0]	[1, 4]	t_6
e_4^1	[1 0 0 0 1 0 1 0 1 0]	[3, 12]	t_7t_9
e_5^1	[0 0 1 0 1 0 0 1 0 1]	[3, 12]	t_{10}
e_6^1	[0 0 0 1 0 0 1 0 0 1]	[2, 11]	t_7t_{10}
e_7^1	[0 1 0 0 0 0 0 1 1 0]	[2, 11]	t_8
e_8^1	[0 0 0 1 0 0 0 0 0 1]	[2, 8]	t_{10}
e_9^1	[0 1 0 0 0 0 0 0 1 0]	[2, 8]	t_9
e_1^2	[0 0 0 0 0 0 1 0 0 0]	[1, 4]	t_7
e_2^2	[0 0 0 0 1 0 1 0 0 0]	[1, 4]	t_7
e_3^2	[0 0 0 0 0 1 1 0 0 0]	[1, 4]	t_6t_7
e_4^2	[0 0 1 0 1 0 0 1 0 1]	[3, 12]	t_{10}
e_5^2	[0 1 0 0 0 1 0 1 1 0]	[3, 12]	t_6t_9
e_6^2	[1 0 0 0 1 0 0 0 1 0]	[2, 11]	t_9
e_7^2	[0 0 0 1 0 1 0 0 0 1]	[2, 11]	t_6t_{10}
e_8^2	[1 0 0 0 0 0 0 0 1 0]	[2, 8]	t_9
e_9^2	[0 0 0 1 0 0 0 0 0 1]	[2, 8]	t_{10}



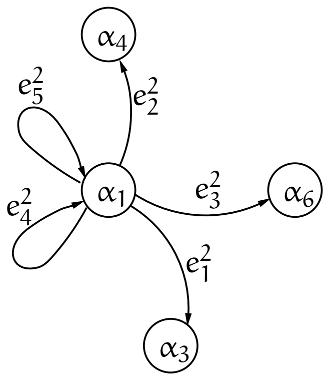
(a) Σ_1 ($w = \epsilon$)



(b) Σ_2 ($w = \epsilon$)



(c) Σ_1 ($w = t_6$)



(d) Σ_2 ($w = t_6$)

Figure 6.5: Local FDGs

two subsystems, the coordinator initializes $W_{co} = \{w_1 = \epsilon, w_2 = \epsilon\}$. The local FDGs of two subsystems are initialized as shown in Figure 6.5(a) and (b). Details of vertices and edges are shown in Table 6.2 and Table 6.3, respectively. In order to explain clearly, if two states in different local FDGs are the same, we give them the same name, e.g., the initial state in two FDGs are the same state and they are denoted as α_1. In the initialization of local FDGs, the sets of unexplored states of Σ_1 and Σ_2 are $W_1 = \{\alpha_2, \alpha_5, \alpha_6\}$ and $W_2 = \{\alpha_3, \alpha_4, \alpha_6\}$, respectively, and the set of consistent states of Σ_1 is $\mathcal{S}_1 = \{\langle \alpha_1, 0, \epsilon \rangle\}$ while for Σ_2 the set is $\mathcal{S}_2 = \{\langle \alpha_1, 0, \epsilon \rangle\}$.

In the diagnosis corresponding to the empty word, two subsystems send messages to the coordinator. Considering Σ_1 sends a message to the coordinator, the message is ϵ, which is the empty observation, and $V_1 = \{\langle 0, \epsilon, N \rangle\}$, where 0 and ϵ are from the element $\langle \alpha_1, 0, \epsilon \rangle$ in $\mathcal{S}_1$ and N, which is the label D_α of α_1 in the FDG of Σ_1 and it means that ε_2 is not fired in the unobservable firing sequences starting at α_1. When receives the information V_1, the coordinator compares the vector 0 in V_1 with w_1 in W_{co}. The information V_1 is consistent with w_i, then the diagnosis state is computed using the message. Because the path vector is 0, which means no fault transition is fired in the path, and in the information the label of fault information is normal, so the diagnosis state of T_f^1 is N.

After the diagnosis, the coordinator sends w_1 to Σ_1 and sets w_1 to the empty word. In Σ_1, element in $\mathcal{S}_1$ is compared with w_1. The received w_1 is ϵ and a list of observable transition in the element $\langle \alpha_1, \mathbf{0}, \epsilon \rangle$ is also the empty word. The communication between Σ_2 and the coordinator is similar to the one between Σ_1 and the coordinator. After the communication, there are:

1. in the coordinator is $W_{co} = \{w_1 = \epsilon, w_2 = \epsilon\}$,
2. in Σ_1 and Σ_2, the sets of unexplored states are $W_1 = \{\alpha_2, \alpha_5, \alpha_6\}$ and $W_2 = \{\alpha_3, \alpha_4, \alpha_6\}$, respectively,
3. the sets of consistent states in Σ_1 and Σ_2 are $\mathcal{S}_1 = \{\langle \alpha_1, \mathbf{0}, \epsilon \rangle\}$ and $\mathcal{S}_2 = \{\langle \alpha_1, \mathbf{0}, \epsilon \rangle\}$, respectively.

The observation is t_6 at time 2 in Σ_1. In the local FDG of Σ_1, from the consistent state α_1, three states α_2, α_5 and α_6 can be reached by observing t_6. Since the three states have not been explored, $W_1 = \{\alpha_2, \alpha_5, \alpha_6\}$, its FDG is updated as shown in Figure 6.5(c). After the update, three states are removed from W_1 and no state is inserted into W_1, then W_1 becomes an empty set. It means the local FDG of Σ_1 has been obtained completely and in further diagnosis, it will not be updated. The consistent states corresponding to the observed word are found in the local FDG and $\mathcal{S}_1$ is constructed as $\{\langle \alpha_2, [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0], t_6 \rangle, \langle \alpha_5, [0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0], t_6 \rangle, \langle \alpha_6, [0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0], t_7 t_6 \rangle\}$. Take the last element in $\mathcal{S}_1$ as an example. The state α_6 is reached from α_1 by the edge e_2^1 and the path vector is obtained by adding $\mathbf{0}$, which is the path vector of α_1 in previous set of consistent state, and $[0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0]$, which is the firing count vector of the edge e_2^1. The sequence $t_7 t_6$ comes from the label of e_2^1. The information sent to the coordinator contains the observed transition t_6 and $V_1 = \{\langle [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0], t_6, U \rangle, \langle [0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0], t_6, U \rangle, \langle [0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0], t_7 t_6, N \rangle\}$. Considering the first element $\langle [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0], t_6, U \rangle$, the vector $[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]$ and t_6 are from the first element in $\mathcal{S}_1$ while the diagnosis information U is the label of α_2 in the local FDG. The message sent to the coordinator is V_1 and t_6, the observed transition. In the coordinator, when the message is received, first the set W_{co} is updated using the observed transition t_6 to $W_{co} = \{w_1 = t_6, w_2 = t_6\}$. Using the updated W_{co}, the coordinator checks the consistency of elements in V_1. Because there are t_6 in the first and second elements in V_1, so they are consistent with w_1 in W_{co}. In the third element in V_1, there is $t_7 t_6$ and it means the global observation generated by this path is $t_7 t_6$. Since w_1 is t_6, so the element is not consistent with w_1 and it is eliminated from V_1 and V_1 becomes $V_1 = \{\langle [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0], t_6, U \rangle, \langle [0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0], t_6, N \rangle\}$. The diagnosis is conducted using the reduced V_1 as following:

1. the first element is labeled as uncertain since the diagnosis label in it is uncertain, which means the fault transition ε_2 may be fired in unobservable firing sequences after the observation;
2. the second element is labeled as uncertain;
3. because the two elements show the fault transition is fired in some cases and not fired in others, then the global diagnosis state is U of T_f^1.

After the diagnosis, the coordinator sends $w_1 = t_6$ to Σ_1 and set $w_1 = \epsilon$. The subsystem Σ_1 uses received $w_1 = t_6$ to reorganize $\mathcal{S}_1$ and the element $\langle \alpha_6, [0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0], t_7 t_6 \rangle\}$ is removed from $\mathcal{S}_1$. Finally, the variables in the coordinator and two subsystems are:

1. in the coordinator, $W_{co} = \{w_1 = \epsilon, w_2 = t_6\}$;
2. in Σ_1, $\mathcal{S}_1 = \{\langle \alpha_2, [0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0], t_6 \rangle, \langle \alpha_5, [0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0], t_6 \rangle\}$ and $W_1 = \emptyset$;
3. in Σ_2, $\mathcal{S}_2 = \{\langle \alpha_1, \mathbf{0}, \epsilon \rangle\}$ and $W_2 = \{\alpha_3, \alpha_4, \alpha_6\}$.

7 CASE STUDY

We apply fault diagnosis graph (Chapter 4) based centralized (Chapter 5) and decentralized diagnosis (Chapter 6) to three cases. There are two systems used in the case study for centralized diagnosis. One first system is a flexible manufacturing system. It contains four robot arms and two vehicles. The second case is a semiconductor production system. We compare our approach with an untimed one using these systems. The third case study focus on a robot motion system. It is represented using a TPN with two processes. The decentralized diagnosis is illustrated with detailed analysis. The diagnosis package is implemented in SimHPN, which can be downloaded from https://webdiis.unizar.es/GISED/.

7.1 INTRODUCTION

We apply the algorithms of centralized diagnosis (Chapter 5) to the example in Section 7.2 and use decentralized diagnosis algorithms (Chapter 6) to the system in Section 7.3.

In the case on centralized diagnosis, our algorithms are applied to a flexible manufacturing with an untimed approach. A numerical comparison shows that: first, our approach can be as fast as the untimed one; second, because timing information is not taken into account by the untimed approach, false diagnosis states may appear, but ours do not have this problem. The case study on the decentralized system illustrates how the algorithms work, including: how local diagnosers construct local fault diagnosis graph and extract information from these graphs for sending to the coordinator; and how the coordinator computes diagnosis states and helps local diagnosers to improve their local fault diagnosis graph.

7.2 CENTRALIZED DIAGNOSIS

7.2.1 *A flexible manufacturing system*

In this section, we apply the proposed approach to a manufacturing system. Let us consider the automated manufacturing system whose layout is sketched in Figure 7.1 and its PN model is shown in Figure 7.2 [44]. Our approach is compared with the one in [15, 51], which focuses on untimed PN, in terms of computational complexity. When the firing of an observable transition is observed, we look in the FDG for the part necessary for the diagnosis. If the part has already been constructed, the algorithm passes to the next step. In this case, the FDG based approach is as fast as the untimed one. The numerical experiment is carried out on a Intel Mac with a clock of 2.3 GHz using Matlab with GLPK [45].

Let us consider the automated manufacturing system whose layout is sketched in Figure 7.1 and its PN model is shown in Figure 7.2 [44].

The plant consists of five machines (M1 to M5), four robots (R1 to R4), a finite capacity buffer B, two inputs of raw parts (I1 and I2) of type 1 and type 2, respectively, two automated guided vehicle (AGV) systems (AGV1 and AGV2), and finally two outputs (O1 and O2) for the processed parts. The plant produces two different types of products from two types of raw materials. An unlimited source of raw parts is assumed.

The plant is described using an untimed PN in which there are 35 places and 24 transitions. The marking of place p_{33} is the number of free buffer slots in the cobuffer. As in [30], we assume that the system is controlled by the addition of three monitor places (p_{36}, p_{37}, p_{38}).

Let the observable transitions be t_1 to t_{12} and unobservable transitions be ε_{13} to ε_{24}, which correspond to regular events. The observable transitions are the introduction of parts in one of the two production lines (transitions t_1 and t_{12}), the introduction of parts in

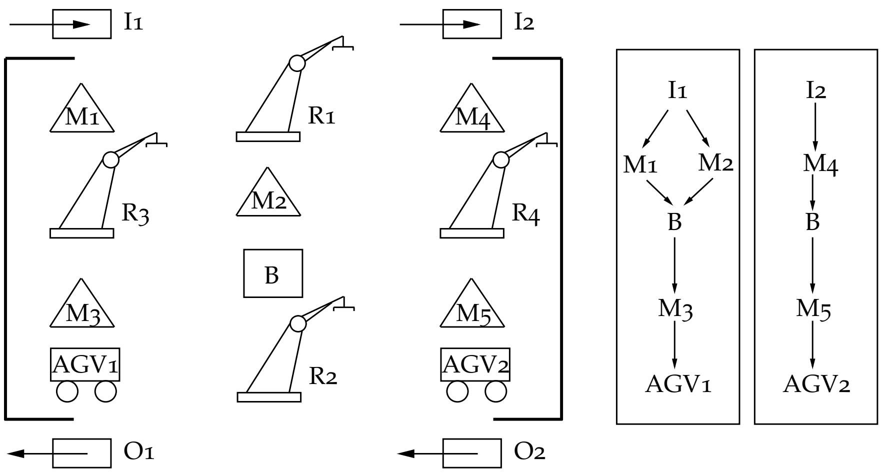


Figure 7.1: An automated manufacturing system

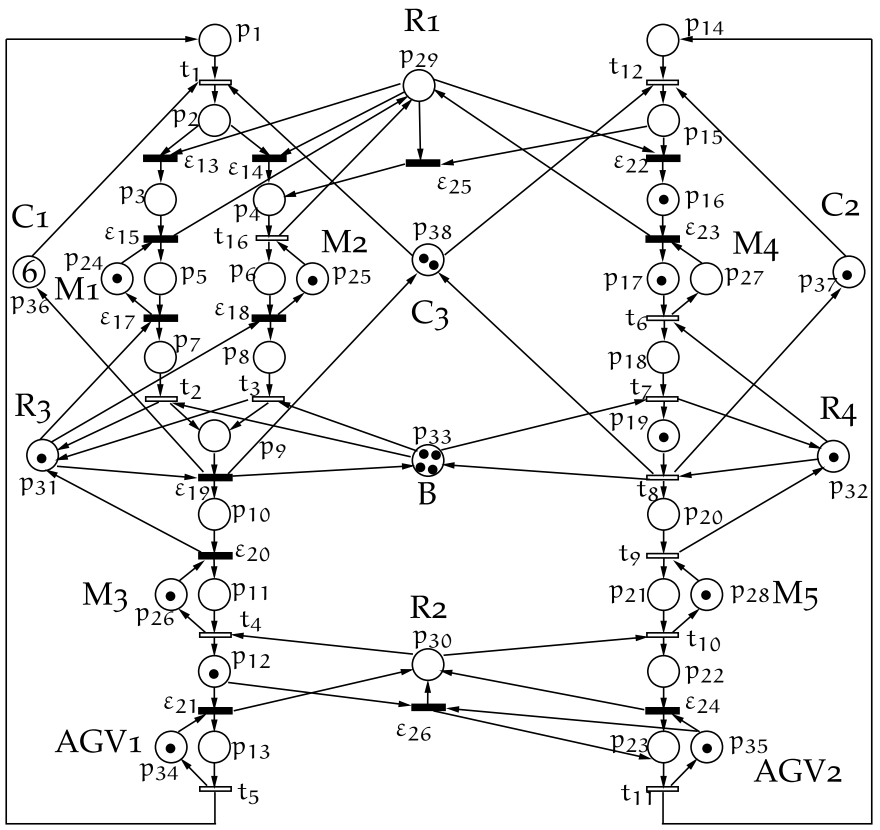


Figure 7.2: PN system of the manufacturing system in Figure 7.1

the buffer by R3 (transitions t_2 and t_3), all operations performed by robot R4 (transitions t_6, t_7, t_8 and t_9), the withdrawal of parts from one of the two production lines by robot R2 (transitions t_4 and t_{10}) and the output parts in the AGV systems AGV1 and AGV2 (transitions t_5 and t_{11}).

Two different types of faults modeled by the unobservable transitions ε_{25} and ε_{26} are considered and we assume two fault classes $T_f^1 = \{\varepsilon_{25}\}$ and $T_f^1 = \{\varepsilon_{26}\}$. The first class of fault corresponds to a malfunctioning of robot R1 that moves one raw part of the second type to the first production line, so that it is processed by machine M2 instead of M4. The second class of fault corresponds to a malfunctioning of robot R2 that moves one part of the first type, after it has been processed by machine M3, and sends it to AGV2 who directs it to the wrong output (O2 instead of O1).

The time intervals of the transitions are: $I(t_1) = [3,8]$, $I(t_2) = [1,8]$, $I(t_3) = [2,6]$, $I(t_4) = [2,7]$, $I(t_5) = [3,5]$, $I(t_6) = [8,8]$, $I(t_7) = [7,8]$, $I(t_8) = [2,5]$, $I(t_9) = [7,7]$, $I(t_{10}) = [5,7]$, $I(t_{11}) = [1,3]$, $I(t_{12}) = [2,2]$, $I(\varepsilon_{13}) = [2,4]$, $I(\varepsilon_{14}) = [1,3]$, $I(\varepsilon_{15}) = [1,3]$, $I(t_{16}) = [3,8]$, $I(\varepsilon_{17}) = [3,3]$, $I(\varepsilon_{18}) = [4,7]$, $I(\varepsilon_{19}) = [5,7]$, $I(\varepsilon_{20}) = [5,6]$, $I(\varepsilon_{21}) = [5,7]$, $I(\varepsilon_{22}) = [3,7]$, $I(\varepsilon_{23}) = [5,7]$, $I(\varepsilon_{24}) = [3,8]$, $I(\varepsilon_{25}) = [5,8]$, $I(\varepsilon_{26}) = [3,4]$, respectively. The initial marking is shown in Figure 7.2, and it corresponds to an intermediate production state. The initial marking implies that: 1) one product is being taken by R2 from M3 to AGV1 (one token in p_{12}); 2) one material is being moved to M4 by R1 (one token in p_{16}); 3) M4 is producing a product (one token in p_{17}); 4) one product is in the buffer B (one token in p_{19}). The observed word is $w = \langle t_{11}, 5\rangle\ \langle t_8, 7\rangle\ \langle t_{12}, 9\rangle\ \langle t_9, 16\rangle\ \langle t_{10}, 23\rangle\ \langle t_6, 26\rangle\ \langle t_{11}, 29\rangle\ \langle t_{12}, 31\rangle\ \langle t_7, 34\rangle\ \langle t_8, 39\rangle\ \langle t_9, 46\rangle\ \langle t_{10}, 53\rangle\ \langle t_6, 56\rangle\ \langle t_{11}, 59\rangle\ \langle t_{12}, 61\rangle\ \langle t_7, 64\rangle\ \langle t_8, 69\rangle\ \langle t_9, 76\rangle$.

The computation results are briefly summarized in Table 7.1. The columns are divided into three parts. In the 'observation' part, each row corresponds to an observed transition in the observed word w. The columns titled 'PN_DIAG' contain the results obtained using the untimed approach [15]. There are three columns in this part: column 'total' consists of the computation time consumed for diagnosis, while the diagnosis states are given in columns 'T_f^1' and 'T_f^2'. The results based on the untimed approach are shown in the last columns. The computation time consumed using FDG are represented in three columns: 1) 'total': the total computation time consumed using FDG; 2) 'generate': the computation time spent on updating the FDG; 3) 'diagnose': the computation time spent on diagnosis on FDG. The time unit is second. The column 'size' contains the size of FDG and the values in the column '$\mathcal{S}$' are the sizes of $\mathcal{S}$ (the set of consistent state classes). The diagnosis states are given in columns 'T_f^1' and 'T_f^2'.

Let us first discuss the time consumed in both approaches. The results are divided into two parts. The first part starts from the first row until the thirteen one, and the rest corresponds to the second part. In the first part, the size of the FDG increases in each step and in the second part it remains constant because the algorithm of updating the FDG (Algorithm 4.2) is not performed. In the first part, the time

Table 7.1: Results of some numerical simulations carried on the system in Figure 7.2 (time unit: s=second). By using the information from time intervals, the FDG based approach computes the diagnosis states in a more precise way (less uncertain states) than the untimed ones. However, in order to use the time information, the time complexity is increased.

	observation	PN_DIAG (Untimed)			FDG						
	t_o	total (s)	T_f^1	T_f^2	total (s)	generate (s)	diagnose (s)	$\mathcal{S}$	size	T_f^1	T_f^2
1	$\langle \epsilon, 0 \rangle$	0.0017	N	U	0.1249	0.1243	0.0006	1	4	N	U
2	$\langle t_{11}, 5 \rangle$	0.0029	N	F	0.0439	0.0428	0.0011	1	5	N	F
3	$\langle t_8, 7 \rangle$	0.0013	N	F	0.0408	0.0391	0.0017	1	6	N	F
4	$\langle t_{12}, 9 \rangle$	0.0009	N	F	0.0507	0.0490	0.0017	1	7	N	F
5	$\langle t_9, 16 \rangle$	0.0012	N	F	0.0601	0.0588	0.0013	1	8	N	F
6	$\langle t_{10}, 13 \rangle$	0.0014	N	F	0.1282	0.1270	0.0012	1	10	N	F
7	$\langle t_6, 26 \rangle$	0.0054	**U**	F	0.7689	0.7671	0.0018	2	22	**N**	F
8	$\langle t_{11}, 29 \rangle$	0.0031	**U**	F	1.0245	1.0198	0.0047	5	35	**N**	F
9	$\langle t_{12}, 31 \rangle$	0.0029	**U**	F	1.2939	1.2873	0.0066	7	40	**N**	F
10	$\langle t_7, 34 \rangle$	0.0024	U	F	1.7514	1.7314	0.0200	5	47	U	F
11	$\langle t_8, 39 \rangle$	0.0036	U	F	1.3957	1.3866	0.0091	6	53	U	F
12	$\langle t_9, 46 \rangle$	0.0019	U	F	0.7983	0.7907	0.0076	5	58	U	F
13	$\langle t_{10}, 53 \rangle$	0.0047	U	F	0.1424	0.1400	0.0025	2	59	U	F
14	$\langle t_6, 56 \rangle$	0.0082	**U**	F	0.0029	0.0003	0.0026	2	59	**N**	F
15	$\langle t_{11}, 59 \rangle$	0.0041	**U**	F	0.0054	0.0003	0.0051	5	59	**N**	F
16	$\langle t_{12}, 61 \rangle$	0.0036	**U**	F	0.0094	0.0004	0.0090	7	59	**N**	F
17	$\langle t_7, 64 \rangle$	0.0037	U	F	0.0147	0.0007	0.0140	5	59	U	F
18	$\langle t_8, 69 \rangle$	0.0039	U	F	0.0133	0.0006	0.0126	6	59	U	F
19	$\langle t_9, 76 \rangle$	0.0037	U	F	0.0085	0.0005	0.0081	5	59	U	F

used by the approach in [15] is shorter than the approach based on FDG, where most of the time is spent on updating the FDG. In the second part, the time spent by the approach based on FDG is decreased because FDG is not updated.

The time spent on diagnosis is influenced by the size of $\mathcal{S}$, because the diagnosis algorithm (Algorithm 5.1) uses the output edges of each state class in $\mathcal{S}$ to compute diagnosis states. Consider the column of the time of diagnosis ('diagnose') and the sizes of $\mathcal{S}$ ('$\mathcal{S}$'). The influence is twofold. On one hand, in the diagnosis after the observation of t_{11} (the second row), the algorithm finds one consistent state class, while after the observation of t_{12} for the second time (the ninth row), seven consistent state classes are found. This makes the time consumed by the diagnosis algorithm after the observation of t_{12} for the second time higher than the one corresponding to t_{11}. If there are n state classes in the consistent state classes computed in the previous step, the diagnosis algorithm has to check the output edges of n state classes for the firings of fault transitions. This influences the time spent on diagnosis.

In the second part of Table 7.1 (from the fourteenth row), the values in the column 'generate' should be very small, because the FDG is not updated. The times in the column 'generate' correspond to the execution of steps 8 to 11 in Algorithm 5.1, which check if consistent state class has been explored or not (step 8) and update the FDG if the consistent state class has not been explored (steps 9 to 11).

In general, the time complexity of updating the FDG is higher than the one of the diagnosis on FDG. In the case in which the FDG is updated, the consumed time by our approach is higher than the untimed approach presented in [15]. On the contrary, the FDG based approach works as fast as the untimed approach. Since the FDG is incrementally updated, the enumeration of all reachable state classes of the time system is avoided. This feature makes the proposed approach to have a low space complexity comparing with other state based ones, such as model checking in [12].

After the discussion of running time, let us look at the diagnosis states. In the seventh row, when t_6 is observed for the first time, the diagnosis states of T_f^1 given by the untimed approach and FDG are U and N, respectively. In the untimed PN, when $w = t_{11}t_8t_{12}t_9t_{10}t_6$ is observed, there exists a consistent marking $m_{10} = p_{15} + p_{16} + p_{18} + p_{23} + p_{24} + p_{25} + p_{26} + p_{27} + p_{28} + p_{30} + p_{31} + p_{34} + 6p_{36} + p_{37} + 2p_{38}$. The enabled transitions at m_{10} are t_7, t_{11} and ε_{23}. If ε_{23} is fired, then a token is generated in p_{29}, enabling the fault transition ε_{25}. Therefore, there exists a consistent firing sequence containing a fault transition and a firing sequence without any fault transition (empty firing sequence for example). In the timed case, there exists two consistent state classes α_9 and α_{10} consistent with w shown in Figure 7.3. Let us consider α_{10}. The marking of α_{10} is m_{10} and the firing domain is $F_{10} = (7 \leqslant x_7 \leqslant 8) \wedge (1 \leqslant x_{11} \leqslant 3) \wedge (5 \leqslant x_{23} \leqslant 7)$. The enabled transitions are t_7, t_{11} and ε_{23} (the same as in the untimed case). However, the firing domain F_{10} indicates that only the observable transition t_{11} can be fired at α_{10}. Therefore, there does

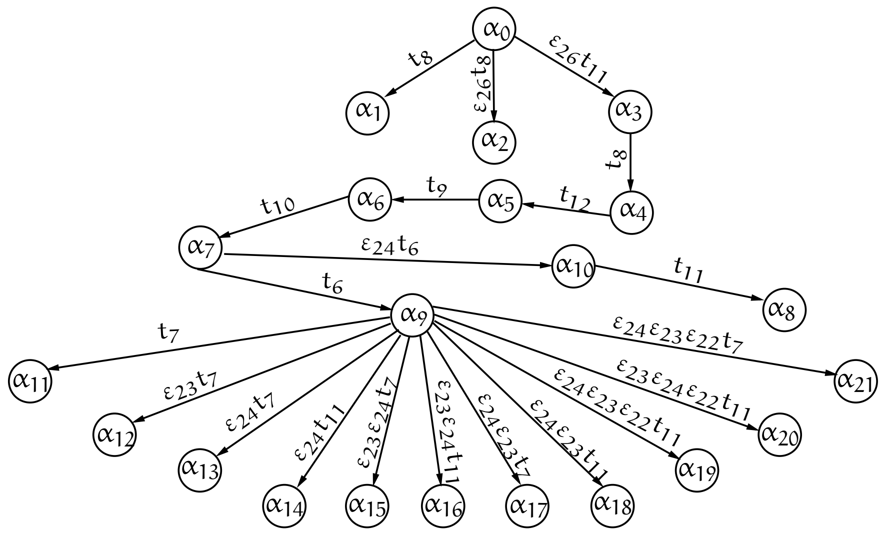


Figure 7.3: The FDG corresponding to the observation word $w = \langle t_{11}, 5\rangle \langle t_8, 7\rangle \langle t_{12}, 9\rangle \langle t_9, 16\rangle \langle t_{10}, 23\rangle \langle t_6, 26\rangle$.

not exist any unobservable firing sequence starting at α_{10} containing the fault transition ε_{25}. The unobservable firing sequence starting at the other consistent state class α_9 do not contain ε_{25} either. Hence, the diagnosis state is normal.

7.2.2 *An IC Wafer Fabrication System*

In this case, we illustrate the FDG based approach using a photolithography (photo) area in a real-world IC wafer fabrication system [72]. The major products of the system are 4MB unpackaged DRAM wafers, which will be diced and packaged in another plant. The whole process is divided into four processes and the system is also organized into four corresponding functional segments. The four processes are photo, etching, diffusion and thin film processes: The photo area makes patterns of photo resistors on wafers and its process is divided into 20 subprocesses called *layers*. Each layer belongs to one of four types: *N1* and *N2*, which are *noncritical* layers, and *C1* and *C2*, which are *critical* layers. Fifty machines belonging to seven types are available in the area. The machines are steppers for alignment and exposure, developers for development and baking individually.

The PN mode of the IC wafer fabrication system is shown in Figure 7.4 and meanings of places and transitions are in Table 7.3. Distinguished by the output transitions of p_6, there are five concurrent processes, which are respectively starting from t_5, t_9, t_{13}, t_{19} and t_{20}, and t_{34} and t_{35}. The process starting from t_{45} and t_{46} deals with the outputs of the previous ones unless faults occur. If no fault occurs in the concurrent processes, they send their outputs to the buffer p_{16}, and the last process continues to deal with the outputs from the buffer. If a fault occurs, e.g., the fault associated to t_{44}, the corresponding process sends directly the output to the stripper p_{19} skipping the last process. Faults may also occur in the last process. In

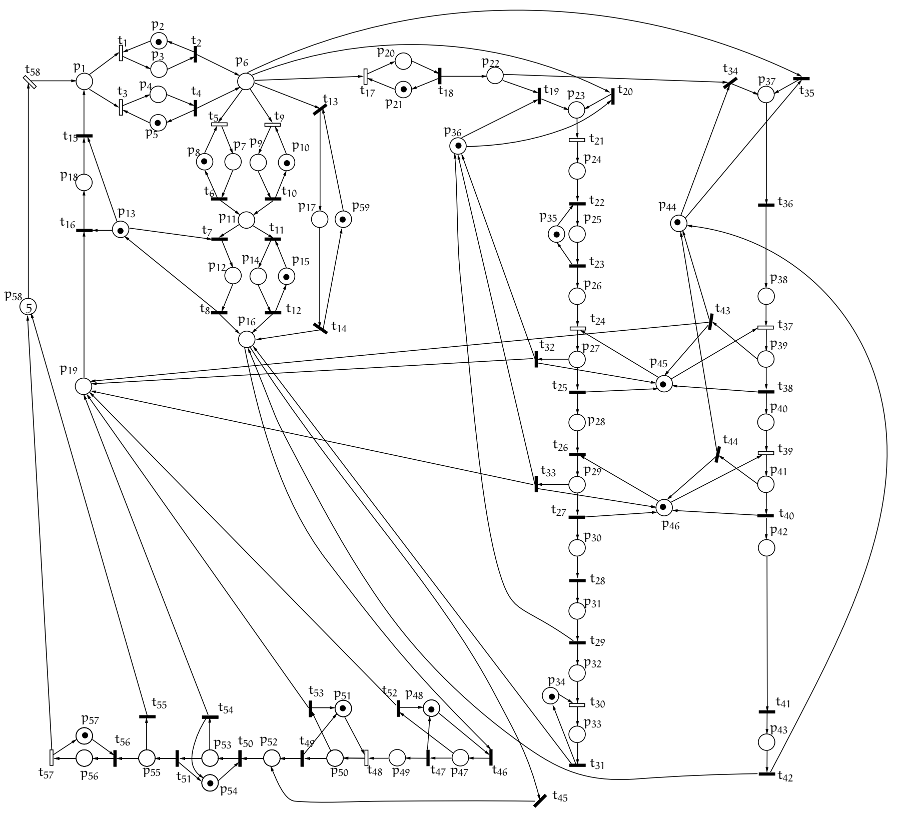


Figure 7.4: PN model of a semiconductor production system [72]

the model, if there is not any fault occurrence in the whole system, tokens are generated in p_{58}; otherwise, the number of tokens in p_{19} increases.

In the model, we associate the time interval $[0, 1000]$ to each transition. The observable transitions are t_1, t_3, t_5, t_9, t_{17}, t_{21}, t_{24}, t_{30}, t_{37}, t_{39}, t_{48}, t_{57} and t_{58}. The fault classes are $T_f^1 = \{t_{32}, t_{33}, t_{43}\}$ and $T_f^2 = \{t_{44}, t_{52}, t_{53}, t_{54}\}$. The meanings of fault transitions are shown in Table 7.3. We assume the observed word is $w = \langle t_{58}, 8\rangle$ $\langle t_1, 16\rangle$ $\langle t_{17}, 24\rangle$ $\langle t_{21}, 32\rangle$ $\langle t_{24}, 40\rangle$ $\langle t_{30}, 48\rangle$ $\langle t_{48}, 56\rangle$ $\langle t_{57}, 64\rangle$ $\langle t_{58}, 72\rangle$ $\langle t_1, 80\rangle$ $\langle t_{21}, 88\rangle$ $\langle t_{24}, 96\rangle$ $\langle t_3, 104\rangle$ $\langle t_{17}, 112\rangle$ $\langle t_{37}, 120\rangle$ $\langle t_{39}, 128\rangle$ $\langle t_{48}, 136\rangle$ $\langle t_{57}, 144\rangle$ $\langle t_{58}, 152\rangle$ $\langle t_1, 160\rangle$ $\langle t_{17}, 168\rangle$.

The computation results are shown in Table 7.3 and the time unit is second. In the beginning of Table 7.3, the size of the FDG increases from 1 to 41 until t_{48} is observed at time 56 (the seventh row). In the table, the column of the sizes of FDG ("FDG.size") and the column of time consumption of updating FDG ("FDG.gen") shows that the FDG is not updated from the following observed transition t_{48} is observed at time 56 (the seventh row) to t_{24} is observed at time 96 (the twelfth row). In the last six observed transitions, the FDG is also not updated. At the time when t_3 is observed, the fault occurrence of the fault class T_f^1 is F. From this moment, the diagnosis state of T_f^1 remains being faulty and has nothing to do with which transition is observed.

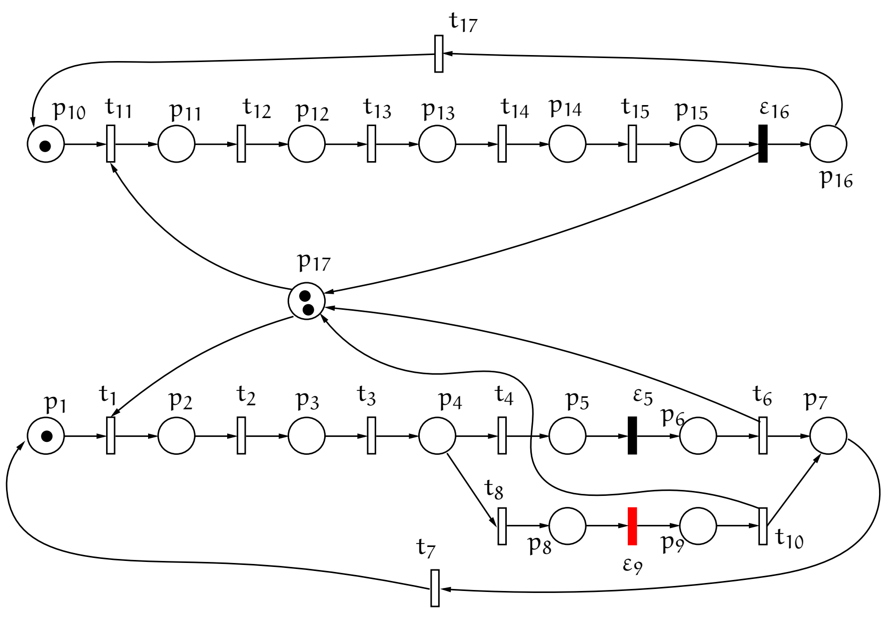


Figure 7.5: A manufacturing system.

7.3 DECENTRALIZED DIAGNOSIS

Here, we apply the decentralized diagnosis approach on a manufacturing system. The TPN model of the system is shown in Figure 7.5. The purpose of this case study is to illustrate how local FDG is constructed and the communication between local diagnosers and the coordinator.

There are two processes in the TPN: one process contains places $p_i, i = 1, \ldots, 9$, and the other one contains $p_j, j = 10, \ldots, 16$. Place p_1 (p_{10}) represents a material is waiting to be processed in the first (second) process. In order to be processed by the first (second) process, there must be a free plate, which is allocated by the first (second) process by firing the transition t_1 (t_{11}). When the first (second) process finishes, transitions t_6 and t_{10} (t_{16}) are fired and the corresponding plate is released. In the first process, there are two subprocesses, which start from t_4 and t_8, respectively. In the subprocess starting from t_8, ε_9 is One material is waiting to be processes (one token in p_1 and p_{10}), and there are two free plates initially (two tokens in p_{17}). The time intervals associated to transitions are xxx. We consider the global observed word $w = \langle t_1, 1\rangle\langle t_{11}, 1\rangle\langle t_2, 2\rangle$.

There are three subsystems such that: 1) in the first subsystem Σ_1, t_1, t_4, t_8, t_{12} and t_{15} are observable; 2) in Σ_2, the observable transitions are t_2, t_6, t_{10}, t_{13} and t_{17}; 3) the observable transitions in Σ_3 include t_3, t_7, t_{11} and t_{14}.

Because in diagnosis we do not use markings and domains of state classes, in the sequel, only labels associated with edges and diagnosis labels associated with nodes are provided.

The labels of edges in $\mathcal{G}_1$ are:

$\alpha_0 \to \alpha_1$: $\langle t_1, [1, 1], (0)\rangle$;

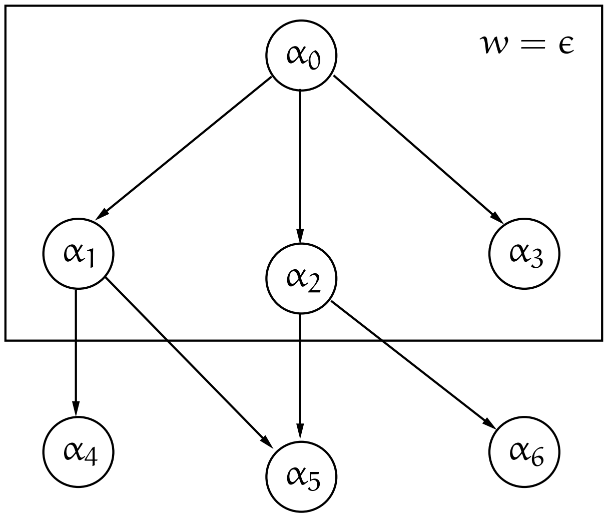


Figure 7.6: The local FDG $\mathcal{G}_1$ of Σ_1 corresponds to the observation of t_1.

$\alpha_0 \to \alpha_2$: $\langle t_{11}t_1, [1,2], (0)\rangle$;

$\alpha_0 \to \alpha_3$: $\langle t_{12}t_{12}, [2,2], (0)\rangle$;

$\alpha_1 \to \alpha_4$: $\langle t_{11}t_{12}, [1,1], (0)\rangle$;

$\alpha_1 \to \alpha_5$: $\langle t_{11}t_2t_{12}, [1,1], (0)\rangle$;

$\alpha_2 \to \alpha_5$: $\langle t_2t_{12}, [1,1], (0)\rangle$;

$\alpha_2 \to \alpha_6$: $\langle t_{12}, [0,1], (0)\rangle$.

The diagnosis labels associated with nodes α_i, $i = 0,1,2,3$, in $\mathcal{G}_1$ are (0).

The labels of edges in $\mathcal{G}_2$ are:

$\alpha_0 \to \alpha_1$: $\langle t_1t_{11}t_{12}t_2, [2,2], (0)\rangle$;

$\alpha_0 \to \alpha_2$: $\langle t_{11}t_1t_2, [2,2], (0)\rangle$, $\langle t_1t_{11}t_{12}, [2,2], (0)\rangle$;

$\alpha_0 \to \alpha_3$: $\langle t_{11}t_1t_{12}t_2, [2,3], (0)\rangle$;

$\alpha_0 \to \alpha_4$: $\langle t_{11}t_{12}t_1t_2, [3,4], (0)\rangle$;

$\alpha_0 \to \alpha_5$: $\langle t_{11}t_{12}t_1t_{13}, [4,4], (0)\rangle$;

$\alpha_1 \to \alpha_6$: $\langle t_{13}, [2,3], (0)\rangle$;

$\alpha_1 \to \alpha_7$: $\langle t_3t_{13}, [2,3], (0)\rangle$;

$\alpha_1 \to \alpha_8$: $\langle t_3t_4t_{13}, [2,3], (0)\rangle$;

$\alpha_1 \to \alpha_9$: $\langle t_3t_8t_{13}, [2,3], (0)\rangle$;

$\alpha_1 \to \alpha_{10}$: $\langle t_3t_8t_9t_{13}, [3,3], (0)\rangle$;

$\alpha_2 \to \alpha_6$: $\langle t_{12}t_{13}, [2,3], (0)\rangle$;

$\alpha_2 \to \alpha_7$: $\langle t_{12}t_3t_{13}, [2,3], (0)\rangle$;

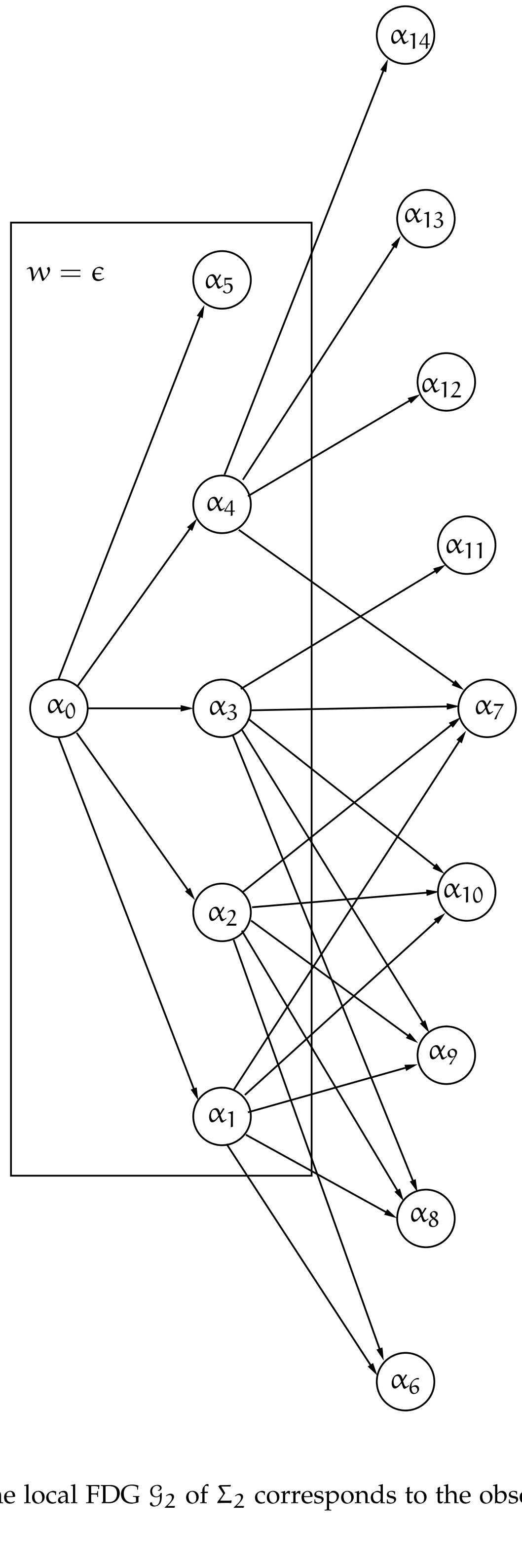


Figure 7.7: The local FDG $\mathcal{G}_2$ of Σ_2 corresponds to the observation of t_2.

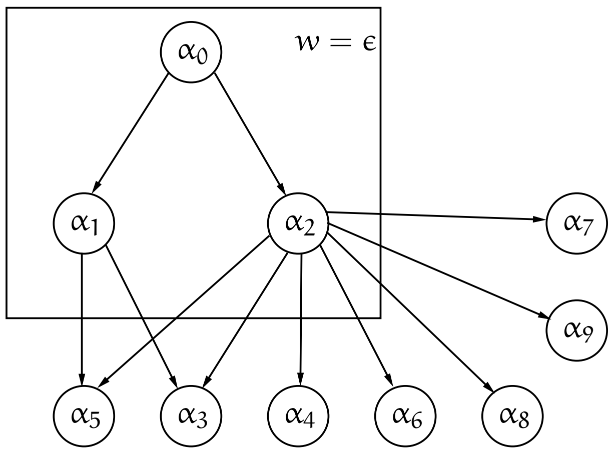


Figure 7.8: The local FDG $\mathcal{G}_3$ of Σ_3 corresponds to the observation of t_{11}.

$\alpha_2 \to \alpha_8$: $\langle t_{12}t_3t_4t_{13}, [2,3], (0)\rangle$;

$\alpha_2 \to \alpha_9$: $\langle t_{12}t_3t_8t_{13}, [2,3], (0)\rangle$;

$\alpha_2 \to \alpha_{10}$: $\langle t_{12}t_3t_8t_{13}, [2,3], (1)\rangle$;

$\alpha_3 \to \alpha_7$: $\langle t_3t_{13}, [1,3], (0)\rangle$;

$\alpha_3 \to \alpha_8$: $\langle t_3t_4t_{13}, [2,3], (0)\rangle$;

$\alpha_3 \to \alpha_9$: $\langle t_3t_8t_{13}, [2,3], (0)\rangle$;

$\alpha_3 \to \alpha_{10}$: $\langle t_3t_8t_{13}, [3,3], (1)\rangle$;

$\alpha_3 \to \alpha_{11}$: $\langle t_{13}, [1,3], (0)\rangle$;

$\alpha_4 \to \alpha_7$: $\langle t_3t_{13}, [1,2], (0)\rangle$;

$\alpha_4 \to \alpha_{12}$: $\langle t_{13}, [0,2], (0)\rangle$;

$\alpha_4 \to \alpha_{13}$: $\langle t_3t_4t_{13}, [2,2], (0)\rangle$;

$\alpha_4 \to \alpha_{14}$: $\langle t_3t_8t_{13}, [2,2], (0)\rangle$;

The diagnosis labels associated with nodes α_i, $i = 0,1,4$, in $\mathcal{G}_2$ are (0), and the one associated with α_2 and α_3 are (1).

The labels of edges in $\mathcal{G}_3$ are:

$\alpha_0 \to \alpha_1$: $\langle t_1t_{11}, [1,1], (0)\rangle$;

$\alpha_0 \to \alpha_2$: $\langle t_{11}, [1,1], (0)\rangle$;

$\alpha_1 \to \alpha_3$: $\langle t_2t_{12}t_3, [2,4], (0)\rangle$, $\langle t_{12}t_2t_3, [2,4], (0)\rangle$;

$\alpha_1 \to \alpha_5$: $\langle t_2t_{12}t_{13}t_3, [3,4], (0)\rangle$, $\langle t_{12}t_2t_{13}t_3, [3,4], (0)\rangle$;

$\alpha_2 \to \alpha_3$: $\langle t_1t_2t_{12}t_3, [2,4], (0)\rangle$, $\langle t_1t_{12}t_2t_3, [2,4], (0)\rangle$;

$\alpha_2 \to \alpha_4$: $\langle t_{12}t_1t_2t_3, [3,4], (0)\rangle$;

$\alpha_2 \to \alpha_5$: $\langle t_1t_2t_{12}t_{13}t_3, [3,4], (0)\rangle$;

$\alpha_2 \rightarrow \alpha_6$: $\langle t_1 t_{12} t_2 t_{13} t_3, [3,5], (0) \rangle$;

$\alpha_2 \rightarrow \alpha_7$: $\langle t_{12} t_1 t_2 t_{13} t_3, [3,6], (0) \rangle$;

$\alpha_2 \rightarrow \alpha_8$: $\langle t_{12} t_1 t_2 t_{13} t_{14}, [6,6], (0) \rangle$, $\langle t_{12} t_1 t_{13} t_2 t_{14}, [6,6], (0) \rangle$;

$\alpha_2 \rightarrow \alpha_9$: $\langle t_{12} t_1 t_{13} t_2 t_3, [4,6], (0) \rangle$;

The diagnosis labels associated with nodes α_0, α_1 and α_2 in $\mathcal{G}_3$ are (0).

In the initial step, each local diagnoser initializes its local FDG. The three local FDG are shown in Figure 7.6, Figure 7.7 and Figure 7.8, where the parts in the block indicated by $w = \epsilon$ are the local FDG in this step. All local diagnosers send messages to the coordinator. Let us consider Σ_1 as an example. The message from Σ_1 to the coordinator contains $\langle \overrightarrow{0}, \epsilon, (0) \rangle$, where $\overrightarrow{0}$ is the firing count vector representing the path from the initial state class to the consistent state class in the local system (in Σ_1, it is α_0), ϵ is the list of observable transitions associated in the label of the input edge corresponding to the observation (In this step, no such edge is considered, and it will be explained in next step), and (0) is the diagnosis label associated with the consistent state class. Because this is the initial step, when the coordinator receives all three messages, it uses the diagnosis labels to compute the diagnosis state. The diagnosis state is normal, because ε_9 is not fired.

Let us assume t_1 is observed at time 1. The local diagnoser corresponding to Σ_1 observes the firing of t_1, and then it updates its local FDG. The updated FDG is shown in Figure 7.6 containing all seven state classes and seven edges. The message that will be sent to the coordinator containing two entries:

- $\langle \overrightarrow{0}, t_1, (0) \rangle$ corresponding to the consistent state class α_1,
- $\langle \overrightarrow{0}, t_{11} t_1, (0) \rangle$ corresponding to the consistent state class α_2.

In the first entry, the second element t_1 is the list of observation associated with the input edge $\alpha_0 \rightarrow \alpha_1$. When this message is sent to the coordinator, the coordinator uses it to compute the diagnosis state corresponding to $w = \langle t_1, 1 \rangle$. Because before receives the message from Σ_1, the coordinator did not receiving any message, it can be inferred that, globally, the first fired observable transition is t_1. Then, $t_{11} t_1$ is not consistent with this result, and it will not be used in diagnosis and will be removed from the received message. At this moment, the received message contains only the first entry mentioned above. Using the message, the diagnosis state is computed as normal. Finally, the coordinator sends the reduced message to Σ_1. In Σ_1, the local diagnoser uses the received message to delete the consistent state classes that are not consistent with the global states (α_2). Meanwhile, the local FDG corresponding to the other subsystems are not updated, because no transition is observed by their local diagnosers.

After t_1, t_{11} is observed by the local diagnoser of Σ_3. Then the local diagnoser updates its FDG as shown in Figure 7.8 containing ten state classes and eleven edges. At last, t_2 is observed at Σ_2 and the corresponding FDG is updated by the local diagnoser and shown in Figure 7.7.

Table 7.2: Meanings of places and events associated with transitions in the photo area model

Places		Transitions	
p_{1}: Coater's buffer	p_{2}: PI Coater	t_{1}: PI Coater setup	t_{2}: PI Coater coating
p_{3}: wafer on PI Coater	p_{4}: wafer on C2 Coater	t_{3}: C2 Coater setup	t_{4}: C2 Coater coating
p_{5}: C2 Coater	p_{6}: buffer	t_{5}: Ultra Stepper setup	t_{6}: Stepping
p_{7}: wafer on Ultra	p_{8}: Ultra Stepper	t_{7}: Developer setup	t_{8}: Developing
p_{9}: wafer on Nikon	p_{10}: Nikon Stepper	t_{9}: Nikon Stepper setup	t_{10}: Stepping
p_{11}: Developer's buffer	p_{12}: wafer on Developer	t_{11}: Developer setup	t_{12}: Developing
p_{13}: Stripper	p_{14}: wafer on Developer	t_{13}: S/D setup	t_{14}: S/D processing
p_{15}: Developer	p_{16}: buffer	t_{15}: Stripping	t_{16}: Stripper setup
p_{17}: wafer on S/D	p_{18}: wafer on Stripper	t_{17}: WEE setup	t_{18}: WEE processing
p_{19}: Stripper's buffer	p_{20}: wafer on WEE	t_{19}: Nikon Stepper setup	t_{20}: Nikon Stepper setup
p_{21}: WEE	p_{22}: buffer	t_{21}: Stepping	t_{22}: Developer setup
p_{23}: wafer on Nikon	p_{24}: Developer's buffer	t_{23}: Developing	t_{24}: Overlay processing
p_{25}: pilot wafer on Developer	p_{26}: Overlay's buffer	t_{25}: Overlay Measuring	t_{26}: CD setup
p_{27}: pilot wafer on Overlay	p_{28}: CD Measure's buffer	t_{27}: CD Measuring	t_{28}: Nikon Stepper setup
p_{29}: pilot wafer on CD Measure	p_{30}: Stepper's buffer	t_{29}: Stepping	t_{30}: Developer setup
p_{31}: body wafer on Stepper	p_{32}: Developer's buffer	t_{31}: Developing	t_{32}: Overlay fails
p_{33}: body wafer on Developer	p_{34}: Developer	t_{33}: CD fails	t_{34}: S/D setup
p_{35}: Developer	p_{36}: Nikon Stepper	t_{35}: S/D setup	t_{36}: S/D processing
p_{37}: pilot wafer on S/D	p_{38}: Overlay's buffer	t_{37}: Overlay setup	t_{38}: Overlay Measuring
p_{39}: pilot wafer on Overlay	p_{40}: CD Measure's buffer	t_{39}: CD setup	t_{40}: CD Measuring
p_{41}: pilot wafer on CD	p_{42}: S/D's buffer	t_{41}: S/D setup	t_{42}: S/D processing
p_{43}: body wafer on S/D	p_{44}: S/D	t_{43}: Overlay fails	t_{44}: CD fails
p_{45}: Machine Overlay	p_{46}: Machine CD	t_{45}: ϵ	t_{46}: Overlay setup
p_{47}: wafer on Overlay	p_{48}: Machine Overlay	t_{47}: Overlay measuring	t_{48}: CD setup
p_{49}: buffer	p_{50}: wafer on CD	t_{49}: CD Measuring	t_{50}: ADI setup
p_{51}: Machine CD	p_{52}: ADI's buffer	t_{51}: ADI processing	t_{52}: Overlay fails
p_{53}: wafer on ADI	p_{54}: Machine ADI	t_{53}: CD fails	t_{54}: ADI fails
p_{55}: buffer	p_{56}: wafer on UV Curer	t_{55}: leaving the area	t_{56}: UV Curer setup
p_{57}: UV Curer	p_{58}: Material buffer	t_{57}: UV Curing	t_{58}: wafer input
p_{59}: S/D			

Table 7.3: Results of some numerical simulations carried on the system in Figure 7.4 (time unit: s=second)

	w	PN_DIAG (s)	FDG.size	FDG.sum (s)	FDG.gen (s)	FDG.diag (s)	$\Delta(w, T_f^1)$	$\Delta(w, T_f^2)$
	$\langle \varepsilon, 0 \rangle$	0.0008	1	0.1077	0.1047	0.0027	N	N
1	$\langle t_{58}, 8 \rangle$	0.0021	5	0.4691	0.4669	0.0022	N	N
2	$\langle t_1, 16 \rangle$	0.0217	26	20.4753	20.4703	0.0023	N	U
3	$\langle t_{17}, 24 \rangle$	0.0066	28	1.8430	1.8408	0.0021	N	N
4	$\langle t_{21}, 32 \rangle$	0.0085	32	0.8348	0.8324	0.0024	N	N
5	$\langle t_{24}, 40 \rangle$	0.0202	39	6.6343	6.6320	0.0023	U	N
6	$\langle t_{30}, 48 \rangle$	0.0201	40	10.0584	10.0563	0.0021	N	U
7	$\langle t_{48}, 56 \rangle$	0.0132	41	6.6065	6.6044	0.0020	N	U
8	$\langle t_{57}, 64 \rangle$	0.0041	41	0.0047	0.0006	0.0040	N	N
9	$\langle t_{58}, 72 \rangle$	0.0020	41	0.0029	0.0008	0.0022	N	N
10	$\langle t_1, 80 \rangle$	0.0241	41	0.0027	0.0005	0.0022	N	U
11	$\langle t_{21}, 88 \rangle$	0.0058	41	0.0033	0.0010	0.0023	N	N
12	$\langle t_{24}, 96 \rangle$	0.0126	41	0.0035	0.0006	0.0029	U	N
13	$\langle t_3, 104 \rangle$	0.0168	42	20.6254	20.6231	0.0023	F	U
14	$\langle t_{17}, 112 \rangle$	0.0043	42	0.0035	0.0008	0.0028	F	N
15	$\langle t_{37}, 120 \rangle$	0.0062	45	2.2392	2.2372	0.0020	F	N
16	$\langle t_{39}, 128 \rangle$	0.0110	49	14.5215	14.5195	0.0020	F	U
17	$\langle t_{48}, 136 \rangle$	0.0078	49	0.0034	0.0006	0.0028	F	U
18	$\langle t_{57}, 144 \rangle$	0.0037	49	0.0037	0.0008	0.0029	F	N
19	$\langle t_{58}, 152 \rangle$	0.0014	49	0.0034	0.0007	0.0026	F	N
20	$\langle t_1, 160 \rangle$	0.0127	49	0.0038	0.0010	0.0028	F	U
21	$\langle t_{17}, 168 \rangle$	0.0046	49	0.0035	0.0008	0.0027	F	N

8 CONCLUSIONS AND FUTURE WORKS ON FAULT DIAGNOSIS

Discrete Event Systems (DES), such as transportation, manufacturing and logistics systems, may contain faulty behaviors, e.g., mis-delivered cargoes in logistics systems. Fault diagnosis is the process to detect and isolate faults. In general, faulty behaviors cannot be observed directly. Therefore, it is critical to use observable events to estimate faulty events (behaviors). Many works consists of building *diagnosers* to achieve this task. Some diagnosers represent an alternative model of the target system containing only normal events in the target system. The diagnoser evolves with the target system and synchronizing with it using the output (observable events) of the target system. If their outputs do not match (synchronization failed), then events not belonging to the diagnoser have occurred meaning faulty behaviors are detected. Other diagnosers (e.g., [29]) contain both normal and faulty behaviors of the target systems, but they remove parts of the target system model that are not used in diagnosis, or add information that can improve the diagnosis process.

Time Petri Net (TPN) is one of many modeling tools of DES. It uses places to represent local states, and its transitions mean basic dynamics of these local states. The timing information (interval time delays) let TPN be capable to describe temporal knowledges of DES. TPN is widely applied in fault diagnosis on DES (also in other aspects of DES, e.g., performance analysis and control). When faults mean faulty events, in TPN, some transitions, called *fault transitions* represents faults. The firing of a fault transition implies the occurrence of corresponding faulty event. Because fault transitions are not observable, diagnosis is using observable information to estimate the firings of fault transitions. The observable information contains observable transitions and/or observable places in which the numbers of tokens can be measured by an external observer.

In this first part of the thesis, we have focused on fault diagnosis on DES modeled with TPN, in which some unobservable transitions represent faults and the observable information contains only the firings of some of other transitions. We have proposed *Fault Diagnosis Graph* (FDG) as the base of diagnosis. FDG contains both faulty and normal behaviors of a target system. An FDG is constructed from an *State Class Graph* (SCG), which is an abstracted reachability graph of a TPN system. In the construction of an FDG, we remove parts of the SCG if they will not be used in diagnosis. In on-line diagnosis, time is critical. Therefore, we introduced two techniques to improve the time complexity of diagnosis using FDG. First, labels are associated with nodes and edges to preserve information that is frequently used in diagnosis. The information contains firing sequences composed of unobservable transitions and it is generated when FDG is being constructed and stored as labels, so that when it is required, it can be taken from labels simply. Second, we proposed an incremental approach to construct the FDG. The incremental approach implements that an FDG is constructed with an observation, that is only parts of the FDG is required for diagnosis. Some parts of the FDG may not be used forever, and then we do not need to spend time on constructing them.

We dealt with two diagnosis approaches: centralized and decentralized diagnosis with FDG. In centralized diagnosis, a single diagnoser reads outputs (observations) of a system, constructs the FDG of the system and computes diagnosis states. Due to the FDG, the diagnoser can quickly update diagnosis states, especially when repetitive behaviors appear (at this time, the construction of FDG is skipped and all information used in diagnosis can be taken from the existing FDG). Using a case study, we illustrated that the use of timing information improves diagnosis. The critical factor is the *firing domain of a given firing sequence*, which represents the time interval that all transitions in a given firing sequence can fire sequentially. It is used to check whether a firing sequence is consistent with an observable transition and the observed time or not. The case study showed that the number of uncertain diagnosis states (i.e., the firings of fault transitions cannot be determined) is reduced by using FDG, comparing with diagnosis without the use of timing information.

In decentralized diagnosis, we consider a system that is distributed. Because it is not convenient to use a centralized diagnosis, we proposed a decentralized diagnosis scheme consisting a *local* diagnoser for each subsystem and a *coordinator* enabling the collaboration among local diagnosers. We adapted FDG to decentralized diagnosis with a properly designed collaboration scheme, in order to reduce the costs of maintaining a coordinator and communication between local diagnosers and the coordinator. First, the processing and storing capability of the coordinator is constrained at a low level such that it cannot build FDG or diagnose on an FDG. Second, the communication between local diagnosers and the coordinator contains only observed transitions and consistent states. The communication does not send any FDG, which is costly to be sent in a decentralized environment.

Some topics based on the work in this part remains open:

- Application of FDG on state estimation: State estimation is a topic related with fault diagnosis. State estimation cares possible states of a system at a given time, instead of firings of some transitions. In general, TPN in state estimation contains observable and unobservable places and transitions, which is similar with the configuration of fault diagnosis.

- Predict the firings of fault transitions: Predicting the firing of a fault transition in a given time interval in future is interesting. One solution is to modify labels to the output edges of each node in FDG so that they contain prediction information. The prediction information contains at which time interval a fault transition can be fired.

Part III

PETRI NET IN ROBOT PLANNING

9 INTRODUCTION TO PETRI NET IN ROBOT PLANNING

9.1 INTRODUCTION

When multiple mobile robots complete their tasks in a shared environment, planning and controlling them becomes critical. Mobile robot planning and controlling has attracted a lot of attention in last decades [19, 39]. An extensively studied problem in this field is mobile robot navigation. It concentrates on automatic building control strategies such that robots can reach target positions avoiding *collisions* (e.g., *deadlock*).

In this part, we discuss the avoidance of collisions in multiple robots systems in shared environments. A shared environment contains a map, in which all robots moves. The map is partitioned into regions, and some regions are constrained by limited capacities (e.g., only one robot can be in the region at a moment). The following two problems have been investigated.

In Chapter 10, some regions' capacities are one, while others are not constrained. Each robot has one or more possible trajectories and it will follow one of them. In order to ensure a deadlock free movement of all robots, on-line (real time) control is applied to all robots. We use S^3PR (short for *Systems of Simple Sequential Processes with Resources*) to deal with this problem. S^3PR is a subclass of PN for resource allocation problems. It contains processes and resources, where resources are shared by processes. We use processes to describe trajectories, and resources imply capacities of regions shared among robots. Deadlock problems in S^3PR are caused by *bad siphons*, which is a PN structure [24]. When a bad siphon is emptied, no token can enter it and it implies a deadlock. Therefore, in S^3PR, deadlock prevention policies ensure all bad siphons will not be emptied (if emptied, it will remain empty forever). Two types of on-line control for avoiding deadlocks are structural control and behavioral control. The structural on-line control in S^3PR is implemented using PN structures, while behavior controller uses the reachability space of a PN. While a behavior controller may extend to other subclasses of PN, its computational complexity is high, because the reachability space of a PN could be huge. In Chapter 10, we construct a structural controller. A classical solution is to use monitor places. A monitor place controls the number of tokens that can leave a bad siphon. It will prevent the last token from leaving the bad siphon, meaning that the bad siphon will always has at least one token in it. For example, the initial marking of the monitor place is set to be less than the number of initial tokens in the bad siphon so that not all tokens can leave the bad siphon. When a token leaves the bad siphon, one token in the monitor place is consumed; when a token enters the bad siphon, one token is added to the monitor place.

Such a controller is compact, but it has a disadvantage on cost. The controller consists of several monitor places, which must be explicit implemented and maintained. The tasks (trajectories) of robots may change, because the robots may be reused for other purpose. When tasks of robots change frequently, the implementation and maintenance of monitor places are costly. When trajectories change, new S^3PR will be used to represent new trajectories. It means the control places for old S^3PR are not applicable and new ones must be built and maintained. We propose a control policy without using any monitor place so that this cost is waived. Our approach needs communicators for robots, but they are reusable with the robots. Our policy uses *inhibitor arcs*, where an inhibitor arc prohibits the firing of a transition when the input place of the inhibitor arc is not empty. We call it *decentralized control policy*, because by using inhibitor arcs, the decision that a robot can move forward or stay is not made centrally, and then a centralized controller is not necessary anymore. Because inhibitor arcs will ensure the mutual exclusion between a set of places, the number of inhibitor arcs will be, in general, greater than the number of arcs necessary in the centralized implementation (i.e., normal arcs necessary to introduce the monitor places). However, the proposed approach is an alternative to the one based on monitor places, which will allow a decentralized implementation and, in some cases, will need a few cost for implementation.

For the formal point of view, the inhibitor arcs in the control policy introduce a new class of PN, named S^3PR^2 (S^3PR with *Reading arcs*), obtained from the S^3PR modeling of the robot trajectories together with the inhibitor arcs controlling bad siphons. This class is characterized and liveness analysis is addressed.

In Chapter 11, we propose an algorithmic procedure for avoiding collisions for multiple mobile robots that are already planned in the same environment. The environment (map) is partitioned into regions, and the capacity (meaning the maximum number of robots can be in a region at the same time) of each region is one. Each robot starts from and returns to a depot adjacent to the environment. We assume that each robot has a set of planned trajectories, and by following any of these trajectories, it completes its motion task. Each trajectory consists of a sequence of regions from the environment that the robot can follow and the time duration of traversing any region is known. We consider that no real time (on-line) controller is applicable. It means that once a robot starts to move on a trajectory, it cannot be paused or rerouted. Two cases are considered, in one case each robot chooses the trajectory to follow and in the other the trajectory is chosen by a central unit that starts or programs the robots. Collisions are avoided by finding some initial time delays for trajectories, i.e., by delaying the moment for starting the motion of a robot. The delays ensure that at each time instant each region is occupied by at most one robot, and two robots moving in opposite direction do not swap their regions by simultaneously crossing the shared region border. Moreover, since robots start from a depot and finally goes into the depot where in the depot no collision can occur, the problem always has a solution. The problem is casted as a *Mixed Integer Linear Programming* (MILP) optimization, and the obtained initial delays guarantee that all robots finish their movement in minimum time, i.e., the shared environment becomes empty as quick as possible. Based on numerical simulations, we include a statistical study comparing the solutions for the considered two cases.

9.2 LITERATURE REVIEW

Ezpeleta et al.[24] proposed a PN supervisor to enforce liveness using monitor places. It is the classical solution using structural analysis techniques to prevent deadlocks in PN. They introduce S^3PR and establish the relationship between minimal siphons and liveness. Moreover, they prove that an S^3PR is live iff no siphon can be emptied forever. A monitor place is added to each minimal siphon that can be emptied (*bad siphon*). The idea is to ensure that, all bad siphons will not be emptied, for every reachable marking. By adding monitor places, new bad siphons may be introduced. Therefore, an iterative procedure is needed to ensure liveness. Eventually, a live system is obtained. The iterative procedure is bounded, according to the numbers of resources and processes. Their approach separates a model and its controller.

A deadlock prevention policy is proposed in [69] for *Production Petri Net* (PPN), a subclass of PN that each transition has only one input activity place and one input resource place. If the input activity place of a transition is marked, then the transition is *process enabled*, and if the resource place of the transition is marked, then the transition is *resource enabled*. If a transition is both process and resource enabled, then it is said to be enabled. The *deadlock structure* is a set of process enabled transitions that are not resource enabled, i.e., no enable resources to complete the corresponding activities. It means that the firings of these transitions need some resources, but these resources are not available in the deadlock structure and no transition in it can be fired. It is important to clarify that a deadlock structure may not be a bad siphon. To ensure liveness, a monitor place is added to each deadlock structure. It is proved that if the number of each critical resource is greater than one, a maximally permissive controller can be obtained, where maximally permissive means in the controlled live net every live marking in the original net can be reached.

An MILP based deadlock detection approach is proposed in [34] to solve the computational complexity in [24]. The authors introduce an iterative deadlock prevention policy for S^3PR consisting two phases. The first phase builds monitors to control siphons. By introducing the monitors, new siphons may be created, and then the second one concentrates on control of new siphons introduced by monitor places. In the first phase, in each iteration, a minimal siphon is derived from a maximal unmarked siphon computed by using the MILP based deadlock detection method. Then, a monitor place is added to control the minimal siphon. By repeating these steps, all siphons in the original S^3PR can be controlled. In the second phase, minimal siphons that contain monitor places are derived by MILP problems, and then a monitor place is used to control each of them. The authors provided examples, in which their control policy is more permissive than the one in [24] by examples.

There are several related works to the problem we consider, as follows. In [38], each robot has multiple possible trajectories, but the traversal time for each region is unknown, meaning no time information is available. Some regions have a limited capacity (possible larger than one) for simultaneously accommodating more robots, and the robots can communicate with others. *Robot Motion Petri Net* (RMPN) is introduced to solve the problem, where RMPH has a similar behavior to S^3PR. [38] constructs a global model for all robots and trajectories in form of a Petri net with special monitor places. Based on these places, the robots are paused during their motion before entering an area where collisions or deadlocks are possible.

[49] focuses on the collision-free coordination of multiple robots with dynamic constraints (i.e., velocity, acceleration and force/torque constraints) following predefined trajectories. An approach is proposed to build multiple robots' continuous collision free velocity profiles satisfying all requirements including dynamic constraints and minimizing the completion time. The approach identifies collisions in robots' trajectories and optimize velocities of robots to avoid collisions. The collisions free constraints are formulated as a *Mixed Integer Nonlinear Programming* (MINLP) problems. In order to reduce the complexity to solve MINLP problems, two MILP problems are proposed in [49]. Only one path is given for each robot and the optimization imposes the necessary crossing time for each region along trajectory, instead of a single initial delay and no controllable crossing times through regions as in our case.

In coordinating the motions of multiple robots in a shared environment, [2] discuss the problem of coordination of the motions of multiple robots with predefined trajectories (both path and velocity). The necessary sufficient and necessary conditions are identified for multiple robots coordination. A MILP formulation is used to represent the optimization problem, and the MILP problems can be solved using commercial solvers. The potential collision conditions are identified by using a collision detection software. The proposed approach can deal with multiple robots with various degrees of freedom, where the number of degrees of freedom is not restricted. The approach can also be applied to *Automated Guided Vehicles* (AGV) with fixed paths.

10 DECENTRALIZED DEADLOCK PREVENTION

In this chapter, we discuss the problem of deadlock prevention in robot planning. In the robot planning, multiple robots move in an area, which is partitioned into regions. Some regions have limited capacities. Deadlock may appear when some robots cannot move forward forever. In order to solve the problem, we use S^3PR , a widely used subclass of PN in resource allocation problems, to represent robots' plans. Based on well developed theories on siphon analysis, we propose a decentralized control policy using inhibitor arcs to prevent the system from going into deadlock states. We will first introduce our problem by using an example, and then quickly recall some concepts on PN that will be used in this chapter. Then we introduce S^3PR and theoretical results on it. Finally, decentralized control policy is addressed.

10.1 INTRODUCTION

In this chapter we consider the design of a deadlock prevention control policy for a team of mobile robots that should follow some trajectories in order to accomplish a given task. The set of possible trajectories are assumed to be known and are computed by using a robot-planning algorithm on a partitioned environment containing some regions of interest. The capacities of regions (i.e., the number of robots that can be simultaneously in that regions is limited) can be seen as limited available resources in a resource allocation system (RAS). We assume that the capacities of some regions are lower than the number of robots. It implies not all robots cannot enter into these regions at the same time.

In [38], we proposed a modeling methodology for this kind of systems and it is proved that the obtained Petri net (PN) model belongs to the well-known class of S^3PR. In the case of manufacturing systems modeled by S^3PR there exist many results for deadlock prevention [43]. However, as we will show in this chapter, many of them imply a centralized implementation. We propose a different method, based on inhibitor arcs that can be applied in a decentralized way. This is an alternative to the deadlock prevention strategy based on monitor places that could be used in several applications since the implementation cost could be smaller. In Chapter 11, we consider the robot planning problem without any real time controller.

10.2 MOTIVATING EXAMPLE

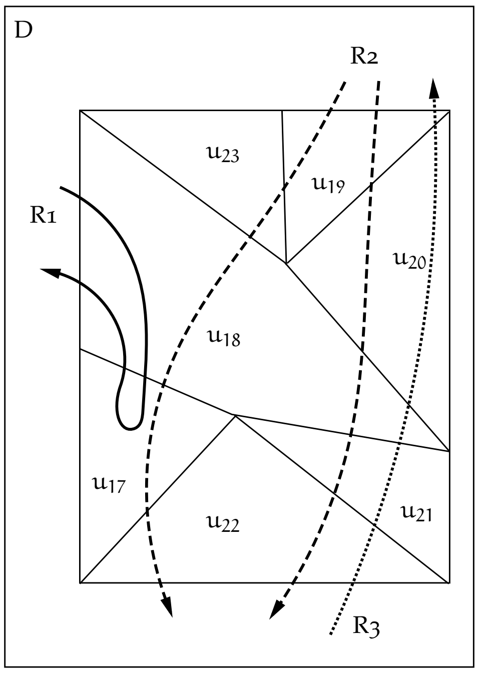
Let us consider a team of three mobile robots evolving in the partitioned environment given in Figure 10.1(a). Initially, the robots are located in a depot D from which they can directly enter any region of the environment. However, once they leave the depot, they are not allowed to return to the depot before completing their tasks. Inside the depot there is no possibility of collisions between the robots. Let us assume that the robots should execute one of the following trajectories previously computed by a path-planning algorithm in order to achieve the task:

- robot R_1: $D \rightarrow u_{18} \rightarrow u_{17} \rightarrow u_{18} \rightarrow D$, meaning that R_1 enters from D to the region u_{18}, then moves to u_{17}, then back to u_{18} and finally exits to D.

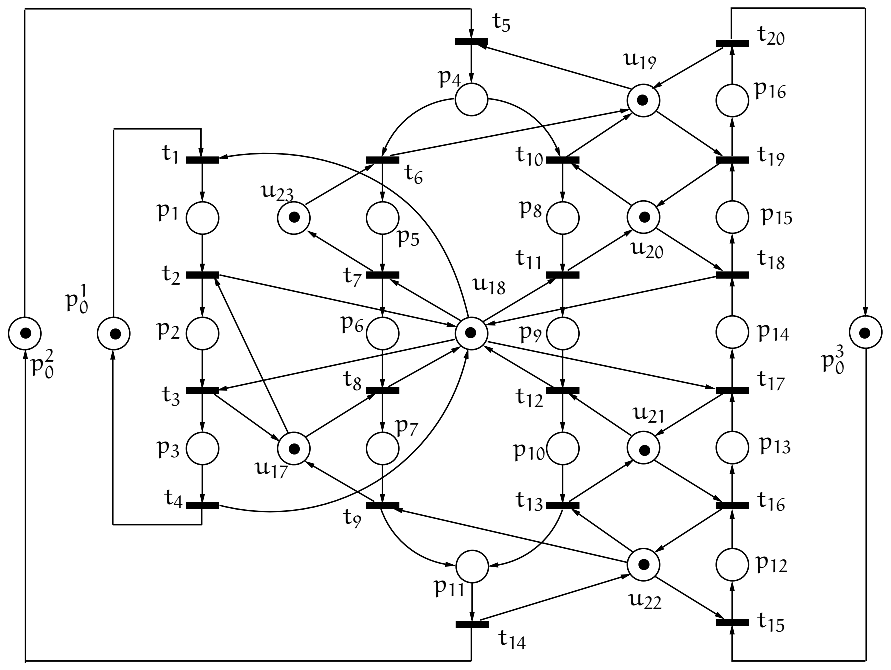
- robot R_2 should follow one of the next two trajectories $D \rightarrow u_{19} \rightarrow u_{23} \rightarrow u_{18} \rightarrow u_{17} \rightarrow u_{22} \rightarrow D$ and $D \rightarrow u_{19} \rightarrow u_{20} \rightarrow u_{18} \rightarrow u_{21} \rightarrow u_{22} \rightarrow D$.

- robot R_3: $D \rightarrow u_{22} \rightarrow u_{21} \rightarrow u_{18} \rightarrow u_{20} \rightarrow u_{19} \rightarrow D$.

By applying the methodology in [38], the S^3PR model of the system is obtained and it is shown in Figure 10.1(b). The approach is to represent every task as a process and capacities as resources. Assuming that each region can contain maximum one robot at any time



(a)



(b)

Figure 10.1: (a) A map in which three robots evolve (R_2 has two possible paths). (b) The S^3PR corresponding to the trajectories of the robots.

Table 10.1: Bad siphons in the S^3PR in Figure 10.1.

i	$\mathcal{S}_i$	$\mathcal{T}(\mathcal{S}_i)$	$\lvert\mathcal{T}(\mathcal{S}_i)\rvert$	m_0	control
1	$u_{21}, u_{22}, p_{11}, p_{13}$	p_{10}, p_{12}	2	2	true
2	$u_{18}, u_{20}, p_1, p_3, p_6, p_9, p_{15}$	p_8, p_{14}	2	2	true
3	$u_{18}, u_{19}, u_{20}, u_{23}, p_1, p_3, p_6, p_9, p_{16}$	$p_4, p_5, p_8, p_{14}, p_{15}$	2	4	false
4	$u_{17}, u_{18}, p_3, p_7, p_9, p_{14}$	p_1, p_2, p_6	2	2	true
5	$u_{17}, u_{18}, u_{20}, p_3, p_7, p_9, p_{15}$	$p_1, p_2, p_6, p_8, p_{14}$	3	3	true
6	$u_{17}, u_{18}, u_{19}, u_{20}, u_{23}, p_3, p_7, p_9, p_{16}$	$p_1, p_2, p_4, p_5, p_6, p_8, p_{14}, p_{15}$	3	5	false
7	$u_{18}, u_{21}, p_1, p_3, p_6, p_{10}, p_{14}$	p_9, p_{13}	2	2	true
8	$u_{18}, u_{20}, u_{21}, p_1, p_3, p_6, p_{10}, p_{15}$	p_8, p_9, p_{13}, p_{14}	2	3	false
9	$u_{18}, u_{19}, u_{20}, u_{21}, u_{23}, p_1, p_3, p_6, p_{10}, p_{16}$	$p_4, p_5, p_8, p_9, p_{13}, p_{14}, p_{15}$	3	5	false
10	$u_{17}, u_{18}, u_{21}, p_3, p_7, p_{10}, p_{14}$	$p_1, p_2, p_6, p_9, p_{13}$	3	3	true
11	$u_{17}, u_{18}, u_{20}, u_{21}, p_3, p_7, p_{10}, p_{15}$	$p_1, p_2, p_6, p_8, p_9, p_{13}, p_{14}$	3	4	false
12	$u_{17}, u_{18}, u_{19}, u_{20}, u_{21}, u_{23}, p_3, p_7, p_{10}, p_{16}$	$p_1, p_2, p_4, p_5, p_6, p_8, p_9, p_{13}, p_{14}, p_{15}$	3	6	false
13	$u_{18}, u_{21}, u_{22}, p_1, p_3, p_6, p_{11}, p_{14}$	$p_9, p_{10}, p_{12}, p_{13}$	2	3	false
14	$u_{18}, u_{20}, u_{21}, u_{22}, p_1, p_3, p_6, p_{11}, p_{15}$	$p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}$	2	4	false
15	$u_{18}, u_{19}, u_{20}, u_{21}, u_{22}, u_{23}, p_1, p_3, p_6, p_{11}, p_{16}$	$p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}, p_{15}$	2	6	false
16	$u_{17}, u_{18}, u_{21}, u_{22}, p_3, p_{11}, p_{14}$	$p_1, p_2, p_6, p_7, p_9, p_{10}, p_{12}, p_{13}$	3	4	false
17	$u_{17}, u_{18}, u_{20}, u_{21}, u_{22}, p_3, p_{11}, p_{15}$	$p_1, p_2, p_6, p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}$	3	5	false
18	$u_{17}, u_{18}, u_{19}, u_{20}, u_{21}, u_{22}, u_{23}, p_3, p_{11}, p_{16}$	$p_1, p_2, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{12}, p_{13}, p_{14}, p_{15}$	3	7	false

moment, the seven resource places $\{u_r | r = 17, \ldots, 23\}$ are modeling the limited capacities of the corresponding regions. Notice that for simplicity we use the same notations for the regions of the environment and for the resource places, i.e., u_r. Assuming the capacity of each region is one, the resource places contain one token in the initial marking since the robots are initially in the depot.

The *process* representing the trajectory of R_1 is modeled by four places, $\{p_0^1, p_1, p_2, p_3\}$, and four transitions, $\{t_1, t_2, t_3, t_4\}$. Place p_0^1 has initially one token since the robot R_1 is waiting in the depot. The other places p_1, p_2 and p_3 are modeling the intermediate regions that should be traveled by R_1. In particular, places p_1 is modeling the presence of R_1 in region u_{18}, p_2 is modeling that the robot is in u_{17} while p_3 is modeling that the robot is again in u_{18}. The firing of the transitions implies that R_1 leaves a region and enters in other region.

It is obvious that the net belongs to S^3PR class since each process place (p_1, p_2 and p_3) *is using* only one resource at any given time. Thus, we can apply the structural results already existing for this class of nets to obtain a deadlock prevention policy. One of such technique is based on the computation of the set of *bad siphons* (a siphon is a set of places such that its set of input transitions is included in its set of output transitions; a siphon is bad if is not containing the support of any P-semiflow, hence it may be emptied) since the existence of such siphons could imply a deadlock state*. In order to ensure a deadlock prevention controller, a new place can be added to the net to prevent the emptiness of the siphon.

The S^3PR in Figure 10.1(b) has 18 bad siphons given in Table 10.1. Some of these siphons can be emptied and some not, depending on the initial marking. Notice that a bad siphon is a structural element of the net. The column "control" indicates whatever the siphon needs to be controlled or not. If a siphon can be emptied, the tokens modeling the resource places initially in the siphon are *allocated* to a set of *thieves places*. The column "$\mathcal{T}(\mathcal{S}_i)$" in Table 10.1 is giving the set of thieves places for each bad siphon; while column "$\mathbf{m}_0$" is showing the initial number of tokens in the bad siphons. Column "$|\mathcal{T}(\mathcal{S}_i)|$" is giving the maximum number of tokens in the thieves and if this number is equal to (or greater than) the initial number of markings of the siphon, the siphon should be controlled because it may be emptied.

In order to prevent the emptiness of the bad siphon $\mathcal{S}_1$, a *monitor place* called c can be introduced to ensure that the thieves places of $\mathcal{S}_1$ can take at most 1 token. Therefore, the initial marking of c will be one and the following arcs will be introduced: (c, t_{12}), (t_{13}, c), (c, t_{15}) and (t_{16}, c). In this case, the place c is necessary for the control and it implies a centralized controller for its implementation such that if R_2 wants to enter in u_{21} it should consult the central unit for an authorization. Moreover, if R_3 will want to enter in u_{22} it should also ask for an authorization from the central unit.

In this chapter we will propose a different approach to control the bad siphons based on *inhibitor arcs*. In order to control $\mathcal{S}_1$, two in-

* In general, finding all siphons is a complex problem. Some papers (e.g., [42]) discuss siphon computation on some subclasses of PN.

hibitor arcs are introduced: (p_{10}, t_{15}) which allow the firing of t_{15} only if p_{10} is empty and (p_{12}, t_{12}) allowing the firing of t_{12} only if p_{12} is empty. Notice that these two inhibitor arcs implement the mutual exclusion condition $\mathbf{m}[p_{10}] + \mathbf{m}[p_{12}] \leqslant 1$. The inhibitor arcs take the replacement of c, meaning no centralized unit is needed. The main advantage of the inhibitor arcs is the fact that they are easier to be implemented. They are simply a checking condition. In the case that the plan of robots may change, the checking condition is the only part needed to be adjusted. Because no control unit is used, the adjustment of checking condition does not introduce cost. In our example, robot R_2 will enter in u_{21} (firing t_{12}) only if there is no robot in u_{22} while R_3 will start entering in u_{22} only if there is no robot in u_{21}.

In Table 10.2 are given all inhibitor arcs to control (thieves of) bad siphons listed in Table 10.1. Due to the introduction of the inhibitor arcs to control the bad siphons, new siphons may be generated. Since these new bad siphons are related to the inhibitor arcs we will call them *virtual siphons*. In this chapter we will characterize these new siphons showing that are very easy to be computed. In Table 10.4 are given the virtual siphons that appear after introducing the inhibitor arcs to control the initial bad siphons.

Finally, all bad siphons and virtual siphons generated by the introduction of the inhibitor arcs are controlled, and then the resulted net which we call S^3PR^2 (an S^3PR with control inhibitor arcs) is live. In the S^3PR^2, no place or arc is added to prevent the system from deadlock states.

10.3 DEADLOCK PREVENTION IN S^3PR

10.3.1 *Liveness of S^3PR*

In order to address the decentralized deadlock prevention problem in the robot planning, we recall some results from [24] on liveness characterization of S^3PR.

Assumption 10.1. *Let $\langle P, T, F\rangle$ be an S^3PR and $i \in I_{\mathcal{N}}$ be an index. The set of places in i-th process is $P_S^i \cup \{p_0^i\}$. In the rest of this chapter, we assume that each process is 1-safe (binary), i.e., let $\langle \mathcal{N}, \mathbf{m}_0\rangle$ be a marked S^3PR, and then for all $\mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0)$ and $\forall i \in I_{\mathcal{N}}$, $\sum_{p \in P_S^i \cup \{p_0^i\}} \mathbf{m}[p] = 1$.*

If a resource token is not in its resource place, then it is held (allocated) by a process. We denote the set of *holders* of resource r (states that use r) by $H(r) = ({}^{\bullet\bullet}r) \cap P_S$.

Example 10.2. *In the S^3PR in Figure 10.2(a), the process places are partitioned into two subsets: $P_S^1 = \{p_1, p_2, p_3, p_4, p_5\}$ and $P_S^2 = \{p_6, p_7, p_8\}$. The set of idle places is $P_0 = \{p_0^1, p_0^2\}$. There are five resource places $\{r_i | i = 1, \ldots, 5\}$. Their holders are $H(r_1) = \{p_1\}$, $H(r_2) = \{p_2, p_8\}$, $H(r_3) = \{p_3, p_7\}$, $H(r_4) = \{p_4, p_6\}$ and $H(r_5) = \{p_5\}$, respectively.*

The characterization of liveness of S^3PR uses P-semiflows. In general PNs, the computation of all minimal P-semiflows is NP-hard [16].

Table 10.2: Control arcs of the S^3PR in Figure 10.1

$\mathcal{S}\vert\mathcal{V}$	inequality	inhibitor arcs
$\mathcal{S}_1$	$\mathbf{m}[p_{10}] + \mathbf{m}[p_{12}] \leqslant 1$	$(p_{10}, t_{15}), (p_{12}, t_{12})$
$\mathcal{S}_2$	$\mathbf{m}[p_8] + \mathbf{m}[p_{14}] \leqslant 1$	$(p_8, t_{17}), (p_{14}, t_{10})$
$\mathcal{S}_4$	$\mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_6] \leqslant 1$	$(p_1, t_7), (p_2, t_7), (p_6, t_1), (p_6, t_2)$
$\mathcal{S}_5$	$\mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_6] + \mathbf{m}[p_8] + \mathbf{m}[p_{14}] \leqslant 2$	–
$\mathcal{S}_7$	$\mathbf{m}[p_9] + \mathbf{m}[p_{13}] \leqslant 1$	$(p_9, t_{16}), (p_{13}, t_{11})$
$\mathcal{S}_{10}$	$\mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_6] + \mathbf{m}[p_9] + \mathbf{m}[p_{13}] \leqslant 2$	–
$\mathcal{V}_1$	$\mathbf{m}[p_8] + \mathbf{m}[p_{13}] \leqslant 1$	$(p_8, t_{16}), (p_{13}, t_{10})$
$\mathcal{V}_2$	$\mathbf{m}[p_9] + \mathbf{m}[p_{12}] \leqslant 1$	$(p_9, t_{16}), (p_{12}, t_{11})$
$\mathcal{V}_3$	$\mathbf{m}[p_8] + \mathbf{m}[p_{12}] \leqslant 1$	$(p_8, t_{15}), (p_{12}, t_{10})$

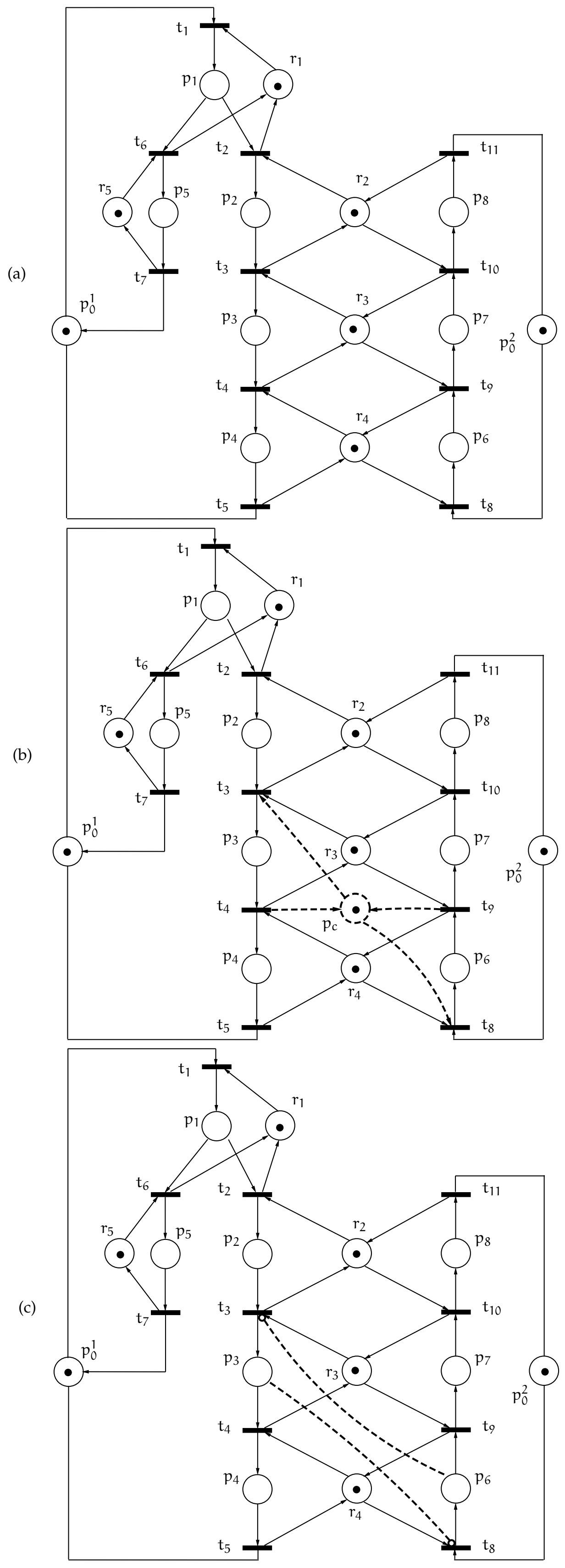


Figure 10.2: (a) An S^3PR with two processes and five resources[24]. (b) The siphon $\mathcal{S}_2$ is controlled using a monitor place p_c. (c) $\mathcal{S}$ is controlled with two inhibitor arcs.

Table 10.3: Set of bad siphons of the S^3PR in Figure 10.2(a)

i	$\mathcal{S}_i$	$\mathcal{S}_i \cap P_R$	$\mathcal{T}(\mathcal{S})$	$\lvert\mathcal{T}(\mathcal{S})\rvert$	$\mathbf{m}_0$
1	r_2, r_3, p_3, p_8	r_2, r_3	p_2, p_7	2	2
2	r_3, r_4, p_4, p_7	r_3, r_4	p_3, p_6	2	2
3	r_2, r_3, r_4, p_4, p_8	r_2, r_3, r_4	p_2, p_3, p_6, p_7	2	3

An improved way to compute all minimal P-semiflow is to use the structural properties of S^3PR. Given a set $X \subseteq P_S \cup P_0 \cup P_R$, by $\mathbf{e}_X$ we denote the $(P_S \cup P_0 \cup P_R)$-indexed vector so that

$$\mathbf{e}_X[p] = \begin{cases} 1, & \text{if } p \in X, \\ 0, & \text{if } p \notin X. \end{cases}$$

Proposition 10.3 (Proposition IV.1 in [24]). *Let $\mathcal{N} = \langle P_S \cup P_0 \cup P_R, T, F\rangle$ be an S^3PR. The set of minimal P-semiflows of $\mathcal{N}$ is $\{e_{P_i \cup P_0} | i \in I_{\mathcal{N}}\} \cup \{e_{H(r)\cup\{r\}} | r \in P_R\}$.*

Proposition 10.3 indicates that in S^3PR, a minimal P-semiflow containing resource places are composed of the resource places and their holders. The number of minimal P-semiflows in an S^3PR is $|I_{\mathcal{N}}| + |P_R|$.

Remark 10.4. *In an S^3PR, for each $p \in P_R$, there exists a unique minimal P-semiflow $\mathbf{y}_p$ such that $\|\mathbf{y}_p\| \cap P_R = \{p\}$ and $\mathbf{y}_p[p] = 1$.*

Definition 10.5. *A siphon $\mathcal{S}$ is called* bad *if there does not exist any P-semiflow $\mathbf{y}$ such that $\|\mathbf{y}\| \subseteq \mathcal{S}$.*

A bad siphon can be emptied. Obviously, when it is emptied, all transitions in its input and output sets are dead.

Definition 10.6. *Let $\mathcal{N}$ be an S^3PR and $\mathcal{S}$ be a siphon. The* thieves *of $\mathcal{S}$ are $\mathcal{T}(\mathcal{S}) = (\cup_{r \in \mathcal{S} \cap P_R} \|\mathbf{y}_r\|) \setminus \mathcal{S}$, where $\mathbf{y}_r$ is a P-semiflow containing only one resource place r.*

The thieves of a siphon consist of the places where tokens can go if they leave the siphon†. According to processes, we partition the thieves $\mathcal{T}(\mathcal{S})$ of a siphon $\mathcal{S}$ into several subsets: $\mathcal{T}_i(\mathcal{S}) = \mathcal{T}(\mathcal{S}) \cap P_S^i$, $i \in I_{\mathcal{N}}$. We use $|\mathcal{T}(\mathcal{S})|$ to denote the number non-empty subsets in the partition. Since each process is binary, each subset of the thieves can hold at most one token. The maximal number of tokens that can be held by the thieves is $|\mathcal{T}(\mathcal{S})|$.

Example 10.7. *Let us consider the PN in Figure 10.2(a) whose bad siphons are shown in Table 10.3. The thieves of $\mathcal{S}_2$ are p_3 and p_6. Because they are distributed in two processes, the maximal number of tokens that can be held is 2.*

† A place in the set of thieves is called a *thief*.

10.3.2 *Decentralized Control of Siphons*

Depending on m_0, not every bad siphon can be emptied. Before controlling a bad siphon, we first verify whether it needs to be controlled or not. When a siphon is emptied, all the resource tokens are in its thieves. The number of resource tokens in a siphon $\mathcal{S}$ is $\sum_{p \in \mathcal{S} \cap P_R} m_0[p]$ and the maximal number of tokens that the thieves can hold is $|\mathcal{T}(\mathcal{S})|$. If $\sum_{p \in \mathcal{S} \cap P_R} m_0[p] > |\mathcal{T}(\mathcal{S})|$, then there will always be tokens in the siphon (the siphon cannot be emptied). In this case, the siphon needs not to be controlled.

Otherwise, the thieves must be controlled to reduce the number of tokens that can be held. In order to do this, at most $\sum_{p \in \mathcal{S} \cap P_R} m_0[p] - 1$ tokens can be held by the thieves, while the maximum number of tokens that can be held by the thieves is $|\mathcal{T}(\mathcal{S})|$. To control the siphon, we reduce the maximum number of tokens that can go to the thieves from $|\mathcal{T}(\mathcal{S})|$ to $\sum_{p \in \mathcal{S} \cap P_R} m_0[p] - 1$ by applying the mutual exclusion condition to $k \leqslant |\mathcal{T}(\mathcal{S})|$ subsets of thieves. Therefore, for every reachable marking, the k subsets of thieves can hold at most one token, and the other subsets can hold $|\mathcal{T}(\mathcal{S})| - k$ tokens. The number of tokens that can be held by the thieves becomes $|\mathcal{T}(\mathcal{S})| - k + 1$. It implies the condition that $|\mathcal{T}(\mathcal{S})| - k + 1 \leqslant \sum_{p \in \mathcal{S} \cap P_R} m_0[p] - 1$, and then the minimal number of subsets of thieves that should be in mutual exclusion is:

$$k \geqslant |\mathcal{T}(\mathcal{S})| - \sum_{p \in \mathcal{S} \cap P_R} m_0[p] + 2. \tag{10.6}$$

It means that if we apply the mutual exclusion condition to k subsets of thieves, where k is the minimal number computed by (10.6), then the siphon will never be emptied. In order to control k subsets $\mathcal{T}_1(\mathcal{S}), \ldots, \mathcal{T}_k(\mathcal{S})$ of $\mathcal{T}(\mathcal{S})$, we add inhibitor arcs from $p \in \mathcal{T}_i(\mathcal{S})$ to ${}^\bullet p'$, $p' \in \mathcal{T}_j(\mathcal{S})$, where $i \neq j$, $i, j = 1, \ldots, k$.

Example 10.8. *Continuing Example 10.7, we compute the inhibitor arcs to control $\mathcal{S}_2$. The initial number of resource tokens in $\mathcal{S}_2$ is 2 and there are $|\mathcal{T}(\mathcal{S})| = 2$ and $\sum_{p \in \mathcal{S} \cap P_R} m_0[p] = 2$. The mutual exclusive condition has to be implied among $2 - 2 + 2 = 2$ thieves. Because the thieves of $\mathcal{S}_2$ are $\{p_3\}$ and $\{p_6\}$, then both p_3 and p_6 have to be controlled. In order to apply mutual exclusion to p_3 and p_6, we add two inhibitor arcs from p_3 to t_8 and from p_6 to t_3, respectively, and the resulted net is shown in Figure 10.2(c). After that, the thieves of $\mathcal{S}_1$ and $\mathcal{S}_3$ are controlled and the net in Figure 10.3(a) is obtained.*

Two siphons $\mathcal{S}$ and $\mathcal{S}'$ may share some thieves, i.e.$\mathcal{T}(\mathcal{S}) \cap \mathcal{T}(\mathcal{S}') \neq \emptyset$. In this case, the inhibitor arcs applied on the thieves of one of them also control the other one. Consider the siphons and inhibitor arcs in Example 10.8. The siphon $\mathcal{S}_3$ shares thieves with the other two siphons, while the other siphons do not share thieve with each other. For example, p_2 is a thief of both $\mathcal{S}_2$ and $\mathcal{S}_3$. The inhibitor arcs controlling $\mathcal{S}_1$ and $\mathcal{S}_2$ also control $\mathcal{S}_3$. The inhibitor arcs of $\mathcal{S}_1$ and $\mathcal{S}_2$ implies that for every reachable marking m, $m[p_2] + m[p_7] \leqslant 1$ and $m[p_3] + m[p_6] \leqslant 1$. Due to the inhibitor arcs, the minimal the

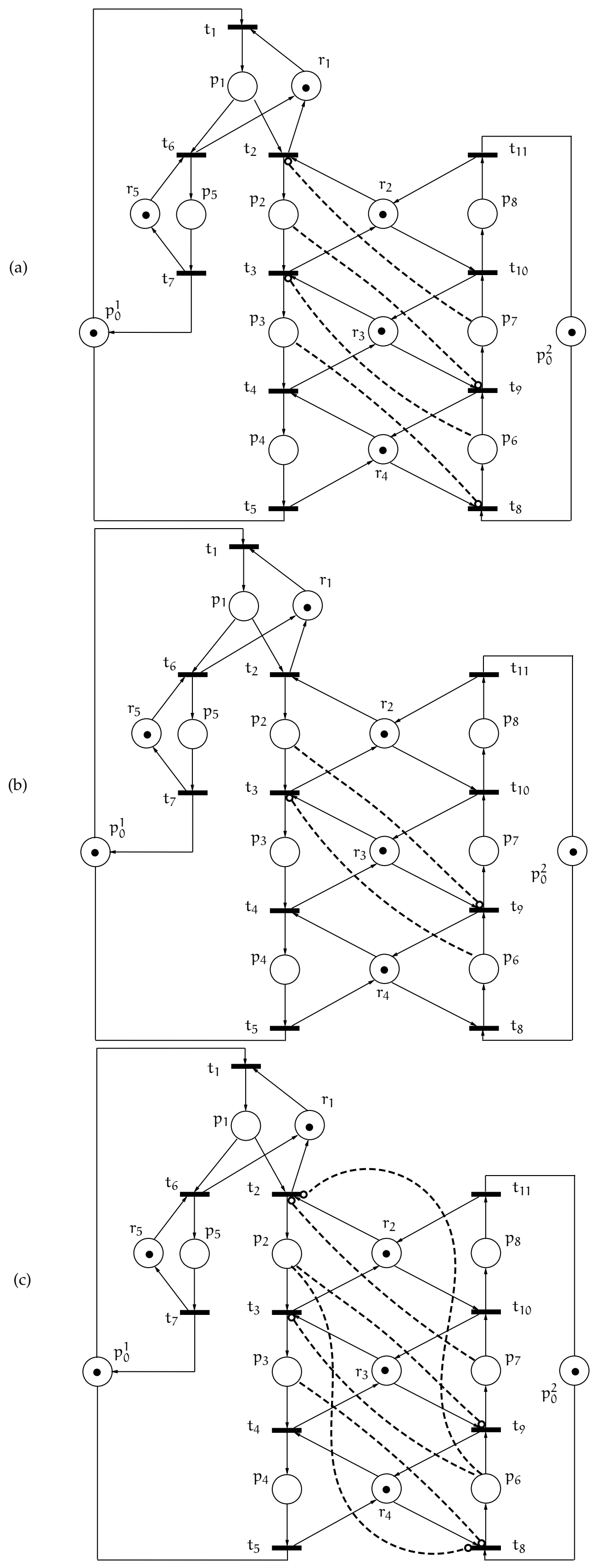


Figure 10.3: (a) All siphons are controlled. (b) Two inhibitor arcs introduce a virtual siphon. (c) All siphons and the only virtual siphon is controlled.

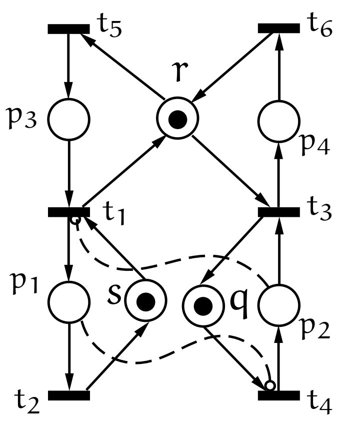


Figure 10.4: Because the two dashed inhibitor arcs are not used to control siphons, the PN is not an S^3PR^2.

number of tokens in $\mathcal{S}_3$ is 1. It implies that $\mathcal{S}_3$ cannot be emptied and no control will be applied to $\mathcal{S}_3$.

10.4 DEADLOCK PREVENTION IN S^3PR^2

10.4.1 *The Class of S^3PR^2*

In the previous section, we proposed a decentralized control policy on S^3PR for the robot planning problem: using this policy, no centralized controller is needed. Nevertheless, the controlled net (an S^3PR with inhibitor arcs) does not belong to the class of S^3PR. In order to discuss the liveness of the controlled net, we extend the definition of S^3PR.

Definition 10.9. *An* S^3PR with Reading arcs *(S^3PR^2) is a PN* $\mathcal{N} = \langle P = P_S \cup P_R \cup P_0, T, F \cup F_r \rangle$ *obtained from an S^3PR by adding inhibitor arcs to control bad siphons, where:*

1. $\langle P, T, F \rangle$ *is a S^3PR;*

2. $F_r \subseteq P_S \times T$ *are inhibitor arcs (they can be transformed to reading arcs using complementary places) such that if* $\exists (p, t) \in F_r$*, then* $\forall t' \in {}^\bullet(t^\bullet)$*, there is* $(p, t') \in F_r$*;*

3. $\forall (p, t) \in F_r$*, if* (p, t) *is removed from* $\mathcal{N}$*, then* $\exists \mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0)$*,* $\mathbf{m}_0 \notin \mathcal{R}(\mathcal{N}, \mathbf{m})$.

Let us consider the PN in Figure 10.4. The two inhibitor arcs (p_2, t_1) and (p_1, t_4) do not control any bad siphon. Therefore, it is not an S^3PR^2. In fact, due to the two inhibitor arcs, a new deadlock is created comparing with the original net without the inhibitor arcs. Observe that by firing the sequence $t_4 t_5$, the system reaches a deadlock.

Note that by applying the control policy proposed in Section 10.3.2, if there is an inhibitor arc from a place p to one input transition of another place p', then there are inhibitor arcs from p to *all* input transitions of p'.

Definition 10.10. *Let $\mathcal{N}^r$ be an S^3PR^2 and $\mathcal{N}$ be the underlying S^3PR. A marking $\mathbf{m}_0$ is an acceptable initial marking of $\mathcal{N}^r$ if $\mathbf{m}_0$ is an acceptable initial marking of the underlying S^3PR.*

Lemma 10.11. *Let $\mathcal{N}^r$ be an S^3PR^2 and $\mathcal{N}$ be the underlying S^3PR. A set of places is a siphon in $\mathcal{N}^r$ iff it is a siphon in $\mathcal{N}$.*

Proof. *Let a set $\mathcal{S} \subseteq P_S$, where P_S is the set of process places in both $\mathcal{N}^r$ and $\mathcal{N}$. The pre and post matrices of $\mathcal{N}^r$ are the same as the ones of $\mathcal{N}$. If in $\mathcal{N}$ there is ${}^\bullet\mathcal{S} \subseteq \mathcal{S}^\bullet$, then ${}^\bullet\mathcal{S} \subseteq \mathcal{S}^\bullet$ holds also in $\mathcal{N}^r$ and vice versa.* □

10.4.2 *Virtual Siphon*

As discussed in Section 10.3.1, the deadlocks in S^3PR appear when a bad siphon is emptied. In S^3PR^2, because inhibitor arcs are added, another kind of "siphon" appears, which can be seen if we introduce the *complementary* places.

Let us consider the PN in Figure 2.11(a), which is a subnet of an S^3PR whose processes are binary. Because the upper bounds of the numbers of tokens in p_1 and p_2 are 1, two implicit places are added in the PN to represent the usable capacities of them (Figure 2.11(b)). Those implicit places are called the *complementary places* of p_1 and p_2, respectively. By adding the complementary places, the resulted PN has the same behavior as the original one. Let $P = \{p_1, p_2\}$ and $P' = \{p'_1, p'_2\}$. There are ${}^\bullet P' = \{t_2, t_3\}$ and $P'^\bullet = \{t_1, t_4\}$. At this moment, P' is not a siphon. Let us add two inhibitor arcs from p_1 and p_2 to t_3 and t_2, respectively, so that the net in Figure 2.11(c) is obtained and it is a subnet of an S^3PR^2. Using the complementary places, we can transform it into the one in Figure 2.11(d), in which the arcs from p'_1 to t_3 and p'_2 to t_2 are reading arcs. The two reading arcs make t_2 and t_3 become output transitions of the places in P', i.e., $P'^\bullet = \{t_1, t_2, t_3, t_4\}$. Due to the reading arcs, P' becomes a siphon, i.e., ${}^\bullet P' \subset P'^\bullet$. In the nets in Figure 2.11(c) and 2.11(d), we can reach a deadlock by firing the sequences $t_1 t_4$ or $t_4 t_1$. The deadlock is caused, because the siphon P' composed of the complementary places of P is emptied. Due to the fact that the places in P' does not exist in the S^3PR^2, we call it the set of *complementary places* of P. Since the set P' is also siphon, we call the set P a *virtual siphon*.

Definition 10.12. *Let $\mathcal{N}^r$ be an S^3PR^2. A* virtual siphon *is a set of places such that the set of their complementary places is a siphon. We call a virtual siphon $\mathcal{V}$ as a* minimal virtual siphon, *if $\nexists p$ such that $\mathcal{V} \setminus \{p\}$ is a virtual siphon.*

Definition 10.13. *Let $\mathcal{N}^r$ be an S^3PR^2. Considering set of places $P' = \{p'_1, p'_2, \ldots, p'_k\}$, a* virtual cycle *composed of the places in P' is $p'_1 \xrightarrow{e_1} p'_2 \xrightarrow{e'_i} \cdots \xrightarrow{e_{i-1}} p'_i \xrightarrow{e'_i} p'_{i+1} \overset{e_{i+1}}{\cdots} p'_k \xrightarrow{e'_k} p'_1$, where $p'_j \xrightarrow{e_j} p'_{j+1}$ means there is one arc $(p'_j, t_j) \in F$ and one inhibitor arc $(p'_{j+1}, t_j) \in F_r$.*

We denote the virtual cycle in Definition 10.13 as $p'_1 p'_2 \ldots p'_k$. With virtual cycle, next proposition is used to find a minimal virtual siphon by computing a *minimal cyclic path*, which does not contain any other cyclic path, such that the places in it construct the minimal virtual siphon.

Proposition 10.14. *Let $\mathcal{N}$ be an S^3PR^2. A set of places $\mathcal{V} = \{p_1^{\mathcal{V}}, p_2^{\mathcal{V}}, \ldots, p_k^{\mathcal{V}}\} \in P_S$ is a minimal virtual siphon iff there exists a minimal cyclic path $p_1 p_2 \ldots p_k$ ($p_i \in \mathcal{V}, i = 1, \ldots, k$, $p_i \neq p_j$, if $i \neq j, i, j = 1, \ldots, k$) such that for all $i \in \{1, \ldots, k-1\}$, there are inhibitor arcs from p_i to all $p_{i+1}{}^{\bullet}$ and from p_k to all $p_1{}^{\bullet}$.*

Proof. *We can see that $\|p_1 p_2 \ldots p_k\| = \mathcal{V}$.*

$\Leftarrow$) Let $\mathcal{V}' = \{p_i' | i = 1, \ldots, k\}$ be the set of complementary places of the ones in $\mathcal{V}$ such that p_i' is the complementary place of p_i. Therefore, $p_i{}^{\bullet} \subseteq {}^{\bullet}p_i'$ and ${}^{\bullet}p_i \subseteq p_i'{}^{\bullet}$ for all $p_i \in \mathcal{V}$ and $p_i' \in \mathcal{V}'$.

On the other hand, the inhibitor arcs from $p_i \in \mathcal{V}$ to all transitions $p_{i+1}{}^{\bullet}$ are equivalent to reading arcs from p_i' to all $p_{i+1}{}^{\bullet}$ for all $i = 1, \ldots, k-1$. If $i = k$, then the inhibitor arcs from p_k to all transitions $p_1{}^{\bullet}$ are equivalent to reading arcs from p_k' to all $p_1{}^{\bullet}$. Therefore, $p_{i+1}{}^{\bullet} \subseteq p_i'{}^{\bullet}$ and $p_{i+1}{}^{\bullet} \subseteq {}^{\bullet}p_i'$ for all $i = 1, \ldots, k-1$ and $p_1{}^{\bullet} \subseteq p_k'{}^{\bullet}$ and $p_1{}^{\bullet} \subseteq {}^{\bullet}p_k'$.

Summing up, ${}^{\bullet}p_i' = p_i{}^{\bullet} \cup p_{i+1}{}^{\bullet}$, $i = 1, \ldots, k-1$ and ${}^{\bullet}p_k' = p_k{}^{\bullet} \cup p_1{}^{\bullet}$ implying ${}^{\bullet}\mathcal{V}' = \mathcal{V}^{\bullet}$. On the other hand, $p_i'{}^{\bullet} = {}^{\bullet}p_i \cup p_{i+1}{}^{\bullet}$, $i = 1, \ldots, k-1$ and $p_k'{}^{\bullet} = {}^{\bullet}p_k \cup p_1{}^{\bullet}$ implying $\mathcal{V}'^{\bullet} = {}^{\bullet}\mathcal{V} \cup \mathcal{V}^{\bullet}$. It means that $\mathcal{V}'$ is a siphon and $\mathcal{V}$ is a virtual siphon.

$\Rightarrow$) Assume $\mathcal{V}$ is a virtual siphon but there exists a place $p_i \in \mathcal{V}$ such that there exists no inhibitor arc from p_i to a transition $t \in p_{i+1}{}^{\bullet}$. Let us consider $i \leqslant k-1$ in this proof and it can be easily extended to the case when $i = k$. Let $p_{i+1}' \in \mathcal{V}'$ be the complementary place of p_{i+1}, hence $t \in {}^{\bullet}p_{i+1}'$ which implies that $t \in {}^{\bullet}\mathcal{V}'$. Because there is no inhibitor arc from p_i to t, then there is no reading arc from p_i' to t. Therefore, $t \in {}^{\bullet}\mathcal{V}'$ and $t \notin \mathcal{V}'^{\bullet}$ implying that ${}^{\bullet}\mathcal{V}' \nsubseteq \mathcal{V}'^{\bullet}$. Hence $\mathcal{V}'$ is not a siphon and $\mathcal{V}$ is not a virtual siphon.

If $\mathcal{V}$ is a minimal virtual siphon and it corresponds to a non-minimal cyclic path, which contains another cyclic path, then the contained cyclic path corresponds to a virtual siphon instead $\mathcal{V}$. It means that $\mathcal{V}$ is not a minimal virtual siphon. Therefore, if $\mathcal{V}$ is a minimal virtual siphon, then it corresponds to a minimal cyclic path, and vice versa. □

10.4.3 *Liveness of S^3PR^2*

In order to characterize the liveness of S^3PR^2 (following Definition 10.9), we use complementary places to transform an S^3PR^2 to an S^4PR and use the results of S^4PR in [16].

Let $\mathcal{N}^r = \langle P_0 \cup P_S \cup P_R, T, F \cup F_r \rangle$ be an S^3PR^2. The S^4PR net $\mathcal{N}^c$ with the same behavior as $\mathcal{N}^r$ is obtained by applying Algorithm 10.1.

Note that this is the iterative application of the procedure of removing inhibitor arcs in a bounded PN.

Remark 10.15. *Let $\mathcal{N}^r$ be an S^3PR^2 and $\mathcal{N}^c$ be corresponding transformed S^4PR. An initial marking $\mathbf{m}_0$ is acceptable for $\mathcal{N}^c$ if it is acceptable for $\mathcal{N}^r$.*

The S^4PR is a superclass of S^3PR. The liveness of S^4PR has been studied [16] and the following result has been proposed.

Algorithm 10.1 Transform an S^3PR^2 to a S^4PR

```
1: Let $\mathcal{N}^c = \langle P_0 \cup P_S \cup P_R, T, F\rangle$ be the underlying $S^3PR$ of $\mathcal{N}^r = \langle P_0 \cup P_S \cup P_R, T, F \cup F_r\rangle$.
2: Let $P_C = \emptyset$, $F_c = \emptyset$
3: for each $(p, t) \in F_r$ do
                    ▷ for each inhibitor arc, add the complementary place $p'$ of $p$
4:   $P_C = P_C \cup \{p'\}$
5:   $\forall t' \in p^\bullet$, let $F_c = F_c \cup \{(t', p')\}$
6:   $\forall t' \in {}^\bullet p$, let $F_c = F_c \cup \{(p', t)\}$
7:   $F_c = F_c \cup \{(p', t), (t, p')\}$            ▷ add the reading arc
8: end for
9: $\mathcal{N}^c = \langle P_0 \cup P_S \cup P_R \cup P_C, T, F \cup F_c\rangle$
```

Theorem 10.16 (Theorem 3 in [16]). *Let $\langle \mathcal{N}, \mathbf{m}_0\rangle$ be a marked S^4PR with $\mathcal{N} = \langle P_0 \cup P_S \cup P_R, T, F\rangle$. The net is non-live iff there exists a siphon $\mathcal{D}$ and a marking $\mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m}_0)$ such that:*

C1) $\|\mathbf{m}\| \cap P_S \neq \emptyset$;

C2) $\|\mathbf{m}\| \cap (P_S \setminus \mathcal{T}(\mathcal{D})) = \emptyset$;

C3) $\forall p \in \mathcal{T}(\mathcal{D})$ *such that* $\|\mathbf{m}\| \cap p \neq \emptyset$, *the firing of each* $t \in p^\bullet$ *is prevented by a set of resource places belonging to* $\mathcal{D}$.

Theorem 10.16 indicates that, in the liveness of S^4PR, siphons containing resource places play an important role. An S^4PR system is non-live if there is a reachable marking $\mathbf{m}$ satisfying that exists a siphon $\mathcal{D}$ and:

(C1 and C2) if a resource token is not in the resource place, then it is in a thief of $\mathcal{D}$;

(C3) if a token is in a thief p of $\mathcal{D}$, then it cannot leave the thief (to move to its resource place) by firing any transition $t \in {}^\bullet p$, because t is not enabled forever due to an input resource place of t which is empty.

Lemma 10.17 (Lemma 2 in [16]). *Let $\mathcal{N}$ be an S^4PR and $\mathcal{D} \subseteq P$ be a non-empty minimal siphon of $\mathcal{N}$. If $\mathcal{D} \cap P_R \neq \emptyset$, then $\mathcal{D}$ is the unique minimal siphon of $\mathcal{N}$ containing exactly the set of resources $\mathcal{D} \cap P_R$.*

The time complexity to compute all minimal siphons in an S^4PR is exponential with the number of resource places [16, 20, 41, 64]. However, we do not have to control all of them and only a subset (called *basic minimal siphons*) has to be controlled.

Definition 10.18. *Let $\mathcal{N}^r = \langle P_0 \cup P_S \cup P_R, T, F\rangle$ be an S^3PR^2 and $\mathcal{N}^c = \langle P_0 \cup P_S \cup P_R \cup P_C, T, F \cup F_c\rangle$ be its transformed S^4PR net by applying Algorithm 10.1. The set of* basic minimal siphons $S \cup V$ *contains:*

1. *the set of minimal siphons S containing resource places in P_R and processes places in P_S;*

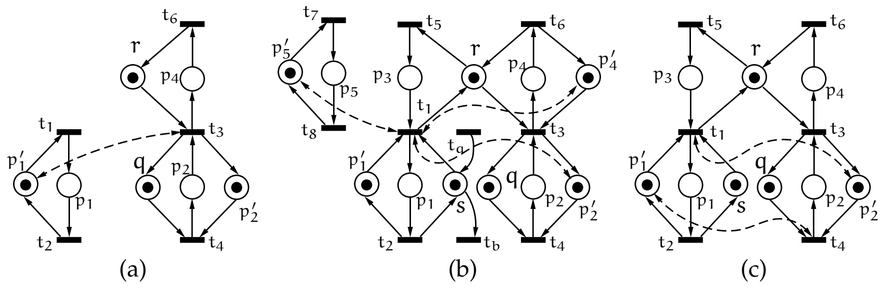


Figure 10.5: (a) Possibilities of transforming t_3 to be an output transition of $\mathcal{S}$. (b) Possibilities of transforming t_1 to be an output transition of $\mathcal{S}$. (c) p_1 and p_2 are in mutual exclusive condition.

2. *the set of siphons* V *such that for any* $\mathcal{V}' \in V$, $\mathcal{V}'$ *corresponds to a virtual siphon in* $\mathcal{N}^r$.

Let $\mathcal{N}^r = \langle P_0 \cup P_S \cup P_R, T, F \rangle$ be an S^3PR^2 and $\mathcal{N}^c = \langle P_0 \cup P_S \cup P_R \cup P_C, T, F \cup F_c \rangle$ be its transformed S^4PR net by applying Algorithm 10.1. A virtual siphon $\mathcal{V}$ in $\mathcal{N}^r$ corresponds to a siphon in $\mathcal{N}^c$ composed only by places in P_C. If all basic minimal siphons are controlled then the other siphons of $\mathcal{N}^c$ which contain places from P_R and from P_C are already controlled.

Lemma 10.19. *Let* $\mathcal{N}^r = \langle P_0 \cup P_S \cup P_R, T, F \rangle$ *be an* S^3PR^2 *and* $\mathcal{N}^c = \langle P_0 \cup P_S \cup P_R \cup P_C, T, F \cup F_c \rangle$ *be its transformed* S^4PR *net by applying Algorithm 10.1. If all basic minimal siphons of* $\mathcal{N}^r$ *are controlled (using inhibitor/reading arcs) then all minimal bad siphons are controlled.*

Proof. *Let* $\mathcal{S}$ *be a minimal bad siphon containing places from both sets* P_R *and* P_C *and we will show that* $\mathcal{S}$ *has been controlled by inhibitor arcs. From hypothesis,* $\exists p_2' \in P_C$ *such that* $p_2' \in \mathcal{S}$. *Assume, without loss of generality that* p_2' *has only one single input and one single output transition, e.g.,* ${}^\bullet p_2' = \{t_3\}$, $p_2'^\bullet = \{t_4\}$. *The following steps will be performed to obtain all places are in* $\mathcal{S}$:

1. *Because* $\mathcal{S}$ *is a siphon, the input transition* t_3 *of* p_2' *should be also an output transition. Hence, an input place in* t_3 *should belong to* $\mathcal{S}$. *There are three different possibilities (see Figure 10.5(a)): (1a) place* $p_2 \in P_S$, *such that* p_2' *is the complementary place of* p_2. *However,* p_2 *cannot belong to* $\mathcal{S}$, *because the siphon will not be bad including the p-semiflow* $p_2 + p_2'$; *(1b) a place from the set* P_C. *If this is the case, step 1 is re-iterated starting with the input transition in this new complementary place. Since the number of complementary places is finite, the number of iteration is finite. On the other hand, since* $\mathcal{S}$ *is not containing only complementary places at the end other type of place (not of type (b)) should be found; (1c) a resource place* $r \in P_R$. *Let* $p_4 \in P_S$ *be a holder place of* r *belonging to the same process as place* p_2 *(Figure 10.5(a)). Now, the input transition of* r, *e.g.,* t_6, *should be also an output transition of* $\mathcal{S}$. *A new resource or complementary place could be considered but in this case step 1 is reiterated until a process place will be considered. Assume, without loss of generality that* $p_4 \in$

$\mathcal{S}$. Resource r cannot be private and used only in p_4, otherwise $\mathcal{S}$ will contain a p-semiflow $r + p_4$.

2. *Now, we have $\{p'_2, p_4, r\} \subseteq \mathcal{S}$. An input transition $t_1 \neq t_6$ of r must be an output transition of $\mathcal{S}$. There are five possibilities (see Figure 10.5(b)). (2a) if a reading arc exists between p'_4 and t_1, then one output transition of p'_4 is t_1. But $\mathcal{S}$ will have a p-semiflow containing p_4 and p'_4. (2b), a reading arc exists between p'_5 and t_1, where p'_5 the a complementary place. But we face the same problem as we had in the previous step that the input transition of p'_5 must be an output transition of $\mathcal{S}$, and eventually $\mathcal{S}$ must have a place belonging to other types. (2c), a resource place s, which has t_1 as one output transition, can belong to $\mathcal{S}$. However, with s, we repeat the problem in this step that other input transitions of s must be output transitions of $\mathcal{S}$, and eventually $\mathcal{S}$ contains places in other types. (2d), p'_1 can belong to $\mathcal{S}$, because its output transition is t_1, but its input transition t_2 must be an output transition of $\mathcal{S}$ and we repeat the problem in the previous step. (2e), p'_2 belongs to $\mathcal{S}$ if a reading arc exists between p'_2 and t_1. From the fact that $\mathcal{S}$ contains p'_2 and the control policy that no mutual exclusive condition is implied between places in the same process, we obtain that t_1 belongs to other process than the one containing p_2.*

Until now, we have $\mathcal{S} = \{p'_2, p_4, r\} \cup P'$. The set of places $\{p'_2, p_4, r\}$ is a minimal siphon and any additional place to this set makes it to be non-minimal. Therefore, $P' = \emptyset$ and $\mathcal{S} = \{p'_2, p_4, r\}$ (otherwise $\mathcal{S}$ is not minimal). The set of thieves of $\mathcal{S}$ is $\mathcal{T}(\mathcal{S}) = \{p_2, p_3\}$. The reading arc between p'_2 and t_1 implies that p_1 and p_2 are in mutual exclusion and another reading arc exists between p'_1 and t_4. These reading arcs may be used to control thieves of a siphon in the S^3PR (e.g., p_1 and p_2 are thieves of $\mathcal{S}'$), or to control a virtual siphon. In both uses, p_1 and p_2 are thieves of one or two siphons.

First, consider the inhibitor arcs used to control a siphon in the S^3PR such that $\{p_1, p_2\} \subseteq \mathcal{T}(\mathcal{S}')$. We can see that p_2 is a holder of q. Because a thief of $\mathcal{S}'$ is a holder of a resource in $\mathcal{S}'$ and p_2 is a thief of $\mathcal{S}'$, so $q \in \mathcal{S}'$. The transition t_3 is an output transition of $\mathcal{S}'$ and the only possibility (in the S^3PR) is that $r \in \mathcal{S}'$. It means that $p_3 \in \mathcal{T}(\mathcal{S}')$. Second, consider the inhibitor arcs control a virtual siphon such that $p_1 \in \mathcal{T}(\mathcal{S}')$ and $p_2 \in \mathcal{T}(\mathcal{S}'')$. We have $r \in \mathcal{S}'$ and $p_3 \in \mathcal{T}(\mathcal{S}')$. Because $p_3 \in \mathcal{T}(\mathcal{S}')$, so $s \in \mathcal{T}(\mathcal{S}')$ meaning that as in the previous case, a siphon $\mathcal{S}'$ contains s, q, r and $\{p_1, p_2, p_3\} \subseteq \mathcal{T}(\mathcal{S}')$. In both cases, in order to control $\mathcal{S}'$, we add inhibitor arcs (p_2, t_5) and (p_3, t_4). Hence, by our control policy, we have inhibitor arcs (p_2, t_5) and (p_3, t_4) to control $\mathcal{S}''$. These inhibitor arcs also control p_3 and p_2 for $\mathcal{S}$. Finally, $\mathcal{S}$ has been controlled when we control $\mathcal{S}'$. □

Now, we give Theorem 10.20 to characterize the liveness of S^3PR^2 such that the system is non-live if, for a reachable marking, there exists an emptied siphon or a fully marked virtual siphon.

Theorem 10.20. *Let $\langle \mathcal{N}^r, m_0 \rangle$ be an S^3PR^2 where $\mathcal{N}^r = \langle P_0 \cup P_S \cup P_R, T, F \cup F_r \rangle$, and the basic minimal siphons are $S \cup V$. The net is non-live iff $\exists m \in \mathcal{R}(\mathcal{N}^r, m_0)$, $\exists$ a minimal siphon $\mathcal{S} \in S$ such that $\mathcal{S} \cap \|m\| = \emptyset$ or $\exists$ a virtual siphon $\mathcal{V} \in V$ such that $\mathcal{V} \cap \|m\| = \mathcal{V}$.*

Proof. *Let $\mathcal{N}^c = \langle P_0 \cup P_S \cup P_R \cup P_C, T, F \cup F_c \rangle$ be the transformed S^4PR net of $\mathcal{N}^r$, where $\forall p' \in P_C$, p' is the complementary place of $p \in P_S$ obtained by applying Algorithm 10.1. We prove the theorem in $\mathcal{N}^c$. The two conditions are:*

C1) $\exists \mathcal{S}$ such that $\mathcal{S}$ is emptied;

C2) $\exists \mathcal{V}$ such that $\mathcal{V}$ is fully marked.

By transforming $\mathcal{V}$ to the set of its complementary places $\mathcal{V}'$, the condition C2 can be converted to $\exists \mathcal{V}'$ such that $\mathcal{V}'$ is emptied. As both $\mathcal{S}$ and $\mathcal{V}'$ are siphons in $\mathcal{N}^c$, then the theorem is rewritten as follows: the net is non-live iff $\exists \mathbf{m} \in \mathcal{R}(\mathcal{N}^c, \mathbf{m}_0)$ such that $\exists$ an empty siphon $\mathcal{D}$ in $\mathcal{N}^c$. According to Theorem 10.16 and Lemma 10.19, it holds that if there is a reachable marking at which there exists an emptied siphon in $\mathcal{N}^c$, and then $\mathcal{N}^c$ is non-live. Finally, the theorem holds. □

10.4.4 *Control of Virtual Siphons*

A fully marked virtual siphon leads the system into deadlock states. Therefore, control a virtual siphon is to ensure that there is always one empty place in it (cannot be fully marked forever). The control policy is implemented by inhibitor arcs such that, by synthesizing the inhibitor arcs, the resulted model still remains in the class of S^3PR^2. When an inhibitor arc is added into an S^3PR^2, new virtual siphons may be introduced. Therefore, iterative searching and controlling for virtual siphons are needed.

Before discussing how to control virtual siphons, let us first consider whether a virtual siphon needs to be controlled or not. Some virtual siphons do not need to be controlled, if they can never be fully marked, i.e., for any reachable marking at least one place is empty. These virtual siphons can be characterized using the PN structure.

Remark 10.21. *Let $\mathcal{N}^r$ be an S^3PR^2. Since every process is binary, a virtual siphon $\mathcal{V}$ does not need to be controlled if $\exists i \in I_{\mathcal{N}^r}$ such that $|\mathcal{V} \cap P_S^i| \geqslant 2$, i.e., there exists two places in $\mathcal{V}$ belonging to the same process (they are mutual exclusive to each other).*

In order to find virtual siphons, the cycles containing inhibitor arcs are computed. The complexity of computing all cyclic paths is exponential. Lemma 10.22 indicates the maximal lengths of virtual siphons, which need to be controlled, and it decreases the complexity for the computation for virtual siphons in an S^3PR^2.

Lemma 10.22. *Let $\langle \mathcal{N}^r, \mathbf{m}_0 \rangle$ be a marked S^3PR^2 and $I_{\mathcal{N}^r}$ be the set of indices. The virtual siphon $\mathcal{V}$ does not needs to be controlled if $|\mathcal{V}| > |I_{\mathcal{N}^r}|$.*

Proof. *If $|\mathcal{V}| > |I_{\mathcal{N}^r}|$, then $\exists i \in I_{\mathcal{N}^r}$, $|\mathcal{V} \cap P_S^i| \geqslant 2$ and $\mathcal{V}$ do not need to be controlled.* □

The number of places in a virtual siphon $\mathcal{V}$ satisfies $2 \leqslant |\mathcal{V}| \leqslant |I_{\mathcal{N}^r}|$. In order to prevent a virtual siphon from being fully marked, we apply the mutual exclusive condition by using inhibitor arcs to two

Table 10.4: Virtual siphons in the controlled S^3PR in Figure 10.1

i	$\mathcal{V}_i$
1	p_8, p_{13}
2	p_9, p_{12}
3	p_8, p_{12}

places in the virtual siphon. New virtual siphons may be constructed due to the newly introduced inhibitor arcs. The computation of control of virtual siphons has to be performed iteratively until no new virtual siphon rises. Moreover, some inhibitor arcs do not raise any virtual siphon, and then we do not consider them in the computation of virtual siphons.

Lemma 10.23. *Let $\langle \mathcal{N}^r, m_0 \rangle$ be a marked S^3PR^2 and t be a transition. The inhibitor arcs pointing to t do not introduce any virtual siphon, if there is no inhibitor arc starting from all ${}^\bullet t$.*

Proof. *Let t be an output transition of p_{i+1} and exists an inhibitor (p_i, t) from p_i to t. According to Proposition 10.14, in order to construct a virtual siphon, there must be inhibitor arcs starting from p_{i+1}. Hence, if there is no inhibitor arc starting from p_{i+1}, the inhibitor arc (p_i, t) cannot raise any virtual siphon.* □

Because all inhibitor arcs in S^3PR^2 start from process places and no inhibitor arc starting from idle places, the inhibitor arcs pointing to the first transition of each process, which is the output transition of idle place, can be removed in the computation of virtual siphons.

Example 10.24. *Let us continue Example 10.7, in which siphons are controlled and the resulted PN is shown in Figure 10.3(a). We first remove the inhibitor arcs, which do not raise any virtual siphon, and the resulted net is shown in Figure 10.3(b). To control all siphons, four inhibitor arcs are applied, and two of them may introduce virtual siphons. There is only one virtual siphon $\mathcal{V} = \{p_2, p_6\}$. To control $\mathcal{V}$, we add two inhibitor arcs from p_2 to t_8 and from p_6 to t_2, respectively, to the net in Figure 10.3(c). Because the newly added inhibitor arcs do not raise any virtual siphon, the controlled system is live.*

10.4.5 *Comparison*

We compare the S^3PR^2 with related works using the benchmark S^3PR shown in Figure 10.1(b) with a new initial marking $m_0 = 3p_0^1 + 11p_0^2 + 7p_0^3 + 2u_{17} + u_{18} + u_{19} + u_{20} + u_{21} + u_{22} + u_{23}$. In order to apply inhibitor arcs, we convert the non-safe S^3PR to a safe one. It is made by duplicating processes. For example, the process with p_0^1 is duplicated to three processes, because there are three tokens in it. The generated three processes contain one token each. Comparison results are shown in Table 10.5, where the columns in the table are:

1. "marking" contains the numbers of markings can be reached in the live system;

Table 10.5: Comparison with related works

	markings	monitors	arcs	inhibitor arcs	complexity
[24]	6287	18	106	0	Exponential
[40]	6287	6	32	0	Exponential
[61]	14850	8	40	0	NP
[18]	21562	19	112	0	Exponential
S^3PR^2	6287	0	0	6896+	Exponential

2. "monitors"composes of the numbers of additional control places;
3. the numbers of arcs used to attach the control places with the original system are in "arc";
4. "inhibitor arcs" consists the numbers of additional inhibitor arcs controlling the original system;
5. the complexity of the corresponding approaches are listed in "complexity".

It is illustrated that the numbers of markings in the live systems produced by [24], [40] and ours are the same. The approach in [18] has the maximal permission among the five approaches. The live system generated by applying [18] has the largest number of monitor places and arcs, followed by the system in [24]. In monitor based approaches, [40] proposed the best result in the number of places and arcs. Because we propose a monitor place-free approach, the numbers of control places and arcs are zero. As a trade-off, we use a large number of inhibitor arcs. As we illustrated, when a robot wants to enter a region, it sends inquiries according to the inhibitor arcs. It means that:

1. the original system is not modified;
2. because there is no monitor place, no monitor state will be maintained.

11 ROBOT PLAN VERIFICATION

In the case where real time control is not applied, we introduce some algorithms to verify robots' plans before they are executed. Our target is to avoid some forbidden states and our solution is to introduce initial delays to robot plans. The initial delays mean, first, each robot must wait for a given time delay before starting to move, and second, when it is moving ,it follows its plan, while no additional control will be introduced. We use three different approaches to determine the delays. Each approach is represented using a mixed integer linear programming problem. We also introduce statistical analysis on experimental results of the three approaches.

11.1 INTRODUCTION

This chapter addresses a collision avoidance problem in a multi-robot system. Each robot has a set of possible trajectories, each trajectory fulfilling its individual task. The trajectories consist of sequences of regions to be followed, and the time for moving inside each region is known. We considered the problem of imposing initial time delays for trajectories of mobile robots, such that a collision-free movement results in a shared environment. Each robot has a set of possible Two solutions are developed, depending on the possibility of imposing a certain trajectory from the available set of paths for each robot. The solutions have the form of mixed integer linear programming optimizations that return the initial time delays and, when necessary, the chosen trajectory for each robot. Finally, we perform a statistical study on the proposed solutions and we conclude that one formulation is preferable to the others.

11.2 PROBLEM DESCRIPTION

Some preliminary ideas and notations to be used are given in subsection 11.2.1, while the problems to be solved are formulated in subsection 11.2.2.

11.2.1 *Preliminaries*

A set of $|R|$ robots moving in a static and known environment is considered. Based on partitioning techniques [21] and control results for classes of dynamical systems [32, 6], the environment and the control capabilities of each robot can be modeled by various types of discrete event systems. Details on such abstraction procedures go beyond the scope of this section, and the interested reader is referred to works as [19, 7, 37] for more details.

The set of robots is denoted by $R = \{r_1, r_2, \dots, r_{|R|}\}$, and the map of the environment is represented by the graph $G = (M \cup \{D\}, E)$, where:

- nodes from set M correspond to regions (places) from the partitioned environment, where the robots move in order to complete their tasks;
- node D correspond to a depot where the robots are initially placed and where they will return after moving through nodes from M;
- edges from $E \subseteq \{M \cup \{D\}\} \times \{M \cup \{D\}\}$ correspond to possible movement of robots between adjacent regions, i.e., $(p_1, p_2) \in E$, with $p_1, p_2 \in M \cup \{D\}$, means that a robot starting from region denoted by p_1 can arrive in finite time to place p_2, without crossing through any other places.

The depot D should be regarded as an area where the robots are handled (e.g.stored, charged, programmed) by a central unit. While the robots are in depot D, they cannot collide. However, once the robots leave D and evolve in regions from M, they move independently, without communicating with other robots or with the central unit. Therefore, it is possible that robots collide in nodes from M, and this work is devoted to avoiding such collisions by imposing some initial delays in robot movements.

Each robot r_i, $i = 1, 2, \dots, |R|$, has a set of trajectories $\mathcal{T}_i = \{s_{i,1}, s_{i,2}, \dots, s_{i,|\mathcal{T}_i|}\}$. Each trajectory $s_{i,j}$ is a sequence of nodes such that $s_{i,j} = D p^1_{i,j}\, p^2_{i,j} \cdots p^{n_{i,j}}_{i,j} D$ and $\forall k = 1, \dots, n_{i,j}$, $p^k_{i,j} \in M$. Each robot r_i should follow any trajectory from $\mathcal{T}_i$ in order to accomplish its task: it leaves the depot by accessing the map nodes M, moves through regions from M and returns to D.

Let $p^k_{i,j}$ be the k-th place (excluding depot) in the j-th trajectory of robot r_i. A crossing time $\tau^k_{i,j}$ is associated with $p^k_{i,j}$ to describe the time needed by r_i from entering region $p^k_{i,j}$ until leaving it.

Note that if the same place from M belongs to multiple trajectories, its associated crossing times may be different, fact in accordance with a model build by partitioning the robotic environment. For example, assume that place $p \in M$ appears in trajectories j and j' of robot r_i: $p = p^k_{i,j} = p^{k'}_{i,j'}$. In j-th trajectory, p is crossed by entering the region from a previous region $p^{k-1}_{i,j}$ and by moving towards next region $p^{k+1}_{i,j}$, *e.g.* by following a line segment or by applying a specific control law. The moving time $\tau^k_{i,j}$ is therefore related to this local continuous trajectory, and it may be different from $\tau^{k'}_{i,j'}$, since in j'-th trajectory region p may be reached (left) from (to) other region by different local control laws and trajectories. On the same ideas, if robots have different speeds, this information is also encapsuled by the crossing times related to each tuple robot-trajectory-place.

There is no crossing time associated with D, meaning that any robot r_i can immediately access M. If all robots start to move at the initial time 0, it is possible that some robots collide, because their trajectories may intersect. Let us assume that two robots collide if they are in the same region from M at the same time, or if they swap two adjacent regions (one moves from p to p' while the other moves from p' to p). Clearly, collisions depend on the intersections of trajectories of different robots and on the crossing times for regions along these trajectories.

Recall that the robots independently move, and therefore they cannot pause their movement in order to avoid collisions and yield crossing priorities to other robots. Even if such pauses were possible, deadlocks may appear in the case of circular waits of some robots [38]. However, since the robots are initially in the depot, where they are handled by a central unit, it is possible to delay the starting time for moving along each possible trajectory, such that no collisions appear. If robot r_i follows trajectory $s_{i,j}$, let us denote by $\delta_{i,j}$ the time delay when r_i starts to move.

Example 11.1. *Consider the graph environment from Figure 11.1, in which* M *includes nine places and the depot that surrounds these regions. There are two robots* r_1 *and* r_2*, each having a single trajectory:*

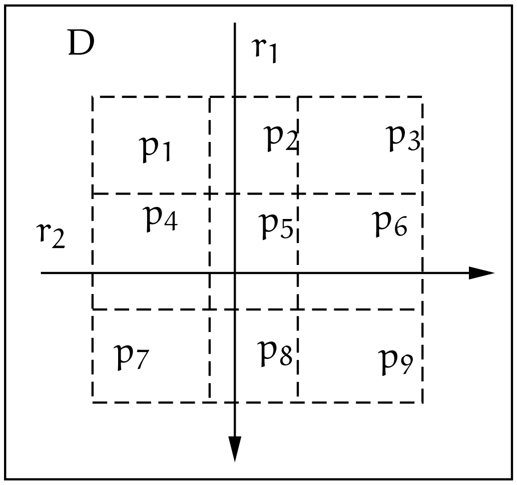


Figure 11.1: Map for example 11.1.

- $\mathcal{T}_1 = \{s_{1,1}\}$, $s_{1,1} = Dp^1_{1,1}p^2_{1,1}p^3_{1,1}D = Dp_2p_5p_8D$;
- $\mathcal{T}_2 = \{s_{2,1}\}$, $s_{2,1} = Dp^1_{2,1}p^2_{2,1}p^3_{2,1}D = Dp_4p_5p_6D$.

The following crossing times are assumed:

- *for* $s_{1,1}$: $\tau^1_{1,1} = 1$, $\tau^2_{1,1} = 2$, $\tau^3_{1,1} = 1$;
- *for* $s_{2,1}$: $\tau^1_{2,1} = 2$, $\tau^2_{2,1} = 1$, $\tau^3_{2,1} = 1$.

If the initial delays of two trajectories are zero ($\delta_{1,1} = \delta_{2,1} = 0$), then the arrival and departure times of r_1 in $p^2_{1,1}$ (region p_5 in the map) are 1 and 3, respectively, while the arrival and departure times of r_2 in $p^2_{2,1}$ (same region p_5) are 2 and 3, respectively. Because both robots are in the place p_5 in the time interval $[2, 3]$, they collide.

11.2.2 *Problem statement*

Our goal is to find initial time delays $\delta_{i,j}$ such that there are no robot collisions and the robots finish their tasks in minimum possible time. A trivial solution for enforcing these delays would be to sequentially move the robots, i.e.first move only robot r_1, when it finishes its trajectory start moving r_2, and so on. Clearly, such an approach forbids simultaneous movements and it implies a long time until each robot finishes its chosen trajectory. In a resource allocation framework, nodes M can be regarded as a shared resource for the $|R|$ robots, while the trivial solution implies a mutual exclusion for accessing the set M.

Two different problems are formulated depending on the ability of the central unit to impose the trajectory of each robot r_i from set $\mathcal{T}_i$.

Problem 11.2 (Decentralized). *Each robot r_i chooses its trajectory from set $\mathcal{T}_i$ without informing the cental unit. Find initial time delays $\delta_{i,j}$, $i = 1, \ldots, |R|$, $j = 1, \ldots, |\mathcal{T}_i|$, such that there are no collisions and all robots finish the movement in shortest time.*

Problem 11.3 (Centralized). *The central unit can impose a trajectory for each robot r_i from set $\mathcal{T}_i$. For each robot r_i, $i = 1, \ldots, |R|$, find the trajectory*

$s_{i,j} \in \mathcal{T}_i$ *it should follow and its initial time delay* $\delta_{i,j}$, *such that there are no collisions and all robots finish the movement in shortest time.*

The common hypothesis for both problems are the sets of trajectories $\mathcal{T}_i$ and the crossing times of robots through regions $\tau^k_{i,j}$, $i = 1, \dots, |R|$, $j = 1, \dots, |\mathcal{T}_i|$, $k = 1, \dots, n_{i,j}$.

Problem 11.2 will have as outcomes a number of $\sum_{i=1}^{|R|} |\mathcal{T}_i|$ initial time delays. Once these are available, each robot r_i (randomly) chooses a trajectory $s_{i,j}$ from $\mathcal{T}_i$, waits for time $\delta_{i,j}$ and then it starts to evolve along $s_{i,j}$.

Problem 11.3 will have a total of $2|R|$ outcomes: the index j for trajectory and the initial time delay $\delta_{i,j}$, for each robot r_i, $i = 1, \dots, |R|$. The trajectories and initial delays imply a minimum completion time (until all robots return to depot D). Of course, the completion time yielded by solution of Problem 11.2 may be longer, since it is minimized by accounting any possible robot-trajectory choice.

Remark 11.4. *The above problems can be formulated by ignoring depot* D *and by assuming that each robot is initially deployed in a place from map* M. *The same procedure can be followed for finding initial time delays. However, in such a scenario it is possible that the above problems become infeasible (e.g., assume two robots each with a single trajectory, the trajectories consist of the same regions from* M, *but are followed in opposite directions by the robots). If depot* D *is assumed, the problems always have solutions (see for example the trivial solution from the beginning of this subsection).*

As possible real scenarios mimicked by the above formulation, one can imagine non-communicating exploring robots, air transportation systems, automated railway systems where specific track segments and intersections correspond to nodes from M. Initial delays rather than pausing motions may be desirable in certain situations (e.g., in air transportation it is cheaper to let planes wait in their departure airport than wait in air, while in railway systems paused trains may perturb other traffic participants).

Example 11.5. *In Example 11.1, a possible solution for avoiding collisions can be* $\delta_{1,1} = 2 + \epsilon$ *(with* ϵ *a very small value) and* $\delta_{2,1} = 0$, *case in which* r_1 *finishes its trajectory at time* $6 + \epsilon$ *(and* r_2 *at time 4). A better solution would be* $\delta_{1,1} = 0$ *and* $\delta_{2,1} = 1 + \epsilon$, *such that the robots are back in depot at time* $5 + \epsilon$.

11.3 SOLUTION

This section proposes some algorithmic solutions for Problems 11.2 and 11.3 based on MILP (Mixed Integer Linear Programming) optimization.

11.3.1 *A solution for Problem 11.2*

If a robot r_i follows trajectory $s_{i,j}$, its arrival time in region $p^k_{i,j}$ is

$$T_A(p^k_{i,j}) = \delta_{i,j} + \sum_{l=1}^{k-1} \tau^l_{i,j}; \tag{11.7}$$

where $\delta_{i,j}$ is the initial delay (unknown). The departure time from $p^k_{i,j}$ is

$$T_D(p^k_{i,j}) = T_A(p^k_{i,j}) + \tau^k_{i,j}. \tag{11.8}$$

Consider two trajectories $s_{i,j}$ and $s_{\alpha,\beta}$ of two robots r_i and r_α, respectively. Assume that the two trajectories intersect in a place $p \in M$, $p = p^k_{i,j} = p^\gamma_{\alpha,\beta}$ (k-th region from trajectory of robot r_i is identical with γ-th region from trajectory of robot r_α). The robots do not collide in this place if either inequality Equation 11.9a or Equation 11.9b is true.

$$\begin{cases} T_D(p^k_{i,j}) \leqslant T_A(p^\gamma_{\alpha,\beta}) - \epsilon & (11.9a) \\ T_D(p^\gamma_{\alpha,\beta}) \leqslant T_A(p^k_{i,j}) - \epsilon & (11.9b) \end{cases}$$

In inequalities Equation 11.9, ϵ is a very small number. It ensures that robots r_i and r_α will never be on the same border of a region at the same time, such that they cannot collide by swapping places (if $p^{k+1}_{i,j} = p^{\gamma-1}_{\alpha,\beta}$, or $p^{k-1}_{i,j} = p^{\gamma+1}_{\alpha,\beta}$). The above inequalities mean that the robots do not collide in place $p = p^k_{i,j} = p^\gamma_{\alpha,\beta}$ if either r_i departs p before r_α arrives there Equation 11.9a, or viceversa Equation 11.9b.

The satisfaction of one inequality from Equation 11.9 will be enforced by appropriate values for delays $\delta_{i,j}$ and $\delta_{\alpha,\beta}$. The *disjunction* from Equation 11.9 can be transformed into a *conjunction* of inequalities Equation 11.10, by a so-called big number method [31]. Let N be a large number, and define a binary variable $b_{ijk,\alpha\beta\gamma}$ such that $b_{ijk,\alpha\beta\gamma} = 0$ if Equation 11.9a holds, and $b_{ijk,\alpha\beta\gamma} = 1$ if Equation 11.9b is true. There is no collision in $p = p^k_{i,j} = p^\gamma_{\alpha,\beta}$ if inequalities from Equation 11.10 simultaneously hold.

$$\begin{cases} T_D(p^k_{i,j}) - T_A(p^\gamma_{\alpha,\beta}) \leqslant N \cdot b_{ijk,\alpha\beta\gamma} - \epsilon \\ T_D(p^\gamma_{\alpha,\beta}) - T_A(p^k_{i,j}) \leqslant N \cdot (1 - b_{ijk,\alpha\beta\gamma}) - \epsilon \end{cases} \tag{11.10}$$

Recall that in scenario of Problem 11.2, every robot r_i will choose a trajectory to follow from its set of trajectories $\mathcal{T}_i$. Since it cannot be determined in advance which trajectory will be followed by each robot, it is necessary to compute initial time delays of all trajectories so that no matter which ones are chosen, the system will be collision free.

By using in Equation 11.10 the expressions of arrival and departure times, the constraints are obtained from Equation 11.11.

$$\begin{cases} T_D(p_{i,j}^k) - T_A(p_{\alpha,\beta}^\gamma) \leqslant N \cdot b_{ijk,\alpha\beta\gamma} - \epsilon \\ T_D(p_{\alpha,\beta}^\gamma) - T_A(p_{i,j}^k) \leqslant N \cdot (1 - b_{ijk,\alpha\beta\gamma}) - \epsilon \\ \qquad \forall i, \alpha \in \{i = 1, \dots, |R|\},\ i \neq \alpha,\ j \in \{1, \dots, |\mathcal{T}_i|\}, \\ \qquad \beta \in \{1, \dots, |\mathcal{T}_\alpha|\},\ k \in \{1, \dots, n_{i,j}\}, \\ \qquad \gamma \in \{1, \dots, n_{\alpha,\beta}\},\ \text{s.t. } p_{i,j}^k = p_{\alpha,\beta}^\gamma. \end{cases} \tag{11.11}$$

In order to solve Problem 11.2, the completion time when all robots are returned to depot is minimized, i.e.,

$$\min\left(\max_{i,j}\ T_A(p_{i,j}^{n_{i,j}})\right) \tag{11.12}$$

The min-max problem Equation 11.12 with constraints Equation 11.11 can be transformed into the standard MILP formulation Equation 11.13 by introducing an auxiliary variable y for the completion time.

$$\begin{aligned} &\min(y) \\ \text{s.t.} &\begin{cases} T_A(p_{i,j}^{n_{i,j}}) \leqslant y \\ \text{constraints Equation 11.11} \end{cases} \end{aligned} \tag{11.13}$$

MILP Equation 11.13 can be solved by using optimization software packages [45]. It has $1 + \sum_{i=1}^{|R|} |\mathcal{T}_i|$ free (real) variables (completion time and initial time delays) and some binary variables $b_{ijk,\alpha\beta\gamma}$ (their number depending on how many intersections exist between trajectories of different robots).

Example 11.6. *The MILP Equation 11.13 corresponding to the system in Ex. 11.1 is:*

$$\begin{aligned} &\min(y) \\ s.t. &\begin{cases} \delta_{1,1} + 4 & \leqslant y \\ \delta_{2,1} + 4 & \leqslant y \\ (\delta_{1,1} + 3) - (\delta_{2,1} + 2) & \leqslant N b_{112,212} - \epsilon \\ (\delta_{2,1} + 3) - (\delta_{1,1} + 1) & \leqslant N(1 - b_{112,212}) - \epsilon \\ 0 & \leqslant \delta_{1,1}, \delta_{2,1} \\ b_{112,212} & \in \{0, 1\}. \end{cases} \end{aligned} \tag{11.14}$$

The solution of the MILP is: $\delta_{1,1} = 0$, $\delta_{2,1} = 1$, $b_{112,121} = 1$ *and* $y = 5$.

11.3.2 *A solution for Problem 11.3*

Let us extend the solution from subsection 11.3.1 to the case when the trajectory of each robot is chosen by a central unit, not by the robot. In order to implement this policy, binary variables $x_{i,j}$ are introduced for the j-th trajectory of the i-th robot. If $x_{i,j} = 1$, then r_i will follow

$s_{i,j}$; otherwise, $s_{i,j}$ will not be followed by r_i. Therefore, $\sum_{j=1}^{|\mathcal{T}_i|} x_{i,j} = 1$, $\forall i = 1, \dots, |R|$. Constraints from optimization Equation 11.13 become Equation 11.15, where the constraints (Equation 11.15a) consider only completion time for the chosen trajectories.

$$\begin{cases} \begin{array}{rll} T_A(p_{i,j}^{n_{i,j}}) - y & \leqslant N \cdot (1 - x_{i,j}) & (a) \\ \sum_{j=1}^{|\mathcal{T}_i|} x_{i,j} & = 1 & (b) \\ T_D(p_{i,j}^k) - T_A(p_{\alpha,\beta}^\gamma) & \leqslant N \cdot b_{ijk,\alpha\beta\gamma} - \epsilon & (c) \\ T_D(p_{\alpha,\beta}^\gamma) - T_A(p_{i,j}^k) & \leqslant N \cdot (1 - b_{ijk,\alpha\beta\gamma}) - \epsilon & (d) \\ \multicolumn{3}{l}{\forall i, \alpha \in \{i = 1, \dots, |R|\},\ i \neq \alpha,\ j \in \{1, \dots, |\mathcal{T}_i|\},} \\ \multicolumn{3}{l}{\beta \in \{1, \dots, |\mathcal{T}_\alpha|\},\ k \in \{1, \dots, n_{i,j}\},} \\ \multicolumn{3}{l}{\gamma \in \{1, \dots, n_{\alpha,\beta}\},\ \text{s.t. } p_{i,j}^k = p_{\alpha,\beta}^\gamma.} \end{array} \end{cases} \tag{11.15}$$

Constraints Equation 11.15 basically mean:

1. central unit chooses one trajectory for each robot by controller, via variables $x_{i,j}$;
2. all possible trajectories are conflict free (as in Equation 11.13), not only the chosen ones.

The second item can be relaxed by guaranteeing conflict free movements only for the chosen trajectories. The constraints (Equation 11.15c) and (Equation 11.15d) are modified based on variables $x_{i,j}$ and Equation 11.16 is obtained.

$$\begin{cases} \begin{array}{rll} T_A(p_{i,j}^{n_{i,j}}) - y & \leqslant N \cdot (1 - x_{i,j}) & (a) \\ \sum_{j=1}^{|\mathcal{T}_i|} x_{i,j} & = 1 & (b) \\ T_D(p_{i,j}^k) - T_A(p_{\alpha,\beta}^\gamma) & \leqslant N \cdot b_{ijk,\alpha\beta\gamma} + & \\ & +N \cdot (1 - x_{i,j}) + & \\ & +N \cdot (1 - x_{\alpha,\beta}) - \epsilon \cdot x_{i,j} & (c) \\ T_D(p_{\alpha,\beta}^\gamma) - T_A(p_{i,j}^k) & \leqslant N \cdot (1 - b_{ijk,\alpha\beta\gamma}) + & \\ & +N \cdot (1 - x_{i,j}) + & \\ & +N \cdot (1 - x_{\alpha,\beta}) - \epsilon \cdot x_{\alpha,\beta} & (d) \\ \multicolumn{3}{l}{\forall i, \alpha \in \{i = 1, \dots, |R|\},\ i \neq \alpha,\ j \in \{1, \dots, |\mathcal{T}_i|\},} \\ \multicolumn{3}{l}{\beta \in \{1, \dots, |\mathcal{T}_\alpha|\},\ k \in \{1, \dots, n_{i,j}\},} \\ \multicolumn{3}{l}{\gamma \in \{1, \dots, n_{\alpha,\beta}\},\ \text{s.t. } p_{i,j}^k = p_{\alpha,\beta}^\gamma.} \end{array} \end{cases} \tag{11.16}$$

Under minimization of y, each set of constraints Equation 11.15 or Equation 11.16 yield a MILP optimization that solves Problem 11.3. Notice that the number of constraints of MILP Equation 11.16 is bigger than the number of constraints of MILP Equation 11.15. However, according to the statistical analysis in the next section, the computational time is smaller.

Remark 11.7. *The solutions of both problems use MILP. It is NP-hard to solve MILP. However, the time delays can be obtained by solving the MILP off-line.*

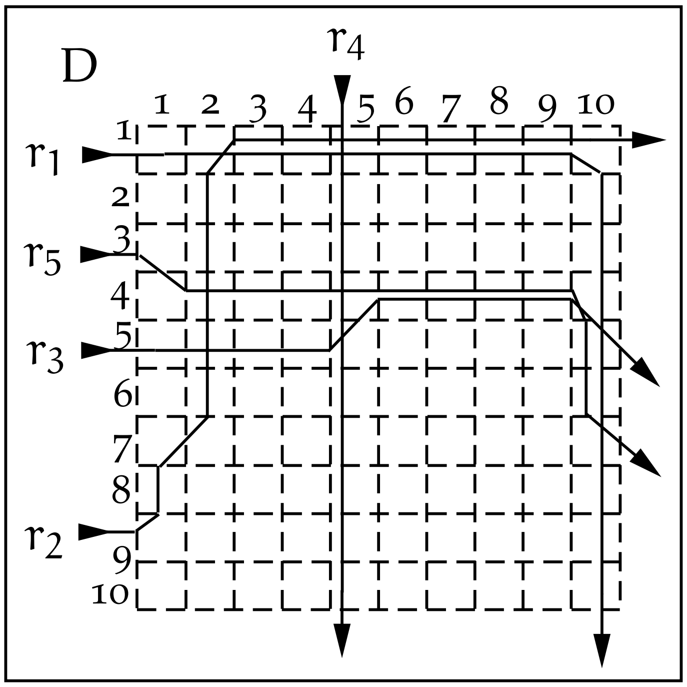


Figure 11.2: A 10×10 map with five robots.

11.4 EXAMPLE AND STATISTICAL STUDY

The solutions from Section 11.3 are compared using five robots moving in a map sketched in Figure 11.2, which includes 100 regions and a depot D. Robots can move from one region to another one following directions *up*, *down*, *left* and *right*. For example, a robot can move from $(3,4)$ to $(3,5)$, but never to $(4,5)$. Each robot should enter in the environment from the external depot D in a specific region, should reach a final region and then leave to D. Let us assume the following requirements for the robots:

- r_1: enters in $(1,1)$ and reaches $(10,10)$,
- r_2: enters in $(9,1)$ and reaches $(1,10)$,
- r_3: enters in $(5,1)$ and reaches $(5,10)$,
- r_4: enters in $(1,5)$ and reaches $(10,5)$,
- r_5: enters in $(3,1)$ and reaches $(7,10)$.

Let us consider first only one trajectory for each robot, trajectories which are shown in Figure 11.2. Assume that the time of crossing a region is 1 time unit if the robot enters in the region from left (right) and leaves to right (left) or enters from top (bottom) and leaves to bottom (top). Otherwise, a smaller duration equals to $\frac{\sqrt{2}}{2}$ time unit is applied. MILP Equation 11.13 has been solved in 0.0088 seconds on a computer with Intel 2.3GHz CPU and the following delays for the robot trajectories are obtained: $\delta_{1,1} = 0$, $\delta_{2,1} = 0.2959$, $\delta_{3,1} = 5.7111$, $\delta_{4,1} = 4.7081$ and $\delta_{5,1} = 4.2949$, respectively. Notice that, if r_3 and r_5

do not wait and start moving at time 0 then they will meet in region $p_{4,5}$ and could collide during the time interval $[4.4707, 5.4142]$. Without delay, r_3 goes through regions $p_{5,1}p_{5,2}p_{5,3}p_{5,4}p_{5,5}$ and reaches $p_{4,5}$ in $1+1+1+1+\frac{\sqrt{2}}{2} = 4.7071$ time units. It will leave $p_{4,5}$ at time $4.7071 + \frac{\sqrt{2}}{2} = 5.4142$. The robot r_5 would arrive at $p_{4,5}$ at 4.4142 and would leave at 5.4142. Therefore, during the time interval $[4.4707, 5.4142]$ both r_3 and r_5 are in $p_{4,5}$ and they could collide. If both robots are delayed according to the solution of MILP Equation 11.13 the collision will not be possible anymore.

Let us now consider other trajectories for the robots but assuming the same requirements (the same initial and final regions as before). In particular, three trajectories per robot are randomly chosen from a number of 30 trajectories obtained by applying K-shortest paths algorithm [70]. Moreover, with the assumption that the crossing time of regions is randomly generated in the interval $[1, 5]$, 100 replicas of the experiment have been conducted to analyze the the computation time to find the optimal solution.

The summary of experiment results is shown in Table 11.1 where the mean and the standard deviation are given. The last three columns correspond to the optimal cost. However, for the same set of trajectories and environment, the optimal cost of MILP Equation 11.16 is always smaller or equal to the optimal cost of MILP Equation 11.15 which is smaller than equal to the optimal cost of MILP Equation 11.13.

MILP Equation 11.13 gives a solution for Problem 11.2. For solving Problem 11.3, one can use MILP Equation 11.15 or Equation 11.16, but the user can also use the solution for Problem 11.2 (MILP Equation 11.13) where the trajectory for each robot is chosen to be the one finishing in shortest time. Thus, the three cases for solving Problem 11.3 in the comparison are further denoted by: case 1 (MILP Equation 11.13), case 2 (MILP Equation 11.15), case 3 (MILP Equation 11.16)). The comparisons are with respect to the computational time using paired one sided *t-test*[*] on the following hypothesis:

- $H_0^{i,j}$: Case i is not better than case j w.r.t. the computation time, $(i, j) \in \{(3, 1), (2, 1), (3, 2)\}$;

- $H_A^{i,j}$: Case i is better than case j w.r.t. the computation time, $(i, j) \in \{(3, 1), (2, 1), (3, 2)\}$.

Hence, three tests are performed and denoted as $\text{test}^{i,j}$.

Using the data obtained by the experiments in Table 11.1 and the values from Student's t-distribution corresponding to a significant interval of 95%, for each test, a *p-value* is computed [26]. For example, in the case of the hypothesis $H_0^{3,1}$ and $H_A^{3,1}$, Two groups of data d_1 and d_2 represent the computation time of the 100 simulations of case 1 and case 3, respectively. The p-values are computed using the means and standard deviations of d_1 and d_2. If the p-value is greater than $1 - 0.95 = 0.05$, where 0.95 is the significant interval, then the alterna-

[*] A t-test is a statistical hypothesis test to determine if two sets of data are significantly different from each other [26].

Table 11.1: Summary of experiment results

	Computation time (unit: second)					Cost		
	Case 1	Case 2	Case 2 without outliers	Case 3	Case 3 without outliers	Case 1	Case 2	Case 3
mean	0.9280	44.3755	1.6819	43.3963	0.1564	64.6410	56.0113	53.2203
standard deviation	0.3214	170.3796	2.7315	172.7728	0.1240	5.7776	5.3267	5.7691

Table 11.2: T-tests results with 95% significant level

test	p-value	accepted	rejected	conclusion
$\text{test}^{3,1}$	0.993	$H_0^{3,1}$	$H_A^{3,1}$	Case 3 is not better than case 1 w.r.t. computation time.
$\text{test}^{3,1}$ (without outliers)	2.2×10^{-16}	$H_A^{3,1}$	$H_0^{3,1}$	Case 3 is better than case 1 w.r.t. computation time.
$\text{test}^{2,1}$	0.994	$H_0^{2,1}$	$H_A^{3,1}$	Case 2 is not better than case 1 w.r.t. the computation time.
$\text{test}^{2,1}$ (without outliers)	0.998	$H_0^{3,1}$	$H_A^{3,1}$	Case 2 is not better than case 1 w.r.t. the computation time.
$\text{test}^{3,2}$	0.034	$H_A^{3,2}$	$H_0^{3,2}$	Case 3 is better than case 2 w.r.t. the computation time.
$\text{test}^{3,2}$ (without outliers)	2.1×10^{-7}	$H_A^{3,1}$	$H_0^{3,2}$	Case 3 is better than case 2 w.r.t. the computation time.

tive hypothesis $H_A^{3,1}$ is rejected and there is no difference on d_1 and d_2.

The p-values for all tests are shown in Table 11.2. Let us first consider the first test on computation time ($H_0^{3,1}$ and $H_A^{3,1}$). The p-value is 0.993, which means the null hypothesis $H_0^{3,1}$ (case 3 is not better than case 1 w.r.t. the computation time) is accepted. In the computational time of case 3, some outliers† exist and by removing these outliers and conduction the test again, the newly obtained p-value is 2.2×10^{-16}, and the null hypothesis is rejected. Hence, $H_A^{3,1}$ is accepted (meaning that case 3 is better than case 1 regarding the computation time, but there may exist extreme cases (the outliers) in case 3).

From the results in Table 11.2, it can be inferred that, in general, the performance w.r.t. the computation time of case 3 is the best among all three cases.

† In statistics, an *outlier* is an observation point that is distant from other observations [26].

12 CONCLUSIONS AND FUTURE WORKS ON ROBOT PLANNING

We proposed a decentralized control policy for deadlock prevention problem in robot planning. The robot planning problem contains a static map partitioned into regions. Some regions have limited capacities. Multiple robots move in the map following predefined trajectories to accomplish their tasks. Among all representations of this robot planning scheme, we choose a subclass of Petri Net (PN) to model robot trajectories and capacities of regions. The subclass is the so-called S^3PR (Systems of Simple Sequential Processes with Resources), and it is widely used in resource allocation and deadlock prevention studies [24]. An S^3PR consists of processes and resources. A process means a sequence of activities (events). Each activity requires a resource. The execution of an activity equals the firing of a transition in the S^3PR. The firing of the transition consumes two tokens from two places, one is the resource place and the other one is a process place. This constraint is relaxed to the case that one activity can require more than one resources in S^4PR (e.g., [16]). It means that the firing of the transition can consume tokens from more than one resource places. In robot planning, an activity means a robot enters into a region and the resources are the capacities of regions.

Many deadlock prevention policies have been developed, based on centralized monitor places. First, they find siphons, which lead the S^3PR into deadlock, and second, they use monitor places to control these siphons. We propose a different control policy by using inhibitor arcs to control siphons, instead of monitor places. The monitor places maintain the numbers of usable resources in siphons. When a robot wants to enter in a region, it must get the permission from the corresponding controller. Our approach waives the cost of building and maintaining of the controllers. It requires that when a robot wants to enter into a region, it reads sensors' states of some regions, where the regions are specified by the inhibitor arcs. In the case when robots' tasks change, monitor places have to be rebuilt, which means extra costs. In our approach, no such extra costs are introduced. Monitor places create new siphons with places in the original S^3PR. These new siphons must be controlled. In our case, *virtual siphons* are introduced by inhibitor arcs, and virtual siphons may lead the system into deadlock. However, virtual siphons can also be controlled using inhibitor arcs.

In some environments, real time control laws are not applicable, e.g., lack of efficient communication. In order to implement a collision-free movement for robots, we imposed initial time delays for trajectories of each mobile robots, assuming once robots start to move, no control can be applied, robots will not fail and no other perturbation exists. Here, multiple robots move in a shared map and the map is partitioned into regions. We assumed the capacities of all regions are one and define collision as that one robot wants to enter a region being occupied by another robot. Our goal was to ensure collision-free movement. Each robot has a set of trajectories and will follow one of them. All of these trajectories let the robot accomplish its task, meaning that they are equivalent in the aspect of completing its task. For example, a task can be that the robot moves a piece of produce from one region to another one. All possible paths between the two regions can be trajectories of the robot. We also assumed that time for crossing each region is known.

Two versions of the problem were considered. In the first, each robot can choose which trajectory to follow by itself, while in the second version, the trajectory followed by each robot is chosen by an external computing unit. In the first version, it must be ensured that the movement is collision-free regardless to which trajectories are chosen by robots. In the second, the computing unit can evaluate all possible combinations of trajectories to choose optimal ones and ensure that only chosen ones are collision-free. Once the initial time delays are obtained, the robot movements are performed and no intermediate motion pauses are allowed. We formulated the two versions using *Mixed Integer Linear Programming problems* (MILP) and provided numerical simulations and statistical analysis.

Open topics as future works can be:

- Maximal permissive deadlock prevention policy could be investigated. Current control policies are not maximal permissive. Some markings are eliminated from the state space to ensure liveness, but system performance may become lower.

- For verification of robots' trajectories, in future, it can be discussed that how to select better trajectories before verify them. Some MILP are complex for current solvers. Hence, it is important to choose better trajectories to reduce the computation time of solving these MILP.

BIBLIOGRAPHY

[1] What is a model? URL http://serc.carleton.edu/introgeo/models/WhatIsAModel.html.

[2] Srinivas Akella and Seth Hutchinson. Coordinating the motions of multiple robots with specified trajectories. In *Proceedings of IEEE International Conference on Robotics and Automation, 2002*, volume 1, pages 624–631. IEEE, 2002.

[3] Francesco Basile, Pasquale Chiacchio, and Gianmaria De Tommasi. Improving on-line fault diagnosis for discrete event systems using time. In *IEEE International Conference on Automation Science and Engineering*, pages 26–32, 2007.

[4] Francesco Basile, Pasquale Chiacchio, and Gianmaria De Tommasi. An efficient approach for online diagnosis of discrete event systems. *IEEE Transactions on Automatic Control*, 54(4):748–759, 2009.

[5] Francesco Basile, Maria Paola Cabasino, and Carla Seatzu. State estimation and fault diagnosis of labeled time Petri net systems with unobservable transitions. *IEEE Transactions on Automatic Control*, 60(4):997–1009, 2015.

[6] Calin Belta and Luc C.G.J.M. Habets. Controlling a class of nonlinear systems on rectangles. *IEEE Transactions on Automatic Control*, 51(11):1749–1759, 2006.

[7] Calin Belta, Volkan Isler, and George J Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874, 2005.

[8] Albert Benveniste, Eric Fabre, Stefan Haar, and Claude Jard. Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, 2003.

[9] Gérard Berthelot, Gérard Roucairol, and Rüdiger Valk. Reductions of nets and parallel programs. In *Net theory and Applications*, pages 277–290. Springer, 1980.

[10] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259 –273, 1991.

[11] Bernard Berthomieu and Miguel Menasche. An enumerative approach for analyzing time Petri nets. In *IFIP Congress Series*, volume 9, pages 41–46, 1983.

[12] Hanifa Boucheneb, Guillaume Gardey, and Olivier H. Roux. TCTL model checking of time Petri nets. *Journal of Logic and Computation*, 19(6):1509–1540, 2009.

[13] Maria Paola Cabasino, Alessandro Giua, Cristian Mahulea, Laura Recalde, Carla Seatzu, and Manuel Silva. State estimation of Petri nets by transformation. In *IEEE International Conference on Automation Science and Engineering*, pages 194–199. IEEE, 2007.

[14] Maria Paola Cabasino, Alessandro Giua, Andrea Paoli, and Carla Seatzu. Decentralized diagnosis of Petri nets. In *American Control Conference*, pages 3371–3377. IEEE, 2010.

[15] Maria Paola Cabasino, Alessandro Giua, and Carla Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9):1531 – 1539, 2010.

[16] Elia Cano, Carlos Rovetto, and José Manuel Colom. An algorithm to compute the minimal siphons in S^4PR nets. *Discrete Event Dynamic Systems*, 22(4):403–428, 2012.

[17] Christos Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2008.

[18] YuFeng Chen, ZhiWu Li, Mohamed Khalgui, and Olfa Mosbahi. Design of a maximally permissive liveness-enforcing Petri net supervisor for flexible manufacturing systems. *IEEE Transactions on Automation Science and Engineering*, 8(2):374–393, 2011.

[19] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George A. Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Boston, 2005.

[20] Roberto Cordone, Luca Ferrarini, and Luigi Piroddi. Enumeration algorithms for minimal siphons in Petri nets based on place constraints. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35(6):844–854, 2005.

[21] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[22] Rami Debouk, Stéphane Lafortune, and Demosthenis Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems*, 10(1):33–86, 2000.

[23] Mariagrazia Dotoli, Maria Pia Fanti, Agostino Marcello Mangini, and Walter Ukovich. On-line fault detection in discrete event systems by Petri nets and integer linear programming. *Automatica*, 45(11):2665–2672, 2009.

[24] Joaquin Ezpeleta, José Manuel Colom, and Javier Martinez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Transactions on Robotics and Automation*, 11 (2):173–184, 1995.

[25] Maria Pia Fanti, Agostino Marcello Mangini, and Walter Ukovich. Fault detection by labeled Petri nets and time constraints. In *2011 3rd International Workshop on Dependable Control of Discrete Systems (DCDS)*, pages 168–173, 2011.

[26] David Freedman, Robert Pisani, and Roger Purves. *Statistics*. WW Norton & Co, 2007.

[27] Sahika Genc and Stéphane Lafortune. Distributed diagnosis of place-bordered Petri nets. *IEEE Transactions on Automation Science and Engineering*, 4(2):206–219, 2007.

[28] A. Giua, D. Corona, and C. Seatzu. State estimation of λ-free labeled Petri nets with contact-free nondeterministic transitions*. *Discrete Event Dynamic Systems*, 15(1):85–108, 2005.

[29] Alessandro Giua and Carla Seatzu. Fault detection for discrete event systems using petri nets with unobservable transitions. In *44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05.*, pages 6323–6328. IEEE, 2005.

[30] Alessandro Giua, Carla Seatzu, and Francesco Basile. Petri net control using event observers and timing information. In *Proceedings of the 41st IEEE Conference on Decision and Control*, volume 1, pages 787–792, 2002.

[31] Igor Griva, Stephen G. Nash, and Ariela Sofer. *Linear and Nonlinear Optimization*. Society for Industrial Mathematics, 2008. 2nd ed.

[32] Luc C.G.J.M Habets, Pieter J Collins, and Jan H van Schuppen. Reachability and control synthesis for piecewise-affine hybrid systems on simplices. *IEEE Transactions on Automatic Control*, 51 (6):938–948, 2006.

[33] Rachid Hadjidj and Hanifa Boucheneb. Efficient Reachability Analysis for Time Petri Nets. *IEEE Transactions on Computers*, 60 (8):1085 –1099, August 2011.

[34] Yisheng Huang, MuDer Jeng, Xiaolan Xie, and Shengluen Chung. Deadlock prevention policy based on Petri nets and siphons. *International Journal of Production Research*, 39(2):283–305, 2001.

[35] George Jiroveanu and René K Boel. A distributed approach for fault detection and diagnosis based on time Petri nets. *Mathematics and Computers in Simulation*, 70(5):287–313, 2006.

[36] Jorge Júlvez and René K Boel. A continuous Petri net approach for model predictive control of traffic systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40 (4):686–697, 2010.

[37] Marius Kloetzer and Cristian Mahulea. A Petri net based approach for multi-robot path planning. *Discrete Event Dynamic Systems*, 24(4):417–445, 2014.

[38] Marius Kloetzer, Cristian Mahulea, and José Manuel Colom. Petri net approach for deadlock prevention in robot planning. In *IEEE 18th Conference on Emerging Technologies & Factory Automation*, pages 1–4. IEEE, 2013.

[39] Steven M. LaValle. *Planning Algorithms*. Cambridge, 2006. Available at `http://planning.cs.uiuc.edu`.

[40] ZhiWu Li and MengChu Zhou. Elementary siphons of Petri nets and their application to deadlock prevention in flexible manufacturing systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 34(1):38–51, 2004.

[41] ZhiWu Li and MengChu Zhou. On siphon computation for deadlock control in a class of Petri nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 38(3):667–679, 2008.

[42] ZhiWu Li and MengChu Zhou. On siphon computation for deadlock control in a class of petri nets. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 38(3):667–679, 2008.

[43] ZhiWu Li, NaiQi Wu, and MengChu Zhou. Deadlock control of automated manufacturing systems based on Petri nets - a literature review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 42(4):437–462, July 2012. ISSN 1094-6977.

[44] Cristian Mahulea, Carla Seatzu, Maria Paola Cabasino, and Manuel Silva. Fault diagnosis of discrete-event systems using continuous Petri nets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 42 (4):970 – 983, 2012.

[45] Andrew Makhorin. GNU linear programming kit, june 2012. http://www.gnu.org/software/glpk/, 2012.

[46] Philip M. Merlin. *A study of the recoverability of communication protocols*. PhD thesis, PhD thesis, University of California, Computer Science Dept., Irvine, 1974.

[47] Jordi Meseguer, Vicenç Puig, and Teresa Escobet. Fault diagnosis using a timed discrete-event approach based on interval observers: application to sewer networks. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40(5): 900–916, 2010.

[48] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[49] Jufeng Peng and Srinivas Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *The International Journal of Robotics Research*, 24(4):295–310, 2005.

[50] Carl Adam Petri. *Kommunikation Mit Automaten*. PhD thesis, Darmstadt University of Technology, Bonn, Germany, 1962.

[51] Marco Pocci. Matlab toolbox for the diagnosis of discrete PNs, 2009. URL http://www.diee.unica.it/giua/TESI/09_Marco.Pocci/.

[52] Laura Recalde, Manuel Silva, Joaquín Ezpeleta, and Enrique Teruel. Petri nets and manufacturing systems: An examples-driven tour. In *Lectures on Concurrency and Petri Nets*, pages 742–788. Springer, 2004.

[53] Yu Ru and Christoforos N Hadjicostis. Bounds on the number of markings consistent with label observations in Petri nets. *IEEE Transactions on Automation Science and Engineering*, 6(2):334–344, 2009.

[54] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[55] Carla Seatzu, Manuel Silva, and Jan H. van Schuppen, editors. *Control of Discrete-Event Systems*, volume 433 of *Lecture Notes in Control and Information Sciences*. Springer London, 2013. ISBN 978-1-4471-4275-1.

[56] Manuel Silva. Introducing Petri nets. In *Practice of Petri Nets in manufacturing*, pages 1–62. Springer, 1993.

[57] Manuel Silva and Robert Valette. Petri nets and flexible manufacturing. In *Advances in Petri nets 1989*, pages 374–417. Springer, 1990.

[58] Manuel Silva, Enrique Teruel, and José Manuel Colom. Linear algebraic and linear programming techniques for the analysis of place/transition net systems. In *Lectures on Petri Nets I: Basic Models*, pages 309–373. Springer, 1998.

[59] Zineb Simeu-Abazi, Maria di Mascolo, and Michal Knotek. Fault diagnosis for discrete event systems: Modelling and verification. *Reliability Engineering & System Safety*, 95(4):369–378, 2010.

[60] Robert H. Sloan and Ugo Buy. Reduction rules for time Petri nets. *Acta Informatica*, 33(7):687–706, 1996.

[61] Fernando Tricas, Fernando Garcia-Valles, José Manuel Colom, and Joaquin Ezpeleta. A Petri net structure-based deadlock prevention solution for sequential resource allocation systems. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 271–277. IEEE, 2005.

[62] WMP Van der Aalst and KM Van Hee. Business process redesign: a Petri-net-based approach. *Computers in industry*, 29(1):15–26, 1996.

[63] Jiacun Wang, Yi Deng, and Gang Xu. Reachability analysis of real-time systems using time Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 30(5):725–736, 2000.

[64] ShouGuang Wang, ChengYing Wang, MengChu Zhou, and ZhiWu Li. A method to compute strict minimal siphons in a class of Petri nets based on loop resource subsets. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 42(1):226–237, 2012.

[65] Xu Wang, Cristian Mahulea, Jorge Jùlvez, and Manuel Silva. On state estimation of timed choice-free Petri nets. In *Proceedings of the 18th IFAC World Congress*, 2011.

[66] Xu Wang, Cristian Mahulea, and Manuel Silva. Decentralized diagnosis based on fault diagnosis graph. In *2013 IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, September 2013.

[67] Xu Wang, Cristian Mahulea, and Manuel Silva. Model checking on fault diagnosis graph. In *12th International Workshop on Discrete Event Systems*, pages 434–439, May 2014.

[68] Xu Wang, Cristian Mahulea, and Manuel Silva. Diagnosis of Time Petri Nets Using Fault Diagnosis Graph. *IEEE Transactions on Automatic Control*, 60:2321–2335, 2015.

[69] Ke-Yi Xing, Bao-Sheng Hu, and Hao-Xun Chen. Deadlock avoidance policy for Petri-net modeling of flexible manufacturing systems with shared resources. *IEEE Transactions on Automatic Control*, 41(2):289–295, 1996.

[70] Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[71] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308 – 320, 2013.

[72] MengChu Zhou and Mu Der Jeng. Modeling, analysis, simulation, scheduling, and control of semiconductor manufacturing systems: A Petri net approach. *IEEE Transactions on Semiconductor Manufacturing*, 11(3):333 –357, aug 1998. ISSN 0894-6507.