Aix-Marseille University
Lab. des Sciences de l'Information et des Systèmes
ED 184 — Ph.D. School in Mathematics and Informatics

Test and diagnosis of discrete event systems using Petri nets

Marco POCCI

Thesis submitted for the PhilosophiæDoctor degrees

at the Aix-Marseille University in
Mathematics and Informatics
and
at the University of Cagliari in
Electronic and Computer Engineering

Defended on: September the 23rd, 2013

Thesis committee:

Mme Isabel Demongodin	Professor, Aix-Marseille University	Advisor
M. Norbert Giambiasi	Professor, Aix-Marseille University	Co-advisor
M. Alessandro Giua	Professor, University of Cagliari & Aix-Marseille University	Co-advisor
M. Hassane Alla	Professor, University of Grenoble	Reviewer
M. Jean-Jacques Lesage	Professor, École normale supérieure de Cachan	Reviewer
M. Francesco Basile	Assistant Professor, University of Salerno	Examiner
Mme Mariagrazia Dotoli	Tenured Assistant Professor, Polytechnic of Bari	Examiner
Mme Carla Seatzu	Associate Professor, University of Cagliari	Examiner

University of Cagliari
Dept. of Electrical and Electronic Engineering
DRIEI — Ph.D. School in Information Engineering

Aix-Marseille University
Lab. des Sciences de l'Information et des Systèmes
ED 184 — Ph.D. School in Mathematics and Informatics

Test and diagnosis of discrete event systems using Petri nets

Marco POCCI

Thesis submitted for the PhilosophiæDoctor degrees

at the Aix-Marseille University in
Mathematics and Informatics
and
at the University of Cagliari in
Electronic and Computer Engineering

Defended on: September the 23rd, 2013

Thesis committee:

Mme Isabel Demongodin	Professor, Aix-Marseille University	Advisor
M. Norbert Giambiasi	Professor, Aix-Marseille University	Co-advisor
M. Alessandro Giua	Professor, University of Cagliari & Aix-Marseille University	Co-advisor
M. Hassane Alla	Professor, University of Grenoble	Reviewer
M. Jean-Jacques Lesage	Professor, École normale supérieure de Cachan	Reviewer
M. Francesco Basile	Assistant Professor, University of Salerno	Examiner
Mme Mariagrazia Dotoli	Tenured Assistant Professor, Polytechnic of Bari	Examiner
Mme Carla Seatzu	Associate Professor, University of Cagliari	Examiner

University of Cagliari
Dept. of Electrical and Electronic Engineering
DRIEI — Ph.D. School in Information Engineering

"*You only live once,*
but if you do it right, once is enough.”

Mae West

Test and diagnosis of discrete event systems using Petri nets

Abstract

State-identification experiments are designed to identify the final state of a *discrete event system* (DES) when its initial state is unknown. A classical solution to this problem, assuming the DES has no observable outputs, consists in determining a *synchronizing sequence*, i.e. a sequence of input events that drives the system to a known state. This problem was addressed and essentially solved in the 60' using finite state machines or automata. The main objective of this thesis is to use Petri nets for solving the state-identification problem more efficiently and for a wider class of systems, with possibly infinite states.

We start showing that the classical method based on automata for constructing a synchronizing sequence can be easily applied with minor changes to *synchronized Petri nets*, a class of non-autonomous nets where each transition is associated with an input event. The proposed approach is fairly general and it works for arbitrary bounded nets with a complexity that is polynomial with the size of the state space. However, as most of the problem solving methods for DES, it incurs in the well-known state-space explosion problem.

Looking for more efficient solutions, we begin by considering a subclass of Petri nets called *state machines*. We first consider strongly connected state machines and propose a framework for synchronizing sequence construction that exploits structural criteria, and thus does not require an exhaustive enumeration of the state space of the net. These results are further extended to larger classes of nets, namely non strongly connected state machines and nets containing state machine subnets. Finally we consider the class of unbounded nets that describe infinite state systems: even in this case we are able to compute a set of sequences to synchronize the marking of bounded places. A Matlab toolbox implementing all approaches previously described has been designed and applied to a series of benchmarks.

An additional problem addressed in this thesis and partially related to state-identification, is the *failure diagnosis* of DES, i.e., the process of identifying the occurrence of a fault based on observable symptoms. Our work is centered on diagnosis of DES using model-

based methods where a common assumption requires that a fault model be available. The system is represented by a *labeled Petri net*, in which transitions can be either observable or silent; a fault is modeled by a silent transition. We consider an efficient approach, based on *basis markings*, to compute a diagnosis state which produces different degrees of alarm, such as "normal" or "faulty" or "uncertain". A Matlab toolbox for solving the diagnosis problem and the related problem of diagnosability has been designed, and tested on a parametric example.

Test et diagnostic des systèmes à événements discrets par les réseaux de Petri

Résumé

L'exécution d'un test d'identification d'état d'un *système à événement discret* (SED) a pour but d'en identifier l'état final, lorsque son état initial est inconnu. Une solution classique à ce problème, en supposant que le SED n'ait pas de sorties observables, consiste à déterminer une *séquence de synchronisation*, c.à-d., une séquence d'événements d'entrée qui conduit le système sur un état connu. Ce problème a été abordé et complètement résolu dans les années 60' à l'aide de *machines à états finis* ou d'*automates*. L'objectif principal de cette thèse est d'utiliser les réseaux de Petri dans le but d'obtenir une résolution plus optimal du problème d'identification d'état et pour une plus large classe de systèmes, dont le nombre d'états pourrait être infini.

Dans une première approche, nous montrons que la méthode classique des automates pour la construction des séquences de synchronisation peut être aisément étendue — par des modifications mineures — aux *réseaux de Petri synchronisés*. Pour cette classe de réseaux non-autonomes, toute transition est associée à un événement d'entrée.

L'approche proposée est assez générale, dans la mesure où elle s'applique à des réseaux bornés arbitraires. Cependant, comme la plupart des méthodes s'appliquant aux SEDs, elle engendre le problème classique d'explosion combinatoire du nombre d'états. Dans le but d'obtenir des meilleures solutions, nous considérons une classe spéciale de réseaux de Petri, connue sous le nom de *graphes d'état* (GdE). Pour ces réseaux, nous considérons d'abord les GdE fortement connexes et nous proposons deux approches pour la construction de séquences de synchronisation. Ces approches exploitent les propriétés structurelles du réseau et ne nécessitent pas ainsi une énumération exhaustive de l'espace d'état. Ces résultats s'étendent par la suite aux GdE non fortement connexes et puis à tout RdP synchronisés composés de GdE.Enfin, nous considérons la classe des réseaux de Petri non bornés et proposons de construire des séquences qui synchronisent le marquage des places non bornées. Une boîte à outils fournit toutes les approches précédemment décrites et est appliquée à des différents bancs d'essai.

Dans cette thèse, nous traitons aussi le diagnostic de fautes de SED, c.à-d., le processus d'identification des causes d'un échec basé sur l'observation de symptômes. Nous nous intéressons au diagnostic de SED à l'aide de méthodes analytiques basées sur modèle, sous l'hypothèse que le modèle de faute soit disponible. Le système est alors modélisé par un *réseau de Petri étiqueté*, pour lequel les transitions sont *observables* si elles sont associées à un événement de sortie, ou *silencieuses*; tout comportement fautif est modélisé par une transition silencieuse. Nous considérons une approche efficace basée sur la notion de *marquages de base* qui calcule un état de diagnostic, qui caractérise différents degrés d'alarme, tels que "normal", "fautif" ou "incertain". Une boîte à outils pour la résolution du problème de diagnostic et de celui de la diagnosticabilité est fournie et testée pour un exemple paramétrique.

Test e diagnosi di sistemi ad eventi discreti mediante le reti di Petri

Riassunto

Il *problema dell'identificazione di stato* è concepito per la determinazione dello stato finale di un *sistema ad eventi discreti* (SED), nell'ipotesi che il suo stato iniziale sia inizialmente sconosciuto. Una soluzione classica a questo problema, assumendo il sistema privo di uscite osservabili, consiste nella costruzione di una *sequenza di sincronizzazione*, cioè una sequenza di ingressi che conduce il sistema ad uno stato noto. Tale approccio è stato risolto negli anni 60' tramite *macchine a stati finiti* ed *automi*. L'obiettivo principale di questa tesi è quello di proporre le *reti di Petri* in questo contesto con lo scopo di proporre soluzioni più efficienti e valide per una più vasta classe di sistemi, con eventualmente un numero infinito di stati.

In un primo tempo, viene dimostrato come l'approccio classico proposto per gli automi possa essere facilmente adattato nel contesto delle *reti di Petri sincronizzate* limitate. Per questa classe di reti non autonome, ciascuna delle transizioni è associata ad un evento di ingresso. L'approccio proposto è piuttosto generale, in quanto applicabile a tutte le reti limitate, ed ha una complessità computazionale polinomiale con la dimensione dello spazio di stato. Tuttavia, come la più parte degli approcci su SED, si imbatte sul problema dell'esplosione combinatoria dello spazio di stato.

In seguito, con lo scopo di fornire tecniche più efficienti, ci si interessa ad una classe di reti di Petri quale le *macchine di stato*. Per questa classe, consideriamo inizialmente strutture fortemente connesse e proponiamo due approcci distinti per la costruzione di sequenze di sincronizzazione. Entrambi gli approcci determinano una sequenza di sincronizzazione sulla base della sola struttura della rete e ciò a prescindere del numero di gettoni contenuto dalla rete stessa, quindi evitando una numerazione esaustiva dello spazio di stato.

Successivamente, i risultati sono estesi anche a reti non fortemente connesse ed a reti contenenti macchine di stato come sottoreti. Infine, il problema della costruzione di sequenze di sincronizzazione è affrontato anche per reti non limitate, ossia reti di Petri il cui spazio di stato è infinito. Anche in questo caso, siamo capaci di determinare una sequenza che sincronizzi il contenuto dei posti limitati. Un Toolbox Matlab implementa tutti gli approcci precedentemente descritti ed è applicato ad una serie di esempi applicativi.

Un ulteriore problema presentato nella tesi, nonché parzialmente connesso a quello della identificazione di stato, è quello della *diagnosi di guasto* di SED, ossia il processo di identificazione delle cause di guasto, sulla base sull'osservazione dei "sintomi". Il nostro interesse è centrato sulla diagnosi di SED ed in particolare considereremo approcci basati sul modello, per i quali viene comunemente assunto che il modello di guasto sia noto. Il sistema è rappresentato da una *rete di Petri etichettata*, in cui una transizione può essere *osservabile*, se associata ad un evento di ingresso, o *non osservabile*, altrimenti; un guasto è associato ad una transizione non osservabile. Consideriamo un approccio efficiente, basato sulla nozione di *marcatura di base*, per il calcolo di uno *stato di diagnosi*, che caratterizza differenti livelli di allarme, quali "normale", "anomalo" o "incerto". Viene fornito un toolbox Matlab per la risoluzione del problema di diagnosi e del connesso problema della diagnosticabilità ed infine testato per un esempio parametrico.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Summary

In this chapter first we present an overview on the problems of the synchronizing sequences and diagnosis. We give a motivation for these studies discussing the contribution of this thesis. Secondly we describe the organization of the thesis and the topics of each chapter.

Introduction

With a growing system complexity, the issues of test and diagnosis to ensure troubleshooting failures of these systems are becoming more and more important. A *failure* is defined to be any deviation of a system from its specified behavior. Failure diagnosis is the process of identifying the causes of a failure based on observable symptoms, i.e., determining which fault led to this erroneous behavior.

Failures are inevitable in today's complex industrial environment and they could arise from several sources. Hence the need for effective means of detecting them is quite apparent if their consequences and impacts are considered not just on the systems involved but on the society as a whole. Moreover, note that efficient methods of failure diagnosis can not only help avoiding the undesirable effects of failures, but can also enhance the operational goals of industries. Improved quality of performance, product integrity and reliability, and reduced cost of equipment maintenance and service are some major benefits that accurate diagnosis schemes can provide, especially for service and product oriented industries such as home and building environment control, office automation, automobile manufacturing, and semiconductor manufacturing. Thus, one can see that accurate and timely methods of failure diagnosis can enhance the safety, reliability, availability, quality, and economy of industrial processes. The need of automated mechanisms for the timely and accurate diagnosis of failures is well understood and appreciated both in industry and in academia. A great deal of research effort has been and is being spent in the design and development of automated diagnostic systems, and a variety of schemes, differing both in their theoretical framework and in their design and implementation philosophy, have been proposed.

Failure diagnosis can be classified as i) *fault-tree based methods*; ii) *analytical model-based methods*; iii) *knowledge-based methods*; iv) *simulation-based methods*. These methods apply to *continuous systems* and *discrete event systems* (DES). Our work is centered on diagnosis of DES by the aid of *Petri nets* (PNs).

Model-based diagnosis of DES is mainly carried out by the way of an observer, i.e., a model that estimates the system state on the basis of observed outputs. Observer and real system must clearly be in the same initial state, otherwise any further analysis would be meaningless or misleading. For this reason, we focus first on a class of problems known as state-identification problems. State-identification experiments are designed to identify the final state of a DES when its initial state is unknown. Note that if the model is synchronizable, then it will allow simple error recovery since, if a faulty behavior is detected — e.g., unexpected reached state or wrong observed output —, it can be initialized into a known state.

It has been first investigated by Natarajan [94] to solve the design of automated parts orienteers, the dual of the so-called navigation problem (aka the motion planning problem). The reader can easily see that for example every device part, when arriving at manufacturing sites, needs to be sorted and oriented before assembly. In his work, Natarajan have considered a merely mathematical model that ignores the dynamics of the situation. The concerned robots do not suffer from finite precision in their measurement location but the planning is done in presence of uncertainties in position measurement. In particular a robot might be ask to assemble a composite object by matching different part, the robot is

capable of precise moves but the grasp is done visionless. The task becomes then to orient that part in order to correctly get the object orientation that satisfies the requirement, such a process of reaching a desired state of the system under an initial uncertainty is called synchronization.

In the domain of discrete event systems synchronization coincides in driving a system, seen as a black box, to a known state when its current state it is not known. A classical solution to this problem, assuming the DES has no observable outputs, consists in determining a *synchronizing sequence* (SS). This problem was addressed and essentially solved completely in the 60' using finite state machines (or automata) to model DES. This problem has many important applications and is of general interest. It is of relevant importance for robotics and robotic manipulation [94, 95, 3], when dealing with part handling and orienting problems in industrial automation such as part feeding, loading, assembly and packing. Synchronization protocols have been developed to address global resource sharing in hierarchical real-time scheduling frameworks [127, 6]. An interesting automotive application can be found in [96]. Synchronization experiments have been done also in biocomputing, where Benenson *et al.* [8, 7] have used DNA molecules as both software and hardware for finite automata of nano-scaling size. They have produced a solution of 3×10^{12} identical automata working in parallel. In order to synchronously bring each automaton to its "ready-to-restart" state, they have spiced it with a DNA molecule whose nucleotide sequence encodes a reset word. Jürgensen [68] has surveyed synchronization issues from the point of view of coding theory in real life communication systems. He has presented the concept of synchronization in information channels, both in the absence and in the presence of noise. Synchronization is an important issue in network time protocol [47, 89], where sharing of time information guaranties the correct internet system functioning. Most of real systems, natural or man-built, have no integrated reset or cannot be equipped with. That is the case of digital circuits, where a reset circuit not only involves human intervention but increases the cost of the device itself reducing its effectiveness. In this field Cho *et al.* [25] have shown how to generate test cases for synchronous circuits with no reset. When classic procedures fail due to large circuit size or because a synchronizing sequence does not exist, Lu *et al.* [83] propose a technique based on partial reset, i.e., special inputs that reset a subset of the flip-flops in the circuit leaving the other flip-flops at their current values. Hierons [62] has presented a method to produce a test sequence with the minimum number of resets.

Nowadays the synchronizing theory is a field of very intensive research, motivated also by the famous Černý conjecture [125]. In 1964 Ján Černý has conjectured that $(n-1)^2$ is the upper bound for the length of the shortest SS for any $n-$state machine. The conjecture is still open except for some special cases [40],[3],[121]. Synchronization allows simple error recovery since, if an error is detected, a SS can be used to initialize the machine into a known state. That is why synchronization plays a key rôle in scientific contexts, without which all system behavior observations may become meaningless. Thus the problem of determining which conditions admit a synchronization is an interesting challenge. This is the case of the *road coloring problem*, where one is asked whether there exists a coloring, i.e. an edge labeling, such that the resulting automaton can be synchronized. It was first stated by Adler in [1]. It has been investigated in various special cases and finally a positive solution has been presented by Trahtman in [123], for which complexity analysis

are provided [109].

In a first approach, we show that the classical method based on automata for synchronizing sequence construction can be easily applied with minor changes to synchronized PNs. For this class of non-autonomous nets, any transition is associated with an input event. The proposed approach is fairly general, as it works for arbitrary bounded nets. However, as most of problem solving methods on DES, it incurs the well known state space explosion problem. Looking for more efficient solutions, we begin by considering the class of PNs called *state machine* PNs. For this class, we first consider strongly connected nets and propose two novel approaches for synchronizing sequence construction. These approaches exploit net structural criteria, thus without an exhaustive enumeration of its state space. Next, these results are extended to non strongly connected nets and then to nets containing state machine subnets. Finally, we consider the larger class of unbounded nets. First, we propose a finite state space representation of these PNs. Second, results given for bounded nets are further extended to these nets.

In addition, we present a diagnosis approach for DES modeled by *labeled PNs*. The diagnosis of discrete event systems (DES) is a research area that has received a lot of attention in the last years. Faults may correspond to any discrete event. As an example, in a telecommunication system, a fault may correspond to a message that is lost or not sent to the appropriate receiver. Similarly, in a transportation system, a fault may be a traffic light that does not switch from red to green according to the given schedule. In a manufacturing system [128, 5, 84, 48], it may be the failure of a certain operation, e.g., a wrong assembly, or a part put in a wrong buffer, and so on.

In the diagnosis framework two different problems can be solved: the problem of diagnosis and the problem of diagnosability.

Solving a problem of diagnosis means that a diagnosis state, such as "normal" or "faulty" or "uncertain" is associated with each observation. Solving a problem of diagnosability is equivalent to determine if the system is diagnosable, i.e., to determine if, once a fault has occurred, the system can detect its occurrence in a finite number of steps. The second investigated topic is focused on the problem of diagnosis; see [16] and [14] for an extension of the methodology here proposed to the diagnosability problem. However, it is well known that diagnosability is an essential property that must hold if a diagnosis approach is to be applied in real life applications. Thus, the manufacturing example considered in this thesis is diagnosable. This property has been tested using the MATLAB tool in [99]. Note that if a system contains a non diagnosable fault there exist sequences of unbounded length that lead the system through diagnosis states that are uncertain. This means that when the fault occurs the diagnoser may not be able to detect its firing.

As discussed later the first results on diagnosis of DES have been presented within the framework of automata. More recently, the diagnosis problem has also been addressed using PNs. In fact, the use of PNs offers significant advantages because of their twofold representation: graphical and mathematical. Moreover, the intrinsically distributed nature of PNs where the notion of state (i.e., marking) and action (i.e., transition) is local reduces the computational complexity involved in solving a diagnosis problem.

This problem is investigated by the way of arbitrary labeled PNs where there is an as-

sociation between sensors and observable events, while no sensor is available for certain activities — such as faults or other unobservable but regular transitions — due to budget constraints or technology limitations. It is assumed that several different transitions might share the same sensor in order to reduce cost or power consumption. If two transitions are simultaneously enabled and one of them fires it is impossible to distinguish which one has fired, thus they are called *undistinguishable*. The diagnosis approach here presented is based on the definition of four diagnosis states modeling different degrees of alarm and it applies to all systems whose unobservable subnet is acyclic. Two are the main advantages of this procedure. First, it is not necessary an exhaustive enumeration of the states in which the system may be: this is due to the introduction of basis markings. Secondly, in the case of bounded net systems the most burdensome part of the procedure, namely building a finite graph called *basis reachability graph* (BRG), can be moved off-line.

Note that the approach here presented, as most of the approaches dealing with diagnosis of discrete event systems [33, 113, 112, 136], assumes that the faulty behavior is completely known, thus a fault model is available. Such an assumption is applicable to interesting classes of problems: this is the case of many manufacturing systems where the set of possible faults is often predictable and finite in number [48, 5, 84, 128]. Moreover, the proposed diagnosis approach allows one to deal with both permanent and intermittent faults. However, in the case of intermittent faults, once a fault is detected, even if a recovery event occurs, the diagnosis state associated to the fault is not reset to a non faulty state. The procedure can be easily extended to overcome this limitation. In particular, if the recovery event is observable a simple reset rule on the diagnosis state should be introduced. On the contrary, if the recovery event is not observable a detection procedure on such an event, based on the same features of the fault detection procedure here presented, should be applied.

The thesis is structured as follows. In Chapter 2 the literature of fault diagnosis of systems and basic testing problems relevant with the presented research is discussed. Formalisms of the discrete event system models used are also provided. Chapter 3 provides results on the computation of synchronizing sequences for systems modeled by PNs. The synchronization problem is here first investigated on systems represented by the class of bounded synchronized PNs. Three main approaches are given, for which computational complexity analysis are provided and comparisons are carried out by the aid of a Matlab toolbox. Finally, in chapter 4, the reader deals with the more general class of unbounded synchronized PNs. First, a modified coverability graph construction is provided to yield a faithful representation of the PN behavior. Then the results given for the bounded case are further extended to this setting. Chapter 5 is dedicated to diagnosis of labeled PNs, i.e., nets where two or more transitions may share the same label. Here, transitions may be undistinguishable, namely transitions that produce an output event that is observable, but that is common to other transitions. The provided online diagnosis approach is based on the notion of *basis marking and justification*, that allow to characterize the set of markings that are consistent with the actual observation. It is shown that the most burdensome part of the procedure can be moved off-line defining a particular graph, called *basis reachability graph*. Chapter 6 concludes the thesis with a discussion of the contributions and listing possible further directions of research. Appendix A reviews the Tarjan's algorithm for graph decomposition in its maximally strongly connected components.

Chapter 2

Formalisms, test and diagnosis of DES

Summary

Fault diagnosis of systems and basic testing problems have received considerable attention in the last decades. In this Chapter the state-of-the art of the two topics are presented within the framework of discrete event systems. The formalisms of the classes of discrete event system models used in the rest of the thesis are provided.

2.1 Formalisms of discrete event systems

In this section a short overview on discrete event systems is given. In particular two discrete event modeling formalisms used in the rest of the thesis are provided: automata and Petri nets.

System and Control Theory have focused from the very first beginning on systems generally described through ordinary differential or partial differential equations describing the corresponding physical phenomena. The state-space can be defined by means of continuous variables, which can get any real/complex value, and upon a certain input function, depicts a state-function (or a family of state-functions for more than one state variable), called *state trajectory* or equivalently *sample path*.

The advent of computers sometimes inclines one to describe their evolution by using discrete time dynamic equations, which do not change the intrinsic continuous nature of this evolution. So models known as *Discrete Event Systems* (DES) or more properly *Discrete Event Dynamic Systems* have followed, in contrast to *Continuous-Variable Dynamic Systems* (CVDS).

A DES is a discrete-state, event-driven system whose state evolution depends entirely on the occurrence of asynchronous discrete events over time [19].

The state space is here a discrete set $\chi = \{x_1, x_2, \ldots, x_n\}$ and, since DES state may change only upon asynchronous instantaneous discrete events, its sample path can be represented by a sequence of states or a sequence of states together with the time instants at which the transition take place. The two cases are respectively referred to the classes of un-timed and timed models. Those trajectories are then typically piecewise constant functions of time. Note that conventional differential equations are not a suitable description for such a "discontinuous" behavior.

Among the major DES models we may cite *automata*, *Markov chains*, *Petri nets*, *queuing models*, *finite state machines* and *Discrete Event System Specification*.

This work deals with the Petri net and automaton models and focuses on the first one. The two formalisms have both a state transition structure defining their dynamics, i.e. defining the state-space of the system upon any possible event.

Automata and Petri nets considered are both dynamic models and they belong to the class of Time-invariant models, i.e., models whose functions describing the dynamic of the model are time-invariant.

Automata and Petri nets are powerful formalisms for modeling DES, also because operations of composition are possible to construct a discrete event model of the system from a set of discrete event models representing the system components. Structural properties of such a transition structure are used to address analysis and synthesis issues.

When a DES describes the behavior of a system evolving in an dependent way, i.e. whose evolution is conditioned by input events and/or time, the model is denoted as non-autonomous (e.g. synchronized Petri nets and input/output automata). On the contrary the model is called autonomous (e.g. labeled Petri nets and any DES receiving no

inputs).

Petri nets were introduced in 60s by Carl Adam Petri — in his Ph.D. thesis [98] dissertation — to define a simple and powerful theoretical framework for studying concurrency problem.

Several reasons motivates the wide use of PNs.

- PNs are a mathematical and graphical formalism to model discrete event systems.
- PNs give a compact representation of the state space. In fact they do not require to explicitly represent all possible reachable states, but only the evolution rules.
- PNs can represent a discrete event system with an infinite number of states with a graph with a finite number of nodes.
- PNs can represent the concept of concurrency.
- PNs give a modular representation. In fact if a system is composed by more than one subsystems and these subsystems interact each other, then it is possible to represent each subsystem as a PN and then, using specific constructs, combine the single units to obtain the entire model.

A large set of extensions have been progressively defined with increasing complexity and capabilities, allowing system designers to handle the functional and the non-functional temporal and stochastic capabilities. Here we provide a short list of existing extensions.

High level Petri nets, do not modify the semantic of the net, attach new information to nodes arcs and tokens, to obtain new dense behavioral semantics using these parameters. These nets allows direct analysis and do not need to unfold the net. In particular we may cite:

- *colored Petri nets* [66], where tokens are distinguishable and typed;
- *algebraic nets* [106], in which elements of user defined data types (called algebraic abstract data types) replace black tokens;
- *labeled Petri nets* [49], which are used to model autonomous systems while observing their evolution. In this setting, it is common to assume that each transition is assigned to a label and its occurrence generates an observable output given by this label;
- *synchronized Petri nets* [91], which belong to the category of non-autonomous PNs. They describes systems whose evolution is dictated by external events associated with each transition;
- *well-formed Petri nets* [23, 24], where the functions of colors are restricted to composition of few elementary function (identity, successor and diffusion) to make it easier to analyze the net;

- *object-oriented Petri nets* [72], which support a thorough integration of object-oriented concepts into Petri nets.

When analyzing the behavior of systems for which time appears, we may use *Time Petri nets* [87], that extend the Petri nets by associating with each transition a firing duration.

Petri nets have been also extended to stochastic features, yielding *stochastic Petri nets*, whose dynamic behaviour could be represented by means of continuous-time homogeneous Markov chains. Petri nets with exponential and immediate distribution, called *generalized stochastic Petri nets*, are considered the as the standard model [85].

As the two used formalisms are concerned with quantitative analysis of real systems, it is needless to say that the they both provide inputs and outputs variables describing system responses to stimulus.

2.1.1 Input/output Automata

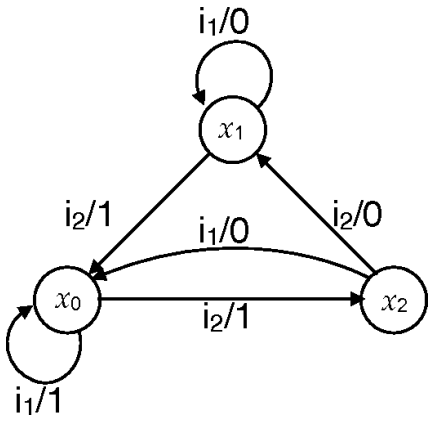


Figure 2.1: An example of automaton.

An *automaton* Λ is a quintuple denoted as $\Lambda = (\chi, I, O, \delta, \lambda)$ where:

- χ is a finite and nonempty set of states;
- I is a finite and nonempty set of input events;
- O is a finite and nonempty set of output events;
- $\delta : \chi \times I \to \chi$ is the state transition function;
- $\lambda : \chi \times I \to O$ is the output function.

When the automaton is in the current state $x \in \chi$ and receives an event $i \in I$, it reaches the next state specified by $\delta(x, i)$, producing an output $o \in O$ given by $\lambda(x, i)$.

Definition 2.1. ***(completely specified automaton)*** *An automaton* $\Lambda = (\chi, I, O, \delta, \lambda)$ *is called completely specified if its state transition function* δ *and its output function* λ *are* total functions*, i.e., functions defined on each element* (x, i) *of their domain.* ■

In other words, if an automaton is completely specified, for any state and upon any input event a state transition occurs producing the corresponding output event.

A compact way to represent an automaton is its state table, where each element gives the next state and the output, for a combination of a present state and input event.

Another simple way to represent any automaton is a graph (see for example Figure 2.1), where states, input and output events are respectively depicted as nodes and arcs. For instance, if it holds that $x' = \delta(x, i)$ and $o = \lambda(x, i)$, then there exists an oriented arc from node x to node x' labeled with the input event i and with the output event o.

Example 2.2. *In Figure 2.1 is shown the graph of an automaton* Λ*, where* $\chi = \{x_0, x_1, x_2\}$*,* $I = \{i_1, i_2\}$*,* $O = \{0, 1\}$*. The transition function* δ *and the output function* λ *are given by the following table:*

δ, λ	i_1	i_2
x_0	x_0, 1	x_2, 1
x_1	x_1, 0	x_0, 1
x_2	x_1, 0	x_0, 0

Let assume that the current state is x_1*. Upon input event* i_2*, the automaton moves to state* x_0 *and outputs* 1*.* ■

Note that I^* (resp. O^*) is the set of all input (resp. output) sequences constructed over the alphabet I (resp. O) and $*$ is the kleen star. For instance $\{i_1, i_2\}^* = \{\varepsilon, i_1, i_2, i_1i_2, i_2i_1, i_1i_2i_1, i_1i_1i_2 \ldots\}$.

The number of states, input and output events are respectively denoted by $n_x = |\chi|$, $n_i = |I|$ and $n_o = |O|$. One can extend the transition function δ and the output function λ from input events to sequences of input events as follows:

a) if ε denotes the empty input sequence, $\delta(x, \varepsilon) = x$ and $\lambda(x, \varepsilon) = \varepsilon$ for all $x \in \chi$;

b) for all $i \in I$ and for all $w \in I^*$ it holds that $\delta(x, wi) = \delta(\delta(x, w), i)$ and $\lambda(x, wi) = \lambda(x, w)\lambda(\delta(x, w), i)$.

Suppose that the automaton in Figure 2.1 is in state x_2. Input sequence $i_2i_1i_2$ drives it through states x_1, x_1 and x_0 and outputs 001.

Functions δ and λ can also be extended to a set of states as follows:

c) for a set of states $\chi' \subseteq \chi$, an input event $i \in I$ yields the set of states $\chi'' = \delta(\chi', i) = \bigcup_{x \in \chi'} \delta(x, i)$ and produces the set of outputs $\lambda(\chi', i) = \bigcup_{x \in \chi'} \lambda(x, i)$.

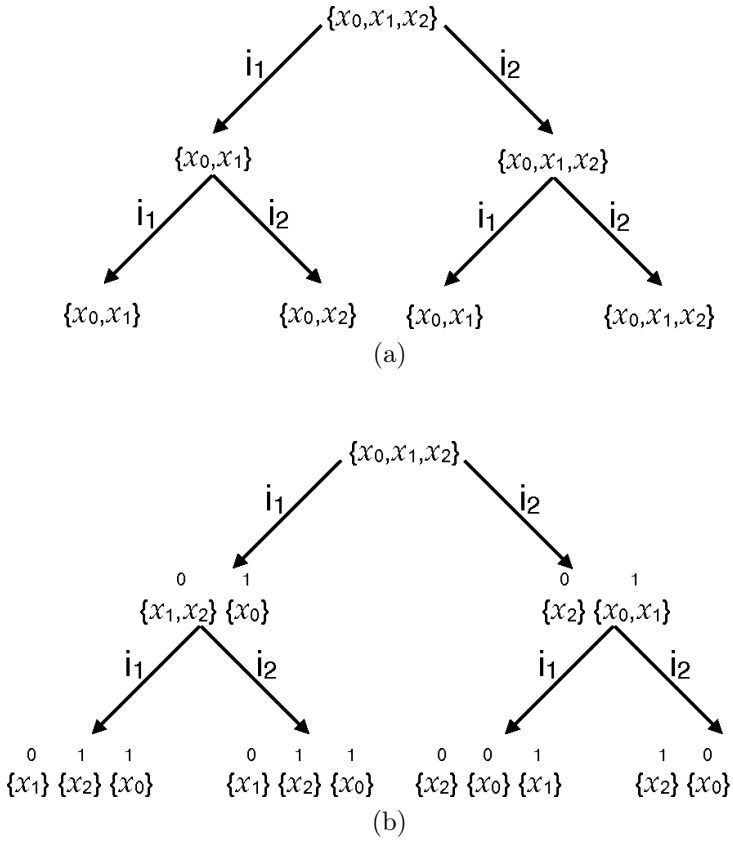


Figure 2.2: The successor trees of the automaton in Figure 2.1 with reference to the current state uncertainty (a) and the initial state uncertainty (b).

The information about the current state of Λ for a given input sequence is given by the *current state uncertainty of* w. After applying an input sequence w and with the hypothesis that the initial state is unknown the current state uncertainty of w is defined by the set $\phi(w) = \delta(\chi, w)$. Otherwise stated, upon an input sequence w the automata may be in one of the states of $\phi(w)$.

Analogously, any input sequence w is associated with a partition of the set of states $\tau(w)$, called *initial state uncertainty*. Two states x_1 and x_2 belong to the same subset of $\tau(w)$ if $\lambda(x_1, w) = \lambda(x_2, w)$.

In other words, if two states are committed to the same subset for a given input sequence, they are not distinguishable on the base of the produced output for this input sequence.

Example 2.3. *Consider again the automaton in Figure 2.1 and the input event* i_1. *The current state may be either* x_0 *or* x_1, *then the current state uncertainty is* $\phi(i_1) = \delta(\chi, i_1) = \{x_0, x_1\}$.

The output produced by Λ *in response to the input* i_1 *tells us that if we observe an output equal to : a)* 0 *the automaton was initially in state* x_1 *or* x_2*; b)* 1, *the automaton was initially in state* x_0. *So the initial state uncertainty is* $\tau(i_1) = \{\{x_1, x_2\}, \{x_0\}\}$. ■

The successor tree of an automaton is a tree-structure that shows the behavior of the

automaton from any possible initial state, i.e. the space-set coinciding with the root, under all possible sequence of input events.

Starting from the root, any path corresponds to an input event sequence and every node is labelled with the corresponding initial (and/or current) state uncertainty.

Example 2.4. *Consider the automaton in Figure 2.1. Figure 2.2 shows the successor trees related to the initial and current state uncertainty for input sequences of length at most* 2*. For both trees, the root is labeled with the whole set of state* $\chi = \{x_0, x_1, x_2\}$*. For instance, consider event* i_1 *as input.*

Figures 2.2a provides the information about the current state uncertainty, which is given by the first left children of the represented tree. Since $\delta(\{x_1, x_2\}, i_1) = x_1$ *and* $\delta(x_0, i_1) = x_0$*, it holds that* $\phi(i_1) = \{x_0, x_1\}$*.*

Analogously, the first left children of the tree depicted in Figure 2.2b provides the initial state uncertainty. Here we associate a different block with reference to the output produced. Since $\lambda(\{x_1, x_2\}, i_1) = 0$ *and* $\lambda(x_0, i_1) = 1$*, it holds that* $\tau(i_1) = \{\{x_1, x_2\}, \{x_0\}\}$*.* ■

Finally some structural properties, that will be used in the following, here are given. An automaton with inputs is said *strongly connected* if there exists a directed path from any node of its graph to any other one. It contains at least one ergodic component since a strongly connected automaton consists of a single ergodic component.

For a non strongly connected automaton, the following classification holds.

Definition 2.5. ***(Maximal strongly connected component)*** *Consider a non strongly connected automaton* $\Lambda = (\chi, I, O, \delta, \lambda)$*. Its nodes can be partitioned into its maximal strongly connected components. A component, i.e., a set of nodes, is called* ergodic*, if its set of output arcs is included in its set of input arcs,* transient*, otherwise.* ■

The algorithm for the decomposition of a graph in its strongly connected components is given in Appendix A.

Maximal strongly connected components can be further classified as follows.

Definition 2.6. ***(Condensed graph)*** *Given an automaton* $\Lambda = (\chi, I, O, \delta, \lambda)$*, one defines its corresponding* condensed graph $\mathcal{C}(\Lambda)$ *as the graph whose nodes represent its maximally strongly connected components (ER or TR) and whose edges represent the arcs connecting these components.* ■

Two states $x_i, x_j \in \chi$ are called *equivalent* if and only if for every input sequence the machine will produce the same output sequence regardless of whether x_i or x_j is the initial state, i.e. for any $w \in I^*$ $\lambda(x_i, w) = \lambda(x_j, w)$. Otherwise x_i and x_j are said *inequivalent* and w is a separating sequence. Two automata Λ and Λ' are equivalentif and only if for every state in Λ there is a corresponding equivalent state in Λ', and vice-versa.

An automaton is *reduced* or *minimized* if and only if no two states are equivalent.

For instance consider the automaton Λ in Figure 2.1. Since it is strongly connected, its condensed graph corresponds to a single node. The automaton is also completely specified and reduced.

2.1.2 Petri nets

In this section, it is recalled the PN formalism used in the thesis. First the basic notions of Petri nets such of marking and firing and the essential characteristics of a Petri net are presented in this section. Then two classes of PNs are presented: the synchronized PNs, a non-autonomous DES model, and labeled PNs, an autonomous DES model.

For more details on PNs the reader is referred to [93, 32].

2.1.2.1 Place/Transition nets

A Petri net (PN), or more properly a *Place/Transition net*, is a structure

$$N = (P, T, Pre, Post),$$

where:

- P is the set of m places;
- T is the set of q transitions;
- $Pre : P \times T \to \mathbb{N}$ is the pre-incidence function;
- $Post : P \times T \to \mathbb{N}$ is the post incidence functions.

Pre and $Post$ matrices specify the weighted arcs and the resulting matrix $C = Post - Pre$ is commonly called the incidence matrix.

Note that $Pre(\cdot, t_i)$ denotes the restriction to Pre to $P \times t_i$, i.e., the i-th column of the pre-incidence matrix. Analogously, $Pre(p_j, \cdot)$ denotes the restriction to Pre to $p_j \times T$, i.e., the j-th row of the pre-incidence matrix.

A *marking* specifies the state of the system described by the PN. A marking is a vector $M : P \to \mathbb{N}$ that assigns to each place of a P/T net a nonnegative integer number of tokens, represented by black dots. The marking of place p, i.e. the number of tokens contained in p, is denoted as $M(p)$. A marked P/T net $\langle N, M_0 \rangle$ is a net N with an initial marking M_0.

The evolution of the net coincides with a marking evolution, which is caused by firing of transitions. A transition t is enabled at M iff $M \geq Pre(\cdot, t)$ and may fire yielding the marking $M' = M + C(\cdot, t)$. The set of transitions enabled at M is denoted $\mathcal{E}(M)$. One writes $M\ [\sigma\rangle$ to denote that the sequence of transitions $\sigma = t_{j_1} \cdots t_{j_k}$ is enabled at M, and

$M\ [\sigma\rangle\ M'$ to denote that the firing of σ yields M'. One writes $t \in \sigma$ to denote that a transition t is contained in σ.

The set of all sequences that are enabled at the initial marking M_0 is denoted $L(N, M_0)$, i.e., $L(N, M_0) = \{\sigma \in T^* \mid M_0[\sigma\rangle\}$. Here symbol $*$ is the *kleen star* and T^* denotes the set of all firing sequences constructed over the set T of all transitions.

Given a sequence $\sigma \in T^*$, let $\pi : T^* \to \mathbb{N}^n$ be the function that associates to σ a vector $y \in \mathbb{N}^n$, called the *firing vector* of σ. In particular, $y = \pi(\sigma)$ is such that $y(t) = k$ if the transition t is contained k times in σ.

A marking M is *reachable* in $\langle N, M_0\rangle$ iff there exists a firing sequence σ such that $M_0\ [\sigma\rangle\ M$. The set of all markings reachable from M_0 defines the *reachability set* of $\langle N, M_0\rangle$ and is denoted $R(N, M_0)$.

The *preset* and *postset* of a place p are respectively denoted ${}^\bullet p$ and $p^\bullet$. The set of input transitions and the set of output transitions for a set of place $\hat{P}$ are defined as ${}^\bullet\hat{P} = \{t : \forall p \in \hat{P},\ t \in {}^\bullet p\}$ and $\hat{P}^\bullet = \{t : \forall p \in \hat{P},\ t \in p^\bullet\}$.

Analogously, the *preset* and *postset* of a transition t are respectively denoted ${}^\bullet t$ and $t^\bullet$. Also, the set of input places and the set of output places for a set of transitions $\hat{T}$ are defined as ${}^\bullet\hat{T} = \{p : \forall t \in \hat{T},\ p \in {}^\bullet t\}$ and $\hat{T}^\bullet = \{p : \forall t \in \hat{T},\ p \in t^\bullet\}$.

Let us give the definition of directed path.

Definition 2.7. ***(Directed path)*** *Given a SM PN $N = (P, T, Pre, Post)$, an alternated sequence of places and transitions $\rho = \langle p'_0 t'_1 p'_1 t'_2 \cdots t'_r p'_r\rangle$ is called a* directed path *if $\forall i = 0, 1, \ldots r$ and $\forall j = 1, \ldots r$ it holds: i) $p'_i \in P$ and $t'_j \in T$; ii) $p'_{j-1} \in {}^\bullet t'_j$ and $t'_j \in {}^\bullet p'_j$. A path non-containing any repeated place is called* elementary. ■

A path which ends at the vertex it begins is called *circuit*. A PN having no directed circuits, is called *acyclic*.

Finally, consider the following definition.

Definition 2.8. ***(T–induced subnets)*** *Given a net $N = (P, T, Pre, Post)$, and a subset $T' \subseteq T$ of its transitions, let us define the T'–*induced subnet of N *as the new net $N' = (P, T', Pre', Post')$ where $Pre', Post'$ are the restrictions of $Pre, Post$ to T'. The net N' can be thought as obtained from N removing all transitions in $T \setminus T'$. Let us also write $N' \prec_{T'} N$.* ■

Particular types of PNs can be obtained by restricting the syntax in a particular way. Recall that *ordinary PNs* are nets where all arc weights are 1's. Restricting further, the following types of ordinary Petri nets are commonly used and studied: *state graph* or more properly *state machine*, *event graph*, *conflict free PN* , *free choice PN* , *simple PN*. Note that the previous classes of PNs are not disjoint.

In the following we focus on the class of *state machines*.

Definition 2.9. ***(State machine PN)***[93] *A* state machine *(SM) PN is an ordinary PN such that each transition t has exactly one input place and exactly one output place,*

i.e.,

$$|{}^\bullet t| = |t^\bullet| = 1 \qquad (\forall t \in T) \qquad \blacksquare$$

One observes that a SM $N = (P, T, Pre, Post)$ may also be represented by an *associated graph* $\mathcal{G}_N = (V, A)$ whose set of vertices $V = P$ coincides with set of places of the net, and whose set of arcs A corresponds to the set of transitions of the net, i.e., $A \subseteq P \times P = \{(p_i, p_j) \mid \exists t \in T, p_i = {}^\bullet t, p_j = t^\bullet\}$.

Such a graph can be partitioned into its maximal strongly connected components, analogously to the automata. These components induce also a partition of the set of places of the associated SM.

Definition 2.10 (Associated graph)**.** *Given a SM $N = (P, T, Pre, Post)$, let $\mathcal{G}_N = (P, A)$ be its associated graph. P can be partitioned into* components *as follows:*

$$P = P_1 \uplus \cdots \uplus P_k$$

such that for all $i = 1, \ldots, k$ and $A_i = A \cap (P_i \times P_i)$ it holds that (P_i, A_i) is a maximal strongly connected sub-graph of $\mathcal{G}_N$. $\blacksquare$

As discussed in Section 2.1.1, components $P_1, \ldots P_k$ can be classified as transient or ergodic components.

For a SM, analogously to the automata, a condensed graph can be constructed for a given associated graph, following Definition 2.6.

Example 2.11. *Consider Figure 2.3a, where an example of SM which is not strongly connected is shown. Transient and ergodic components are respectively identified by dashed and dotted boxes. For such a net the transient components are $TR_1 = \{p_1,\ p_4\}$, $TR_2 = \{p_3\}$, whereas the ergodic components are $ER_1 = \{p_5,\ p_6\}$ and $ER_2 = \{p_2\}$. The corresponding $\mathcal{C}(N)$ is shown in Figure 2.3b, where subnets induced by each component are represented by single nodes.* $\blacksquare$

2.1.2.2 Synchronized Petri nets

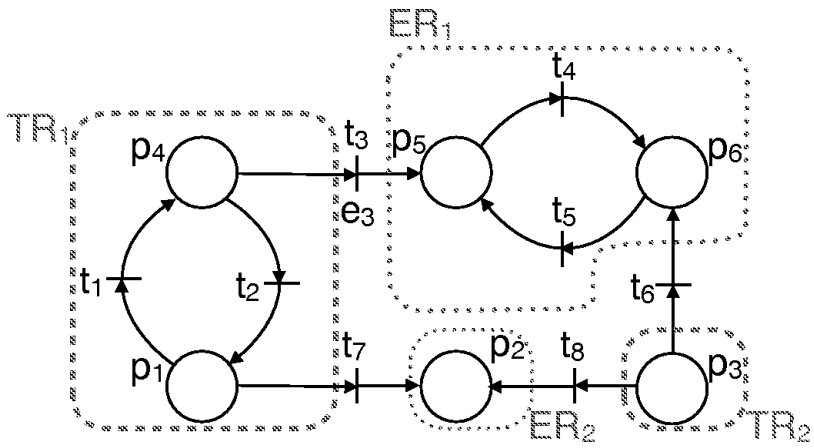
The synchronized PNs belong to the category of non-autonomous PNs. They describe systems whose evolution is dictated by external events whose occurrence has no duration.

A *synchronized PN* [32] is a structure

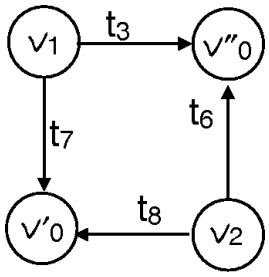
$$\langle N, E, f \rangle$$

such that: i) N is a P/T net; ii) E is an input alphabet of external events; iii) $f : T \to E$ is a labeling function that associates with each transition t an input event $f(t)$.

When there exists a one-to-one mapping between the alphabet of external events and the set of transition, i.e. each event is associated with a different single transition, the net is called *totally synchronized*.



(a)



(b)

Figure 2.3: A not strongly connected SM with two transient components and two ergodic ones (a) and its corresponding condensed graph.

Given an initial marking M_0, a marked *synchronized PN* is a structure $\langle N, M_0, E, f \rangle$.

One extends the labeling function to sequences of transitions as follows: if $\sigma = t_1 t_2 \dots t_k$ then $f^*(\sigma) = f(t_1) f(t_2) \dots f(t_k)$.

The set T_e of transitions associated with the input event e is defined as follows: $T_e = \{t \mid t \in T, f(t) = e\}$. One equivalently says that all enabled transitions in T_e are receptive to input event e.

The evolution of a synchronized PN is driven by input sequences as follows. At marking M, transition $t \in T$ is fired iff:

1. it is enabled, i.e., $t \in \mathcal{E}(M)$;
2. the event $e = f(t)$ occurs.

On the contrary, the occurrence of an event associated with a transition $t \notin \mathcal{E}(M)$ does not produce any firing. Note that a single server semantic is here adopted [91], i.e., when input event e occurs, the enabled transitions in T_e fire only once regardless of their enabling degree. However, a more general model, called *extended synchronized PN* [32], consider q multiple firings of a transition that is q–enabled.

Note that synchronized PNs with single-server semantics and with infinite-server semantics are proven to have the same modeling power [91], since their behavior can be equivalently modeled by one of these models.

Enabled transitions associated with event e are denoted $\mathcal{E}_e(M) = \{t : t \in T_e \bigcap \mathcal{E}(M)\}$.

Several transitions can be fired on the occurrence of a single event. Since simultaneous firings are not allowed, there can be one sequential net-evolution identifying a set of possible sequences of firing transitions. Such a set is called *Step*.

Definition 2.12. ***(Steps)*** *Given a marked synchronized PN $\langle N, M_0, E, f\rangle$, the set of transition sequences $\Sigma_e(M)$ is a step for event $e \in E$ and marking $M \in R(N, M_0)$, if the following four conditions hold:*

i) *any $\sigma \in \Sigma_e(M)$ is a feasible firing transition sequence at marking M, i.e. $Pre \cdot y \leq M$, having $y = \pi(\sigma)$;*

ii) *every transition in any $\sigma \in \Sigma_e(M)$ is such that $y(t) = 0$ if $t \notin T_e$ and $y(t) \leq 1$ otherwise;*

iii) *any sequence σ', obtained by permuting transitions in σ, still belongs to $\Sigma_e(M)$;*

iv) *there exists no sequence σ'' meeting conditions i) to iii) such that $y'' = \pi(\sigma'')$ and $y'' \ngeq y$.* ■

In the following, for a given marking M, any step is denoted as $\Sigma_e(M) = [t_1, t_2, \ldots, t_k]$, where the square brackets are used to denote that the order of the firing of transitions can be any whatsoever. For instance $[t_1, t_2, t_3] = \{t_1t_2t_3,\ t_1t_3t_2,\ t_2t_1t_3,\ t_2t_3t_1,\ t_3t_1t_2,\ t_3t_2t_1\}$.

The application of input event sequence $w = e_1 \ldots e_k$ from M yields marking M' and this is denoted $M \xrightarrow{w} M'$.

In Figure 2.4a is shown an example of synchronized PN. Note that labels next to each transition denote its name and the associated input event.

Definition 2.13. ***(Effective conflict)*** *There is an* effective conflict *at a marking M between two enabled transitions $t, t' \in \mathcal{E}(M)$ if the following relation holds:*

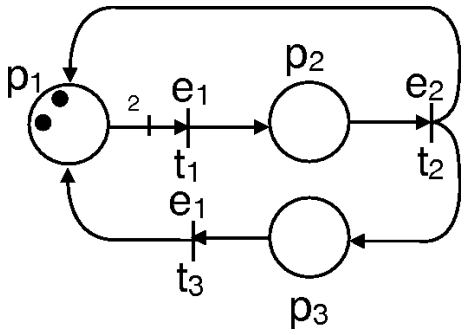
$$\exists p : t, t' \in p^{\bullet},\ M(p) < Pre(p, t) + Pre(p, t').$$ ■

Definition 2.14. ***(Deterministic PN)*** *A synchronized PN $\langle N, M_0, E, f\rangle$ is said to be* deterministic *if the following relation holds:*

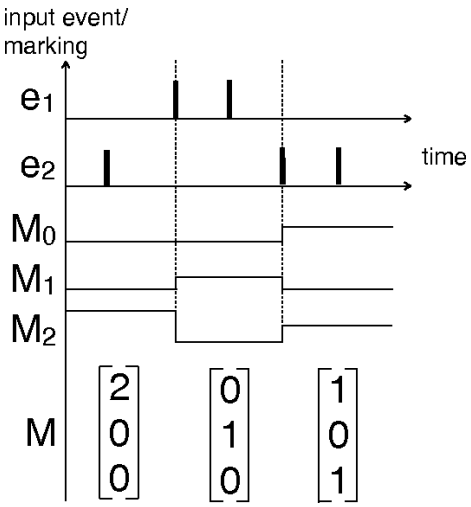
$$\forall M \in R(N, M_0), \forall e \in E :\ M \geq \sum_{t \in T_e \bigcap \mathcal{E}(M)} Pre(\cdot, t).$$ ■

In other words, a synchronized PN is said to be deterministic if for all reachable markings there is no effective conflict between enabled transitions sharing the same event.

The following sufficient condition provides a structural characterization of determinism in synchronized PNs.



(a)



(b)

Figure 2.4: A synchronized PN (a) and a possible behavior (b).

Proposition 2.15. *A synchronized PN* $\langle N, M_0, E, f \rangle$ *is deterministic if there does not exist a place* p *such that* $t, t' \in p^\bullet$ *and* $f(t) = f(t')$.

Proof. The existence of a place p such that $t, t' \in p^\bullet$ and $f(t) = f(t')$ is a necessary condition for non-determinism according to Definitions 2.13 and 2.14. □

In the rest of this thesis we will only deal with the class of synchronized PNs that satisfy this above structural deterministic condition.

When an event occurs in a deterministic net, all enabled transitions receptive to that event can simultaneously fire. Thus an input sequence $w = e_1 e_2 \cdots e_k \in E^*$ drives a deterministic net through the sequence of markings M_0, M_1, M_2, $\cdots$, M_k where M_0 is the initial marking and

$$M_{i+1} = M_i + \sum_{t \in T_{e_{i+1}} \cap \mathcal{E}(M_i)} (Post(\cdot, t) - Pre(\cdot, t)) .$$

The evolution of a synchronized PN, assuming that Proposition 2.15 holds, is defined by the following algorithm. This is specified upon any input event occurrence.

Algorithm 2.16. ***(Simulation of a deterministic synchronized PN)***
Input: *a synchronized PN* $\langle N, E, f \rangle$ *an initial marking* M_0.
Output: *updated marking* M'*, reached by the occurrence of a stream of input events.*

1. *Initialization of* M' *to* M_0*. Wait for new event occurrences.*

2. ***When*** *a new event* e *occurs,* ***do***

2.1. *let* $\mathcal{E}_e(M')$ *be the set of transitions associated with event* e *and enabled at marking* M'.

2.2. ***If*** $\mathcal{E}_e(M') \neq \emptyset$, ***do***

2.2.1. $M'' = M' + \sum_{t \in \mathcal{E}_e(M')} (Post(\cdot, t) - Pre(\cdot, t))$ *be the updated marking by applying input event* e *at* M'*, i.e. by firing all transition in* $\mathcal{E}_e(M')$.

Else,

2.2.2. $M'' = M'$.

End if

2.3. $M' = M''$.

End while ■

In Figure 2.4b an example of net evolution for the synchronized PN in Figure 2.4a is presented over a possible input sequence $w = e_2e_1e_1e_2e_2$ starting from marking M_0.

Example 2.17. *Consider the deterministic PN of Figure 2.4a and let the current marking be* $M = [2\,0\,1]^T$*. Transitions* t_1 *and* t_3 *are both associated with event* e_1*. They are both said to be receptive to event* e_1*, since they are both enabled and upon the occurrence of event* e_1 *will be fired, yielding marking* $M' = [1\,1\,0]^T$*. Note that markings* $[0\,1\,1]^T$ *and* $[3\,0\,0]^T$*, respectively obtained by the independent firing of* t_1 *and* t_3*, are never reachable.*

Let now the current marking be again M *and consider the occurrence of event* e_2*. Although* t_2 *is synchronized on* e_2*, it is not fired because not enabled when the event occurs.* t_2 *is said to be not receptive to this event at marking* M. ■

As a consequence, the reachability set of a synchronized PN could be either a subset or equal to the set of the reachable markings of the corresponding P/T net, depending on the labeling function.

A marked PN $\langle N, M_0 \rangle$ is said to be bounded if there exists a positive constant k such that for all $M \in R(N, M_0)$, $M(p) \leq k \ \forall p \in P$. A place in Petri net is called k-bounded if it does not contain more than k tokens in all reachable markings, including the initial marking; it is said to be safe if it is 1-bounded.

A bounded net has a finite reachability set. In this case, the behavior of the net can be represented by the *reachability graph* (RG), a directed graph whose vertices correspond to reachable markings and whose edges correspond to the transitions causing a change of marking. In the case of synchronized PNs it is common to show the event next to the

arc. The RG is obtained by exhaustively applying Algorithm 2.16, i.e. computing all markings reached by the application of every possible input event starting from the given initial marking.

In the following algorithm the reachability tree (RT) construction is provided.

Algorithm 2.18. ***(Reachability tree construction)***
Input: *a deterministic synchronized PN* $\langle N, E, f \rangle$ *and the initial marking* M_0.
Output: *a reachability tree* $\tilde{\mathcal{T}}$.

1. *Label the root node* q_0 *with the initial marking* M_0 *and tag it "new".*

2. ***While*** *a node tagged "new" exists* ***do***

2.1. *Select a node* q *tagged "new" and let* M *be its label.*

2.2. ***For*** *all* $e \in E$*:*

2.2.1. *Let* $M' = M + \sum_{t \in \mathcal{E}_e(M)} (Post(\cdot, t) - Pre(\cdot, t))$ *be the marking reached firing all enabled* $t \in T_e$.

2.2.2. *Add a new node* q' *and label it* M'.

2.2.3. *Add an arc labeled* $e|\Sigma_e(M)$ *from* q *to* q'.

2.2.4. ***If*** *there exists already in the tree a node with label* M',

2.2.4.1. *tag node* q' *"duplicate",*

else,

2.2.4.2. *tag it "new".*

End if

End for

2.3. *Untag node* q.

End while ■

From the RT one can obtain the RG by fusing duplicate nodes with the untagged node with the same label: any RT can always be converted in its corresponding graph and vice versa.

The initial marking, together with the firing rules, characterizes the RG via Algorithm 2.18. It is important to note that a different initial marking may give a completely different reachability set as shown in the following example.

Example 2.19. *Consider the PN in Figure 2.5a. Let the initial marking be* $M_0 = [1\,0\,0]^T$. *Its RG is shown in Figure 2.5b. Let now initial marking be* $M_0 = [2\,0\,0]^T$. *This new setting allows a different reachability set and then a different RG, as depicted in Figure 2.5c.* ■

Note that in a RG there is a one-to-one mapping between a node and its label, that are used without making any distinction in the rest of the thesis.

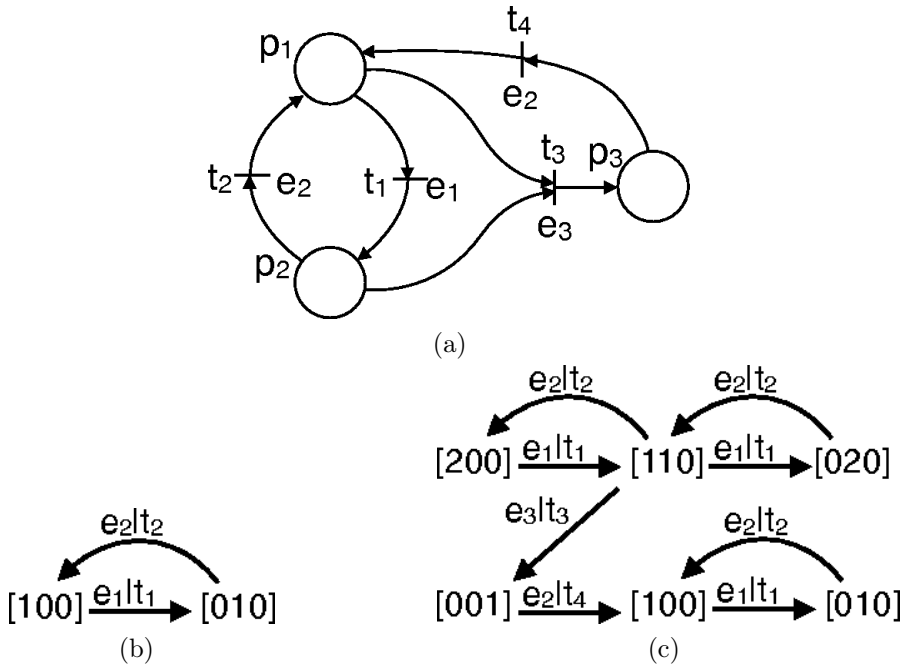


Figure 2.5: A synchronized PN (a) and its RGs for 1 token (b) and 2 tokens (c)

The set of reachable markings of a PN is not necessarily finite. Karp and Miller [69] have proposed an algorithmic computation of a finite representation of the state-space of unbounded PNs.

In particular, this algorithm is based on an acceleration technique, which computes sequences of transitions that strictly increase the number of tokens in certain places. These sequences are called *increasing sequences*. The acceleration technique works because PNs are strictly monotonic, i.e. a sequence of transitions which fires from a marking M fires from all markings M' such that $M' \geqq M$.

This representation, called Coverability Graph (CG) is not unique and certainly not minimal. So a minimal CG has been proposed by Finkel [45], using reduction rules based on comparison between computed markings. The approach has been demonstrated to be incorrect and more efficient techniques have been set to correctly determine minimal coverability sets by constructing handle sets by Geeraerts *et al.* [50] of pair markings or by pruning technics by Reynier and Servais [107].

However, the CG entails loss of information in terms of reachable markings and firing sequences, that can somehow prevent one to use it for systematically investigating the net properties. In this context, a different CG construction has been proposed by Mengchu *et al.* [129, 36]. Their modified reachability tree allows to determine whether an unbounded PN has deadlocks or not, but the graph contains also non reachable markings, so that no other properties such as finiteness and liveness can be checked.

Each node of the CG is labeled with an ω-marking, defined as follows.

Definition 2.20. ***(ω–marking)*** *Let $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$. An ω–marking of a PN N with m places is a column vector $M_\omega \in \mathbb{N}_\omega^m$, whose components may either be an integer number*

or be equal to ω. ■

Symbol ω denotes that the marking of the corresponding place may grow indefinitely. Note that for all $n \in \mathbb{N}$ it holds $\omega > n$ and $\omega \pm n = \omega$.

Let P_u and I_u (resp. P_b and I_b) denote the set of unbounded (resp. bounded) places and the corresponding indexes s.t. $I_u = \{i : p_i \in P_u\}$ (resp. $I_b = \{i : p_i \in P_b\}$). Let $m_u = |P_u|$ and $m_b = |P_b|$. We denote $M \uparrow_b$ (resp. $M \uparrow_u$) the *projection* of the marking M onto the set of bounded (resp. unbounded) places P_b (resp. P_u).

The notion of covering set is now introduced, that is a (not necessarily strict) superset of $R(N, M_0)$.

Definition 2.21. ***(Covering set)*** *Given a marked net* $\langle N, M_0 \rangle$*, let* V *be the set of nodes of its CG. The covering set of* $\langle N, M_0 \rangle$ *is*

$$CS(N, M_0) = \bigcup_{M_\omega \in V} cov(M_\omega),$$

where $cov(M_\omega) = \{M \in \mathbb{N}^m : M(p) = M_\omega(p) \text{ if } M_\omega(p) \neq \omega\}$. ■

A marking M_ω is called a *covering marking* for M if $M \in cov(M_\omega)$. In this case, we write $M_\omega \geq_\omega M$.

In the framework of synchronized PNs, David and Alla have used the Karp and Miller algorithm [69] to construct the coverability tree to visualize the infinite state-space of a synchronized PN.

Such an algorithm can be presented as an extension of Algorithm 2.18 for the bounded case. In particular, between step 2.2.1 and step 2.2.2, marking M' is updated following the following condition.

- *Let* $\bar{Q}$ *be the set of nodes* $\bar{q}_i$ *met on a backward path from* q *to* q_0 *whose label is* $\bar{M}_i \not\leq M'$.
  If *any of such nodes exists* ***then*** *for all* $p \in P$ *such that* $M'(p) > \bar{M}_i(p)$ *let* $M'(p) = \omega$.

Note that a place containing ω tokens cannot be emptied. This is not possible since single firings are considered.

A place p is said to be unbounded if $\exists M \in R(N, M_0)$ s.t. $M(p) = \omega$. Symbol ω replaces the computed value whenever a sequence of events w leads from some $\bar{q} \in \bar{Q}$ to q. This sequence, denoted as *increasing input sequence*, makes the marking of the corresponding place grow.

Definition 2.22. ***(Repetitive input sequence)*** *Consider a marked synchronized PN* $\langle N, M_0, E, f \rangle$*. An input sequence* $w \in E^*$ *is called* repetitive *if there exists a marking* $M_1 \in R(N, M_0)$ *such that*

$$M_1[w|\sigma\rangle M_2[w|\sigma\rangle M_3[w|\sigma\rangle \cdots \tag{2.1}$$

i.e. if it can fire infinitely often starting from M_1*, always producing the same firing of transitions. It is possible to distinguish two different types of repetitive sequence:*

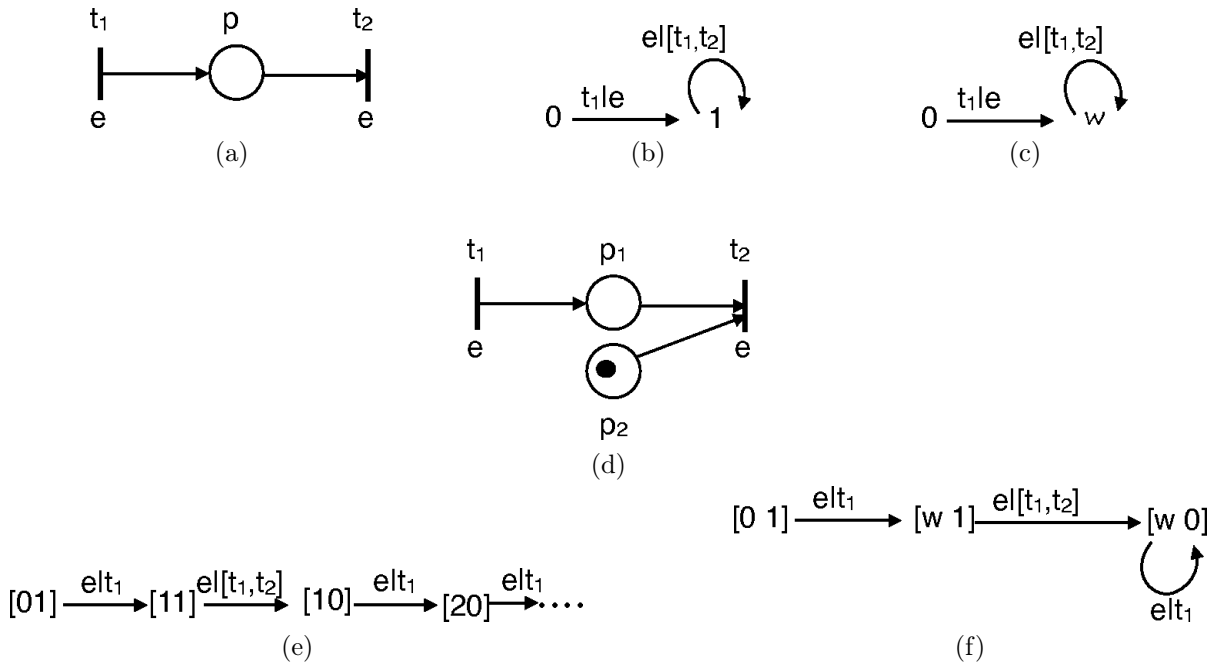


Figure 2.6: Examples of coverability graph construction (1).

- stationary input sequence*: if in 2.1 it holds* $M_i = M_{i+1}$ *for all* $i = 1, 2, \ldots$
- increasing input sequence*: if in 2.1 it holds* $M_i \lneqq M_{i+1}$ *for all* $i = 1, 2, \ldots$ ■

This algorithm, as later pointed out by Devillers and Begin [34], does not correctly construct the evolution of a synchronized net.

In fact, obtaining an increased marking after the application of an input sequence is only a necessary condition to be an increasing sequence, due to the non-necessarily monotonic evolution of the net. Such a condition must hold for every successive application of w.

As the same authors have later shown [32], this algorithmic construction does not work in all cases and so far the coverability graph could be obtained just thanks to pertinent reasoning.

In fact, monotonicity is crucial to the automated verification of nets and other *well structured transition systems* and synchronized PNs are not monotonic. For instance consider a marking enabling a certain evolution under an input sequence. A larger marking, under the same input event application may not allow the same evolution.

That is why in general, the unbounded places of the net and places denoted as unbounded in the $CG(N, M_0)$ do not coincide. In the following, some examples of CG, constructed by adding the above condition to Algorithm 2.18, are first presented.

Example 2.23. *Consider the PN in Figure 2.6a. Let* $M_0 = [0]$ *be the initial marking. The only possible input event is* e *and the reachable marking are shown in Figure 2.6b. Hence* p *is bounded in* N*. The CG in Figure 2.6c has a path* $M[e|t_1\rangle\omega[e|\{t_1, t_e\}\rangle\omega \ldots$,

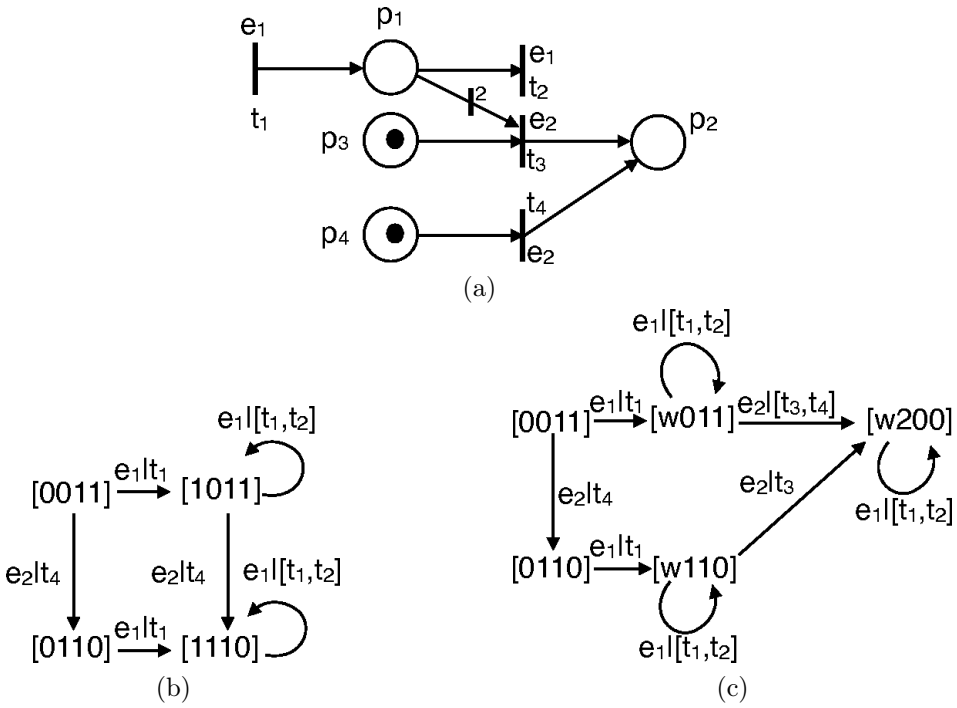


Figure 2.7: Examples of coverability graph construction (2).

hence p is unbounded in that graph. Thus, the Algorithm is proven to fail in determining increasing input sequences of Definition 2.22. ■

Monotonicity is generally assumed while constructing a covering graph. Here firing rules are shown to be non-monotonic.

Example 2.24. *Consider the PN in Figure 2.6d. Let* $M_0 = [0\,1]^T$ *be the initial marking. If the input event e occurs at the initial marking* $M = [0\,1]^T$*, transition* t_1 *will fire yielding marking* $M' = [1\,1]^T \ngeq M$*. A second occurrence of event e would make both transitions* t_1 *and* t_2 *fire, yielding marking* $M'' = [1\,0]^T \not\geq M'$*. This shows that the firing rule is not monotonic in the reached markings, since the underlying firing transition sequences are not the same under the same input event application.* ■

Example 2.25. *Consider the PN in Figure 2.7a. Let* $M_0 = [0\,0\,1\,1]^T$ *be the initial marking. The net is bounded but its CG is not. For instance, the sequence* e_1e_2 *drives the net from* M_0 *to the bounded marking* $M = [1\,1\,1\,0]^T$*. However, in the CG shown in Figure 2.7c, this same sequence leads the net from* M_0 *to an unbounded marking, e.g.* $M' = [\omega\,2\,0\,0]^T$*, that does not cover any of the reachable markings of the net.* ■

Under this consideration one can clearly say that the previously so-constructed CG cannot be used for property verification of the net.

2.1.2.3 Labeled Petri nets

The previous syntactic definition of synchronized PNs is also common to the so-called *labeled PNs* [49]. However, while in the case of labelled PNs the evolution is autonomous and the events are usually interpreted as outputs, in the case of synchronized nets the events are inputs that drive the net evolution as explained in the following.

Furthermore, as shown later, in a synchronized nets two or more transitions can simultaneously fire, while this is not possible for labelled nets.

When observing the evolution of a net, it is common to assume that each transition t is assigned a label $\varphi(t)$ and the occurrence of t generates an observable output $\varphi(t)$. If $\varphi(t) = \varepsilon$, i.e., if the transition is labeled with the empty string, its firing cannot be observed.

Thus the association between sensors and transitions can be captured by a *labeling function* $\mathcal{L} : T \to L \cup \{\varepsilon\}$.

Definition 2.26. ***(Labeled PNs)*** *Given a PN* $N = (P, T, Pre, Post)$*, a* labeling function $\varphi : T \to E \cup \{\varepsilon\}$ *assigns to each transition* $t \in T$ *a symbol, from a given alphabet* E*, or assigns to it the empty string* ε*.*

A marked labeled *PN is a 3-tuple* $G = \langle N, M_0, \varphi \rangle$ *where* $N = (P, T, Pre, Post)$*,* M_0 *is the initial marking and* $\varphi : T \to E \cup \{\varepsilon\}$ *is the labeling function.* ■

Four classes of labeling functions may be defined.

Definition 2.27. ***(Labeling functions)*** *The labeling function of a labeled PN system* $\langle N, M_0, \varphi \rangle$ *can be classified as follows.*

- Free*: if all transitions are labeled distinctly, namely a different label is associated to each transition, and no transition is labeled with the empty string.*
- Deterministic*: if no transition is labeled with the empty string, and the following condition*[1] *holds: for all* $t, t' \in T$*, with* $t \neq t'$*, and for all* $M \in R(N, M_0)$*:* $M[t\rangle \wedge M[t'\rangle \Rightarrow [\varphi(t) \neq \varphi(t')]$ *i.e., two transitions simultaneously enabled may not share the same label. This ensures that the knowledge of the firing label* $\varphi(t)$ *is sufficient to reconstruct the marking that the firing of* t *yields.*
- λ-free*: if no transition is labeled with the empty string*[2]*.*
- Arbitrary*: if no restriction is posed on the labeling function* φ*.* ■

[1] A less restrictive condition can be given: for all $t, t' \in T$, with $t \neq t'$, and for all $M \in R(N, M_0)$: $M[t\rangle \wedge M[t'\rangle \Rightarrow [\varphi(t) \neq \varphi(t')] \vee [Post(\cdot, t) - Pre(\cdot, t) = Post(\cdot, t') - Pre(\cdot, t')]$. Thus two transitions with the same label may be simultaneously enabled at a marking M, if the two markings reached from M by firing t and t' are the same.

[2] In the PN literature the empty string is denoted λ, while in the formal language literature it is denoted ε. In this thesis we denote the empty string ε but, for consistency with the PN literature, we still use the term λ*-free* for a non erasing labeling function $\varphi : T \to E$.

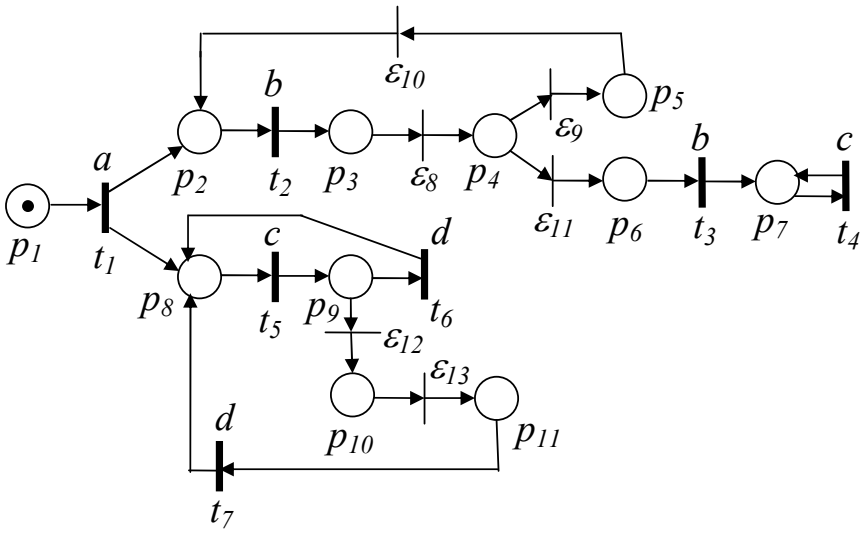


Figure 2.8: A labeled marked PN

Each of these types of labeling is a generalization of the previous one.

In the particular case in which the labeling function is free, being an isomorphism between the alphabet E and the set of transitions T, it is usual to choose $E = T$, or equivalently to assume that the transitions are not labeled and their firing can be directly observed.

The set of transitions whose label is ε is denoted as T_u, i.e., $T_u = \{t \in T \mid \mathcal{L}(t) = \varepsilon\}$. Transitions in T_u are called *unobservable* or *silent*. T_o denotes the set of transitions labeled with a symbol in L. Transitions in T_o are called *observable* because when they fire their label can be observed. Note that in this thesis it is assumed that the same label $l \in L$ can be associated to more than one transition. In particular, two transitions $t_1, t_2 \in T_o$ are called *undistinguishable* if they share the same label, i.e., $\mathcal{L}(t_1) = \mathcal{L}(t_2)$. The set of transitions sharing the same label l are denoted as T_l.

In the following let C_u (C_o) be the restriction of the incidence matrix to T_u (T_o) and n_u and n_o, respectively, be the cardinality of the above sets. Moreover, given a sequence $\sigma \in T^*$, $P_u(\sigma)$, resp., $P_o(\sigma)$, denotes the projection of σ over T_u, resp., T_o.

The word w of events associated to sequence σ is $w = P_o(\sigma)$. Note that the length of a sequence σ (denoted $|\sigma|$) is always greater than or equal to the length of the corresponding word w (denoted $|w|$). In fact, if σ contains k' transitions in T_u then $|\sigma| = k' + |w|$.

Definition 2.28. *[17]**(Consistent firing sequence and consistent markings)** Let $\langle N, M_0\rangle$ be a labeled marked net with labeling function $\mathcal{L} : T \to L \cup \{\varepsilon\}$, where $N = (P, T, Pre, Post)$ and $T = T_o \cup T_u$. Let $w \in L^*$ be an observed word. Let*

$$\mathcal{S}(w) = \{\sigma \in L(N, M_0) \mid P_o(\sigma) = w\}$$

be the set of firing sequences consistent *with $w \in L^*$, and*

$$\mathcal{C}(w) = \{M \in \mathbb{N}^m \mid \exists \sigma \in T^* : P_o(\sigma) = w \wedge M_0[\sigma\rangle M\}$$

be the set of reachable markings consistent *with $w \in L^*$.* ■

In other words, given an observation w, $\mathcal{S}(w)$ is the set of sequences that may have fired, while $\mathcal{C}(w)$ is the set of markings in which the system may actually be.

Example 2.29. *Consider the PN in Figure 5.2. Assume* $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ *and* $T_u = \{\varepsilon_8, \varepsilon_9, \varepsilon_{10}, \varepsilon_{11}, \varepsilon_{12}, \varepsilon_{13}\}$*, where for a better understanding unobservable transitions have been denoted* ε_i *rather than* t_i*. The labeling function is defined as follows:* $\mathcal{L}(t_1) = a$*,* $\mathcal{L}(t_2) = \mathcal{L}(t_3) = b$*,* $\mathcal{L}(t_4) = \mathcal{L}(t_5) = c$*,* $\mathcal{L}(t_6) = \mathcal{L}(t_7) = d$*.*

First consider $w = ab$*. The set of firing sequences that is consistent with* w *is* $\mathcal{S}(w) = \{t_1t_2, t_1t_2\varepsilon_8, t_1t_2\varepsilon_8\varepsilon_9, t_1t_2\varepsilon_8\varepsilon_9\varepsilon_{10}, t_1t_2\varepsilon_8\varepsilon_{11}\}$*, and the set of markings consistent with* w *is* $\mathcal{C}(w) = \{[0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0]^T, [0\,0\,0\,1\,0\,0\,0\,1\,0\,0\,0]^T, [0\,0\,0\,0\,1\,0\,0\,1\,0\,0\,0]^T, [0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0]^T, [0\,0\,0\,0\,0\,1\,0\,1\,0\,0\,0]^T\}$*.*

If $w = acd$ *is considered the set of firing sequences that are consistent with* w *is* $\mathcal{S}(w) = \{t_1t_5t_6, t_1t_5\varepsilon_{12}\varepsilon_{13}t_7\}$*, and the set of markings consistent with* w *is* $\mathcal{C}(w) = \{[0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0]^T\}$*. Thus two different firing sequences may have fired (the second one also involving silent transitions), but they both lead to the same marking.* ■

2.2 Test of Discrete Event Systems

Due to always larger size and rising complexity, systems needs of checking performances increase and testing problems periodically resurface.

These problems have been introduced by the pioneering paper of Moore [92], where the main focus is to understand which type of conclusions it is possible to draw about internal conditions of the system under test from external experiments.

2.2.1 Fundamental testing problems

In a testing problem we have a machine about which we lack some information; we would like to deduce this information by providing a sequence of inputs, called test or experiment, to the machine and observing the outputs produced.

A test can be *preset*, if the sequence is entirely provided before its application, or *adaptive*, if each input event of the sequence is determined step by step regarding the previously observed outputs. An adaptive test might be seen as a decision tree rather than a test sequence. Adaptive tests are associated with sequences of shorter length, but their algorithms are more complicated than the ones for the preset case. In fact, sometimes the main drawback of using adaptive experiments is designing them.

In his *gedanken-experiment*, i.e., thought-experiment, Moore stated that the system under investigation is a fixed semiautomaton seen as a black box. Lee and Yannakakis in [76] have widely reviewed those problems and the techniques to solve them. They have stated five fundamental problems of test.

In the three first problems, it is given a complete description of the machine $M = (\chi, I, O, \delta, \lambda)$, but the current state is unknown.

i) homing/synchronizing sequences: determining the final state after a test;

ii) state identification: identify the unknown initial state;

ii) state verification: the machine is supposed to be in an initial state, verify that the hypothesis is indeed true;

In the last two problems, the machine M is tested as a black box, i.e. it can be viewed solely in terms of its input, output behavior, while it is given a *specification machine* $A = (\chi_A, I_A, O_A, \delta_A, \lambda_A)$.

iv) machine verification/fault detection/conformance testing: in addition to the machine M the experimenter is given another machine A and has to verify whether M is equivalent to A;

v) machine identification: identify the unknown machine M.

Problem i) has been solved in the 60s by the way of *finite state machines* (FSMs) with *homing sequences* (HS) and *synchronizing sequences* (SS).

In the HS problem it is given a machine for which we lack of information about its current state. The goal is to perform a test to drive it to a known state by observing the produced output. A HS can be constructed only for reduced FSMs, since equivalent states cannot be distinguished by using any test, and all FSMs have a HS. An input sequence is a HS if its block of the current state uncertainty are singletons. It is easy to see then that the shortest HS can be determined from the successor tree of the automaton [70]. Such a HS is determined by a node with least depth d which is labeled by a current state uncertainty consisting in singletons. Nevertheless constructing the successor tree up to depth d takes exponential time and finding shortest HSs is proved to be a NP-hard problem [92].

A SS takes the machine to a known state regardless of the initial state and without observing the outputs. Every SS is clearly a HS but it is not true the contrary. The synchronizing tree method [60, 70] have been proposed to provide shortest SSs. Analogously to the HS problem, such a method is suitable only for small size systems and becomes useless when the size grows. As a matter of fact the problem of finding shortest SSs is known to be NP-complete [40]. Two polynomial algorithms have been mainly used to provide a SS that is not necessarily the shortest one: the so-called *greedy* and *cycle* algorithms — respectively of Eppstein [40] and Trahtman [122] — that have equivalent complexity. Heuristic methods have been proposed by Roman [108], where he gives two measures — the average difference of length between the returned SSs and the shortest expected SS (seSS) and the ratio of returned SSs of length equal to the seSS — to compare his methods to the classic ones by numerical simulations.

Since the problem is the main topic of this thesis, section 2.2.2 focuses on the SS construction and the classic approach is explained in detail.

Problem ii) concerns performing an experiment apt to identify the unknown initial state. An input sequence that solves this problem is called distinguishing sequence. Note that a

HS solves a slightly different problem, since every distinguishing sequence also determines a HS, because the final state can be simply obtained starting from the initial one. Classic approaches [53, 54, 70] provide preset distinguishing sequences by the way of a type of successor tree, the so-called *distinguishing tree*. Each node of the distinguishing tree is labeled by the corresponding initial state uncertainty and it identifies a distinguishing sequence if all of its blocks are singletons. Unfortunately the existence of a preset distinguishing sequence is decidable but PSPACE-complete, that is why at least an exponential time is required to construct one. Besides adaptive distinguishing sequences can be determined via polynomial time algorithm based on decision trees [75]. Machines with preset sequences have also adaptive ones, that can also be shorter [118].

Problem iii) is an easier problem, in fact a sequence that solves it only verifies that the machine was in a given initial state. This is possible if and only if that state has a *unique input output sequence* (UIO) [117, 110]. A UIO sequence of a state is an input sequence such that the output sequence produced from any other state is different. This concept is known also like "simple I/O sequence" [63] and "checkword" [58]. An UIO sequence for a state x can still be performed by the way of a distinguishing tree and it is identified by a node containing an initial state uncertainty with a singleton block s. No efficient approach for determining such sequences have been given and only exponential-time algorithm tree-based known.

Problem iv) we have a *specification machine* A and a black-box machine M knowns as the *implementation machine*, for which only its input/output behavior can be observed. The goal is to perform a test sequence such that it is decidable if M is a good implementation of A, i.e. if M is equivalent (conforms) to A. This is easy and proved in [70, 92]. The problem is also known as machine verification or fault detection in the circuits and switching systems literature. In this field, people build test sequences using fault hypothesis on a model and apply these sequences to the real system in order to detect the considered fault. The implementation machine is supposed to be in an unknown state, we first resolve a HS/SS problem and then apply a sequence called checking experiment. This sequence verify if M conforms A by checking if their outputs match. The checking experiments are all provided under the same structure, every transition for every state is checked on M wrt A using subsequences. The existing methods differ by the kind of subsequences they use: distinguishing sequences, UIO sequences, characterizing sequences and identifying sequences. However, distinguishing and UIO sequences are mostly used [2, 21, 20, 26, 116, 61, 70].

Problem v) is a problem pretty much related to the conformance testing. Given an implementation machine M, seen as a black box, the purpose is to find a test sequence that allows to determine the transition diagram of M from its response to the test itself. The test sequence construction is clearly done under specific a priori knowledge of the machine such an upper bound of the space-state. The input and output alphabet, supposed respectively of cardinality p and q are also known. Standard approaches [92, 54] construct all $n-$state machines and perform conformance testing to find the equivalent machine. This encounters the space explosion problem since for n states, p inputs and q outputs one has $\frac{n \cdot p^{n \cdot p}}{n!}$ machines. This is partially solved by constructing a *direct sum machine* in which each state corresponds to a state in either the correct or in one of the "non-conforming" machines. Cross verifications with M are done by pairs of states and at least one machine

is then discharged. Practical applications can be found in communication protocol, where for instance one would track down the proprietary protocol standards by observing the implemented behavior [74].

These results can also be applied for conformance test of industrial logic controllers, when their specifications are given in standardized specification languages such *Sequential Function Chart* (SFC) [103, 102, 104]. In this setting, Provost *et al.* provide a method to obtain an equivalent and semantic loss-free automaton representation to deal with the conformance testing problem.

At present and only few works have been done in the area of testing for systems specified as PNs. The question of automatically testing PNs has been investigated by Jourdan and Bochmann in [46]. They have adapted methods originally developed for *Finite State Machines* (FSMs) and, classifying the possible occurring types of error, identified some cases where *free choice* and $1-safe$ PNs [93] provide more significant results especially in concurrent systems. Later the authors have extended their results also to $k-safe$ PNs [10]. Zhu and He have given an interesting classification of testing criteria [139] — without testing algorithms — and presented a theory of testing high-level Petri nets by adapting some of their general results in testing concurrent software systems.

In the PN modeling framework, one of the main supervisory control tasks is to guide the system from a given initial marking to a desired one similarly to the synchronization problem. Yamalidou *et al.* have presented a formulation based on linear optimization [135, 134] Giua *et al.* have investigated the *State identification problem*. They have proposed an algorithm to calculate an estimate — and a corresponding error bound — for the initial marking of a given PN based on the observation of an event sequence. A complete description of their approach can be found in [57], where labelled PNs are taken into consideration also in the case of silent transitions, i.e., transitions that do not produce any observation. Similar techniques are proposed by Lingxi *et al.* in [78] to get a minimum estimate of initial markings. Their aim is to characterize the minimum number of resources required at the initialization for a variety of systems.

As far as we know, the problem of providing synchronizing sequences has not been investigated via the Petri nets (PNs).

2.2.2 Synchronizing sequences using automata

In this section, the SS classic construction is presented by the aid of finite automata.

Definition 2.30. ***(SS on automata)*** *Consider an automaton with inputs* $\Lambda = (\chi, E, \delta)$ *and a state* $\bar{x} \in \chi$*. The input sequence* $\bar{w}$ *is called* synchronizing for state $\bar{x}$ *if it drives the automaton to* $\bar{x}$*, regardless of the initial state, i.e.,* $\forall x \in \chi$ *it holds that* $\delta(x, w) = \bar{x}$*.* ■

The information about the current state of Λ after applying an input sequence w is defined by the set $\phi(w) = \delta(\chi, w)$, called the *current state uncertainty of* w. In other words w is a synchronizing sequence (SS) that takes the automaton to the final state $\bar{x}$ iff $\phi(w) = \{\bar{x}\}$.

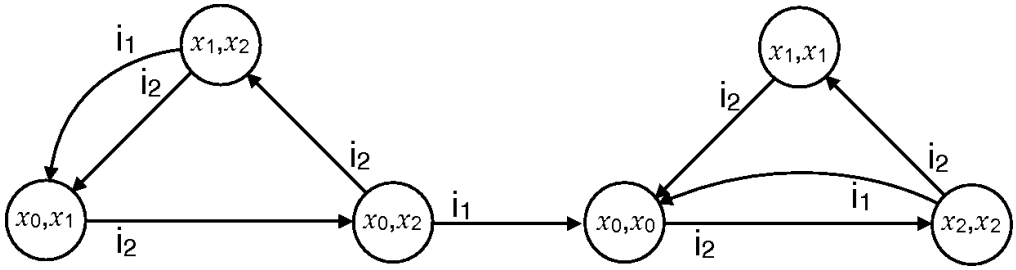


Figure 2.9: The auxiliary graph of the automaton in Figure 2.1.

The synchronizing tree method [60, 70] has been proposed to provide shortest SSs. Such a method is suitable only for small size systems, since the memory required to build up the tree is high, and becomes useless when the size grows. As a matter of fact the problem of finding shortest SSs is known to be NP-complete [40].

As previously said, *greedy* and *cycle* algorithms have been mainly used to provide a SS in polynomial time.

In fact, both algorithms work in $O(n_\chi^3 + |n_i|n_\chi^2)$ time, where n_χ and $|n_i|$ are, respectively, the number of states and the input alphabet cardinality of the automaton. Proofs can be respectively found in [40] and [122].

In particular, the *greedy* algorithm [40] determines an input sequence that takes a given automaton, regardless of its initial state, to a known target state: note that the target state is determined by the algorithm and cannot be specified by the user.

Here we propose a slightly different implementation of the greedy algorithm (see Algorithm 2.32), that takes as input also a state $\bar{x}$ and determines a sequence that synchronizes to that state.

This algorithm is later used as a building block to determine a SS to reach a given marking among those in the reachability set of a bounded PN.

Greedy and cycle algorithms are based on the auxiliary graph construction, defined as follows.

Definition 2.31. ***(Auxiliary graph)*** *Given an automaton with inputs Λ with n states, let $\mathcal{A}(\Lambda)$ be its* auxiliary graph. *$\mathcal{A}(\Lambda)$ contains $n(n+1)/2$ nodes, one for every unordered pair (x', x'') of states of Λ, including pairs (x, x) of identical states. There exists an edge from node (x', x'') to $(\hat{x}', \hat{x}'')$ labeled with input event $e \in E$ iff $\delta(x', e) = \hat{x}'$ and $\delta(x'', e) = \hat{x}''$.* ■

For instance, Figure 2.9 shows the auxiliary graph of the automaton in Figure 2.1.

Algorithm 2.32. ***(Greedy computation of SSs on automata with inputs)***
Input: An auxiliary graph $\mathcal{A}(\Lambda)$, associated with an automaton with inputs $\Lambda = (\chi, E, \delta)$, and a target state $\bar{x} \in \chi$.
Ouput: A SS $\bar{w}$ for state $\bar{x}$.

1. Let $i = 0$.

2. Let $w_0 = \varepsilon$, the empty initial input sequence.

3. Let $\phi(w_0) = \chi$, the initial current state uncertainty.

4. While $\phi(w_i) \neq \{\bar{x}\}$, do

4.1. $i = i + 1$.

4.2. Pick two states $x, x' \in \phi(w_{i-1})$ such that $x \neq x'$.

4.3. If there does not exist any path in $\mathcal{A}(\Lambda)$ from node (x', x'') to $(\bar{x}, \bar{x})$, stop the computation, there exists no SS for $\bar{x}$.

Else find the shortest path from node (x', x'') to $(\bar{x}, \bar{x})$ and let w be the input sequence along this path, do

4.3.1. $w_i = w_{i-1}w$

4.3.2. $\phi(w_i) = \delta(\phi(w_{i-1}), w)$.

End if

End while

5. $\bar{w} = w_i$. ■

A possible execution of the above algorithm can be explained by the following example.

Example 2.33. *Let state x_0 be the target state. Let $w_0 = \varepsilon$ be empty the initial input sequence and $\phi(w_0) = \{x_0, x_1, x_2\}$ the corresponding initial current state uncertainty. If at the first iteration of step 4.2 the algorithm picked states x_0 and x_2, it would obtain $w = i_1$, since $(x_0, x_2) \xrightarrow{i_1} (x_0, x_0)$. Hence $w_1 = i_1$ and $\phi(w_1) = \{x_0, x_1\}$, which does not satisfy the exiting condition of the while loop. At the second iteration the algorithm is forced to pick states x_0 and x_1. The shortest path is $(x_0, x_1) \xrightarrow{i_2} (x_0, x_2) \xrightarrow{i_1} (x_0, x_0)$, thus $w = i_2 i_1$, $w_2 = w_1 w = i_1 i_2 i_1$ and $\phi(w_2) = \{x_0\}$. The current state uncertainty is now singleton and corresponds to state x_0, so $\bar{w} = w_2$ is a SS for state x_0.* ■

The following theorem provides a necessary and sufficient condition for the existence of a SS for a target final state.

Theorem 2.34. *The following three propositions are equivalent.*

1. *Given an automaton with inputs $\Lambda = (\chi, E, \delta)$, there exists a SS for state $\bar{x} \in \chi$;*
2. *$\mathcal{A}(\Lambda)$ contains a path from every node (x', x''), where $x', x'' \in \chi$, to node $(\bar{x}, \bar{x})$;*
3. *Algorithm 2.32 determines a SS $\bar{w}$ for state $\bar{x} \in \chi$, if there exists any SS.*

Proof. [1) implies 2)] If there exists a SS for state $\bar{x} \in \chi$, there exists an input sequence w for $\bar{x}$ s.t. for any $x', x'' \in \chi$ it holds that $\delta(x', w) = \delta(x'', w) = \bar{x}$. Hence there exists a path labeled w from any (x', x'') to $(\bar{x}, \bar{x})$.

[2) implies 3)] Consider iteration i of the while loop of Algorithm 2.32. If there exists a path labeled w from any (x', x'') to $(\bar{x}, \bar{x})$, then it holds that $\delta(\{x', x''\}, w) = \{\bar{x}\}$. Hence the following inequality holds:

$$|\phi(w_i)| = |\delta(\phi(w_{i-1})\backslash\{x, x'\}, w) \cup \{\bar{x}\}| \leq |\phi(w_i)| - 1.$$

The existence of such a sequence for every couple of states $x', x'' \in \chi$ assures that the current state uncertainty will be reduced to singleton $\{\bar{x}\}$ after no more than n iteration.

[3) implies 1)] Since Algorithm 2.32 requires the current state uncertainty to be singleton and uses it as a stop criterium, if it terminates at step 5, then the sequence found is clearly a SS. □

One can easily understand that, when the automaton is not strongly connected, the above reachability condition will be verified only when there exists only one ergodic component (see Definition 2.5) and there may exist a SS only for those states belonging to this ergodic component.

2.3 Diagnosis of Discrete Event Systems

In this section the state of the art of diagnosis of DES using automata and PNs is presented.

2.3.1 Diagnosis with Automata

In the contest of DES several original theoretical approaches have been proposed using *automata*.

Lin *et al.* [81, 82] propose a state-based DES approach to failure diagnosis. The problems of off-line and on-line diagnosis are addressed separately and notions of diagnosability in both of these cases are presented. The authors give an algorithm for computing a diagnostic control, i.e., a sequence of test commands for diagnosing system failures. This algorithm is guaranteed to converge if the system satisfies the conditions for on-line diagnosability.

Sampath et al. [113, 114] propose an approach to failure diagnosis where the system is modeled as a DES in which the failures are treated as unobservable events. The level of detail in a discrete event model appears to be quite adequate for a large class of systems and for a wide variety of failures to be diagnosed. The approach is applicable whenever failures cause a distinct change in the system status but do not necessarily bring the system to a halt. Sampath *et al.* [113] provide a definition of diagnosability in the framework of formal languages and present necessary and sufficient conditions for diagnosability of systems. Moreover a systematic approach to solve the problem of diagnosis using diagnosers is introduced.

In a related work Sampath et al. [112] present an integrated approach to control and diagnosis. More specifically, authors present an approach for the design of diagnosable

systems by appropriate design of the system controller and this approach is called *active diagnosis*. They formulate the active diagnosis problem as a supervisory control problem. The adopted procedure for solving the active diagnosis problem is the following: given the non-diagnosable language generated by the system of interest, they first select an "appropriate" sublanguage of this language as the legal language. Choice of the legal language is a design issue and typically depends on considerations such as acceptable system behavior (which ensures that the system behavior is not restricted more than necessary in order to eventually make it diagnosable) and detection delay for the failures. Once the appropriate legal language is chosen, they then design a controller (diagnostic controller), that achieves a closed-loop language that is within the legal language and is diagnosable. This controller is designed based on the formal framework and the synthesis techniques that supervisory control theory provides, with the additional constraint of diagnosability.

Debouk *et al.* [33] deal with the problem of failure diagnosis in DES with decentralized information. In particular, they propose a coordinated decentralized architecture consisting of two local sites communicating with a coordinator that is responsible for diagnosing the failures occurring in the system. They extend the notion of diagnosability, originally introduced in Sampath *et al.* [113] for centralized systems, to the proposed coordinated decentralized architecture. In particular, they specify three protocols that realize the proposed architecture and analyze the diagnostic properties of these protocols.

Boel and van Shuppen [12] address the problem of synthesizing communication protocols and failure diagnosis algorithms for decentralized failure diagnosis of DES with costly communication between diagnosers. The costs on the communication channels may be described in terms of bits and complexity. The costs of communication and computation force the trade-off between the control objective of failure diagnosis and that of minimization of the costs of communication and computation. The results of this paper is an algorithm for decentralized failure diagnosis of DES for the special case of only two diagnosers.

Zad *et al.* [137] present a state-based approach for on-line passive fault diagnosis. In this framework, the system and the diagnoser (the fault detection system) do not have to be initialized at the same time. Furthermore, no information about the state or even the condition (failure status) of the system before the initiation of diagnosis is required. The design of the fault detection system, in the worst case, has exponential complexity. A model reduction scheme with polynomial time complexity is introduced to reduce the computational complexity of the design. Diagnosability of failures is studied, and necessary and sufficient conditions for failure diagnosability are derived.

2.3.2 Diagnosis with labeled Petri nets

Among the first pioneer works dealing with PNs, let us recall the approach of Prock [101]. He proposes an on-line technique for fault detection that is based on monitoring the number of tokens residing into P-invariants: when the number of tokens inside P-invariants changes, then the error is detected.

Sreenivas and Jafari [119] employ time PNs to model the DES controller and backfiring transitions to determine whether a given state is invalid. Later on, time PNs have been employed by Ghazel *et al.* [52] that propose a monitoring approach for DES with unobservable events and to represent the "a priori" known behavior of the system, and track on-line its state to identify the events that occur.

Hadjicostis and Veghese [59] use PN models to introduce redundancy into the system and additional P-invariants allow the detection and isolation of faulty markings.

Wu and Hadjicostis [132] use redundancy into a given PN to enable fault detection and identification using algebraic decoding techniques. In this paper the authors consider two types of faults: place faults that corrupt the net marking, and transition faults that cause a not correct update of the marking after event occurrence. Although this approach is general, the net marking has to be periodically observable even if unobservable events occur. Analogously, Lefebvre and Delherm [77] investigate on the determination of the set of places that must be observed for the exact and immediate estimation of faults occurrence.

Miyagi and Riascos [90] introduce a methodology, based on the hierarchical and modular integration of PNs, for modeling and analyzing fault-tolerant manufacturing systems that not only optimizes normal productive processes, but also performs detection and treatment of faults.

Ramirez-Treviño [105] employ Interpreted PNs to model the system behavior that includes both events and states partially observable. Based on the Interpreted PN model derived from an on-line methodology, a scheme utilizing a solution of a programming problem is proposed to solve the problem of diagnosis.

Note that all papers in this topic assume that faults are modeled by unobservable transitions. However, while the above mentioned papers assume that the marking of certain places may be observed, a series of papers have been recently presented that are based on the assumption that no place is observable [4, 9, 39, 51].

In particular, Genc and Lafortune [51] propose a diagnoser on the basis of a modular approach that performs the diagnosis of faults in each module. Subsequently, the diagnosers recover the monolithic diagnosis information obtained when all the modules are combined into a single module that preserves the behavior of the underlying modular system. A communication system connects the different modules and updates the diagnosis information. Even if the approach does not avoid the state explosion problem, an improvement is obtained when the system can be modeled as a collection of PN modules coupled through common places.

The main advantage of the approaches of Genc and Lafortune [51] consists in the fact that, if the net is bounded, the diagnoser may be constructed off-line, thus moving off-line the most burdensome part of the procedure. Nevertheless, a characterization of the set of markings consistent with the actual observation is needed. Thus, large memory may be required.

An improvement in this respect has been given in [9, 4, 39].

In particular, Benveniste *et al.* [9] use a net unfolding approach for designing an on-line asynchronous diagnoser. The state explosion is avoided but the on-line computation can be high due to the on-line building of the PN structures by means of the unfolding.

Basile *et al.* [4] build the diagnoser on-line by defining and solving Integer Linear Programming (ILP) problems. Assuming that the fault transitions are not observable, the net marking is computed by the state equation and, if the marking has negative components, an unobservable sequence is occurred. The linear programming solution provides the sequence and detects the fault occurrences. Moreover, an off-line analysis of the PN structure reduces the computational complexity of the ILP problem.

Dotoli *et al.* [39] propose a diagnoser that works on-line in order to avoid the redesign and the redefinition of the diagnoser when the structure of the system changes. In particular, the diagnoser waits for an observable event and an algorithm decides whether the system behavior is normal or may exhibit some possible faults. To this aim, some ILP problems are defined and provide eventually the minimal sequences of unobservable transitions containing the faults that may have occurred. The proposed approach is a general technique since no assumption is imposed on the reachable state set that can be unlimited, and only few properties must be fulfilled by the structure of the PN modeling the system fault behavior. A problem strictly related to diagnosis has been recently studied by Dotoli *et al.* [38]. They address the problem of identifying the model of the unobservable behavior of PN systems in the industrial automation framework. Assuming that the fault-free system structure and dynamics are known, the paper proposes an algorithm that monitors the system on-line, storing the occurred observable event sequence and the corresponding reached states.

A series of contributions dealing with diagnosis of PNs [18, 71, 17] have also been proposed. In particular, in [18, 71] *free-labeled* PNs are considered, while in [17], as well as in this thesis, the focus is on *labeled* PNs.

Some authors of this paper have also addressed the problem of diagnosability, namely the problem of providing a procedure to verify if it is possible to reconstruct the occurrence of fault events observing words of finite length. In particular, two different approaches for bounded [16] and unbounded [14] PNs have been proposed.

Very few other results deal with diagnosability within the framework of PNs.

The first contribution was given by Ushio *et al.* [126] that extend a necessary and sufficient condition for diagnosability given in [113, 114] to unbounded PN. They assume that the set of places is partitioned into observable and unobservable places, while all transitions are unobservable in the sense that their occurrences cannot be observed. Starting from the PN they build a diagnoser called *simple ω diagnoser* that gives them sufficient conditions for diagnosability of unbounded PNs.

Chung [27], in contrast with Ushio's paper, assumes that part of the transitions of the PN modelling is observable and shows as the additional information from observed transitions in general adds diagnosability to the analysed system. Moreover starting from the diagnoser he proposes an automaton called *verifier* that allows a polynomial check mechanism on diagnosability but for finite state automata models.

Finally, Wen and Jeng [130] propose an approach to test diagnosability by checking the structure property of T-invariants of the nets. They use Ushio's diagnoser to prove that their method is correct, however they don't construct a diagnoser for the system to do diagnosis. Moreover Wen *et al.* [131] also present an algorithm, based on a linear programming problem, of polynomial complexity in the number of nodes for computing a sufficient condition of diagnosability of DES modeled by PNs.

Chapter 3

Synchronizing sequences on bounded Petri nets

Summary

This chapter presents new results on the computation of synchronizing sequences for systems modeled by synchronized Petri nets.

First we show how the classic automaton approach [76] can be easily adapted to systems represented by the class of bounded synchronized Petri nets. We refer to this approach as the RG approach.

Then it is shown that the same problem can be efficiently solved using Petri nets by taking into account the net structure, without an exhaustive enumeration of its state space. The approach proposed (*cf.* STS) for computing synchronizing sequences exploits the net structure and leads to viable algorithms that can be applied to large scale systems.

Finally, a different third approach (*cf.* MRG) combines the structural criteria of the approach STS and the efficacy of the approach RG.

Computational complexity analysis of RG, STS and MRG approaches are provided and comparisons carried out by using different test-benches.

3.1 An adaptation of the automata technique

When computing a SS for real systems modeled by automata, it is assumed that it is given a complete description of the model in terms of space-set, input events and transition function. The idea is that the experimenter knows all possible states in which the system may be.

A similar notion can be given for PNs, where clearly the information on the net structure is not sufficient but it is also necessary to univocally determine the the entire state-space. In fact, as shown in Section 2.1.2, generally the reachability set of synchronized PN depends on the initial marking.

Given a synchronized PN $\langle N, E, f \rangle$, a straightforward approach to determine a SS consists in adapting the existing approach for automata to the reachability graph (RG) of the PN. In the rest of this thesis, this approach will be called RG approach.

This setting is further extended to unbounded PNs, i.e., nets whose reachability set is infinite.

3.1.1 A reachability graph approach for synchronized Petri nets

As previously seen,to construct the RG and characterize the space-set of the considered PN, it is necessary to know the initial marking together with the net structure. Equivalently, one can say that the experimenter knows a possible marking of the net, i.e., a marking previously reached by the net and then construct the space-set starting from this marking. We call this possible marking the *starting marking*.

The synchronization problem via PNs can be so reformulated: it is given a PN $N = (P, T, Pre, Post)$ and a starting marking M_0. The current marking M is unknown, but it is assumed to be reachable from M_0, i.e., $M \in R(N, M_0)$.

This starting marking, together with the firing rules, provides a characterization of the initial state uncertainty, that coincides with the set of markings reachable from it, i.e., $\mathcal{M}_0 = R(N, M_0)$.

The goal is to find an input sequence that, regardless of the current marking, drives the net to a known target marking $\bar{M}$.

It is easy to verify that this direct adaptation presents one shortcoming that makes it not always applicable: the greedy approach, defined by Algorithm 2.32, requires the graph to be completely specified, while in a RG of a PN this condition is not always true. In fact, from a marking not all transitions are necessarily enabled, causing the RG of the PN to be partially specified.

Example 3.1. *Consider the PN in Figure 2.4a. Marking* $M = [2\ 0\ 0]^T$ *enables only transition* t_1 *thus all the events not associated with* t_1 *are not specified, i.e., not depicted*

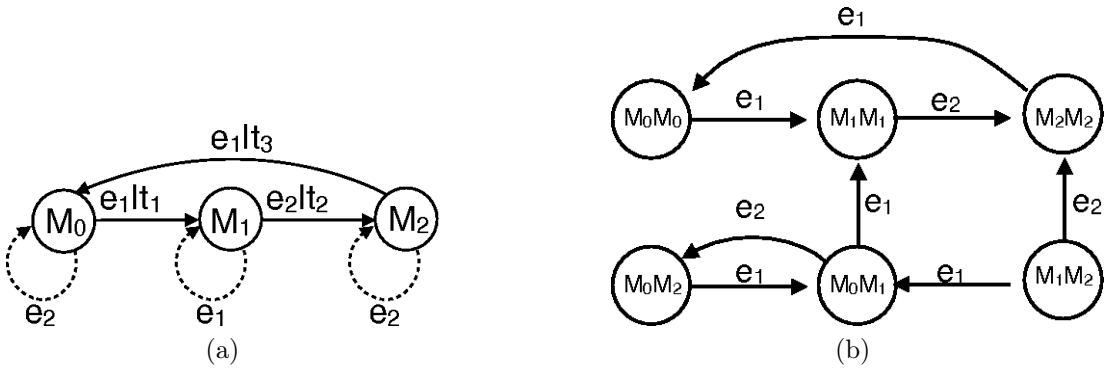


Figure 3.1: The completely specified RG of the PN in Figure 2.4a (a) and its corresponding auxiliary graph (b).

in the RG for this node. Hence for that marking one adds a self loop labelled e_2 and so on for the rest of the reachable markings. ■

In order to use the aforementioned approach it is necessary to turn a RG $\mathcal{G}$ into a completely specified graph $\tilde{\mathcal{G}}$. This is obtained by applying the following algorithm.

Algorithm 3.2. ***(Obtaining a completely specified RG for a synchronized PN)***
Input: *a RG $\mathcal{G}$ of a marked synchronized PN $\langle N, M_0, E, f\rangle$.*
Output: *a completely specified RG $\tilde{\mathcal{G}}$.*

1. ***While*** *there exists a non-tagged node q in $\mathcal{G}$,* ***do***

  1.1. *pick node q and tag it;*

  1.2. *let M be the label associated with q.*

  1.3. ***For*** *all $e \in E$, do*

    1.3.1. *let $\mathcal{E}_e(M)$ be the set of transitions associated with event e and enabled at marking M.*

    1.3.2. ***if*** *$\mathcal{E}_e(M) = \varnothing$, add a self-loop to q and label it with event e.*

  End for

End while

2. *Remove all tags.* ■

Note that the RG and the obtained completely specified RG are equivalent in behavior and both correspond to the given marked PN. In fact, all the self-loops added by the above algorithm at step 1.3.2 correspond to those events that do not produce any firing of transition. These events do not alter the marking of the net, hence neither the state of the corresponding automaton.

In Figure 3.1a is shown the RG of the PN in Figure 2.4a and dashed edges are added in order to make it completely specified. These arcs are labelled by the only non- specified

event because they are associated with no firing transition. In Figure 3.1b is shown its corresponding auxiliary graph.

The modified approach for PNs can now be summarized by the following algorithm.

Algorithm 3.3. ***(RG computation of a SS on synchronized PNs)***
Input: *A bounded synchronized PN* $\langle N, E, f\rangle$*, a starting marking* M_0 *and a target marking* $\bar{M}$*.*
Ouput: *A SS* $\bar{w}$ *for marking* $\bar{M} \in R(N, M_0)$*.*

1. *Let* $\mathcal{G}$ *be the RG of* $\langle N, M_0\rangle$*.*
2. *Let* $\tilde{\mathcal{G}}$ *be the completely specified RG obtained by applying Algorithm 3.2 to* $\mathcal{G}$*.*
3. *Construct the corresponding auxiliary graph* $\mathcal{A}(\tilde{\mathcal{G}})$*.*
4. *A SS for marking* $\bar{M}$*, if such a sequence exists, is given by the direct application of Algorithm 2.32 to* $\mathcal{A}(\tilde{\mathcal{G}})$*, having* $\bar{M}$ *as target.* ■

The following proposition can now be stated.

Proposition 3.4. *Given a deterministic bounded synchronized PN* N *and a starting marking* M_0*, there exists a SS leading to a marking* $\bar{M} \in R(N, M_0)$ *iff the reachability condition on its auxiliary graph* $\mathcal{A}(\tilde{\mathcal{G}})$ *is verified, i.e., there is a path from every node* (M_i, M_j)*, with* $M_i, M_j \in R(N, M_0)$*, to node* $(\bar{M}, \bar{M})$*.*

Proof. : Consider a marked PN net $\langle N, M_0\rangle$ and its RG $\mathcal{G}$. Given a marking $M \in R(N, M_0)$, a sequence $\sigma = t_{j1}t_{j2}\dots t_{jp}$ generates the trajectory $M[t_{j1}\rangle M_1[t_{j2}\dots t_{jp}\rangle M_p$ iff there exists an oriented path $\gamma = Mt_{j1}M_1t_{j2}\dots t_{jp}M_p$ in $\mathcal{G}$. The same equivalence holds between a synchronized PN and its completely specified RG $\tilde{\mathcal{G}}$. Thus an input sequence $w = e_{j1}e_{j2}\dots e_{jp}$ drives the net from M to M_p iff there exists an oriented path $\gamma = Me_{j1}M_1e_{j2}\dots e_{jp}M_p$ in $\tilde{\mathcal{G}}$.

Since the completely specified RG $\tilde{\mathcal{G}}$ can be considered as an automaton whose behavior is equivalent to that of the synchronized PN itself, Theorem 2.34 easily applies. □

3.1.2 Complexity via reachability graph analysis

As previously said in Section 2.2.2, the *greedy* computation of a SS works in $O(n_\chi^3 + |n_i|n_\chi^2)$ time, where n_χ and $|n_i|$ are, resp., the number of states and the input alphabet cardinality of the given automaton. Such an algorithm is applicable to synchronized PNs by first exhaustively enumerating the state space of the net, i.e., constructing its RG. Although alternative techniques are proposed to decrease its complexity (e.g. [13, 44]), the RG generation suffers from the problem of exponential space and time complexity. In particular, for a SM PN the reachability set of markings significantly increases with the number of tokens, as discussed in the following proposition.

Proposition 3.5. *Given a strongly connected SM PN $N = (P, T, Pre, Post)$, with k tokens, let m be the number of its places. The RG $\mathcal{G}$ of this net has a number of nodes equal to:*

$$\binom{m+k-1}{m-1} \leq \frac{1}{(m-1)!}k^{m-1}$$

Proof. Consider the given net once a new place is added. It can be easily shown that the reachability set cardinality is given by the following formula:

$$|\mathcal{G}(m+1,k)| = \sum_{i=0}^{k} |\mathcal{G}(m,i)| = |\mathcal{G}(m,0)| + |\mathcal{G}(m,1)| + \ldots + |\mathcal{G}(m,k)|,$$

where $|\mathcal{G}(m,i)|$ is the cardinality of the reachability graph of the obtained PN with $k-i$ tokens in the added place, that is also equal to the cardinality of reachability graph of the initial PN with i tokens. Such a result can be reported in a matrix form, obtaining the well known Pascal matrix, that comes out from the Pascal's triangle. The elements of the symmetric Pascal matrix are the binomial coefficients, i.e., it holds that

$$\binom{i+j-2}{i-1}$$

having $i = m$, $j = k+1$. □

Considering the above result, one can state the following lemma.

Lemma 3.6. *Consider a strongly connected SM PN with k tokens. Let m be the number of its places and E be its input alphabet. For such a net, Algorithm 3.3 requires a time*

$$O(|\mathcal{G}(m,k)|^3 + |E||\mathcal{G}(m,k)|^2) \leq O\left(\left[\frac{k^{m-1}}{(m-1)!}\right]^3 + |E|\left[\frac{k^{m-1}}{(m-1)!}\right]^2\right).$$ ■

3.2 Synchronizing sequences on synchronized strongly connected state machine Petri nets

Consider a strongly connected SM PN defined in Section 2.1.2. Knowing the number of tokens k initially contained in the net — regardless of their initial distribution — is sufficient to exactly determine the reachability set of the net: in fact, the number of tokens will remain constant as the net evolves and any distribution of the k tokens can be reached.

If a SM PN is not strongly connected, knowing the number of tokens k initially contained in the net — but not their initial distribution — will give a larger approximation of the reachability set that may be used to design a SS. The knowledge of the number of tokens initially contained in each component — but not their initial distribution within each component — will provide an exact characterization of the reachability set.

This new setting aims to determine a SS without constructing the whole state-space. Hence a new formal definition of SS for SM PNs has to be given.

Definition 3.7. ***(SS on state machine PNs)*** *Given a synchronized SM PN $\langle N, E, f\rangle$, assume that the initial marking M_0 is not given but is known to belong to a set*

$$\mathcal{M}_0 = \{M \in \mathbb{N}^m \mid \sum_i M(p_i) = k\}.$$

Sequence $\bar{w}$ is called a k-SS if for all $M \in \mathcal{M}_0$ it holds $M \xrightarrow{\bar{w}} \bar{M}$. ■

In this section, two different techniques to solve the synchronization problem on the framework of strongly connected state machine PNs are proposed.

The first approach exploits the net structure and obtains a SS, by constructing paths that satisfy certain labeling conditions. Such paths are called *synchronizing transition sequences* (STS).

The second approach is based on a greedy computation made on a *modified reachability graph* (MRG), which is obtained by pruning the RG of the net.

3.2.1 A synchronizing transition sequence approach

In this subsection we present a particular technique to determine SSs via sufficient conditions over the net structure.

Such a technique can be more efficient than the approach presented in Algorithm 3.3, as discussed later in Section 3.1.2, Section 3.2.2 and Section 3.2.4.

The problem of determining a 1-SS is here first analyzed and then the more general k-SS construction, starting from a 1-SS, is addressed.

3.2.1.1 1-SS on strongly connected state machines

1-SS are first taken into account.

The notion of *synchronizing transition sequence* for a set of places $\hat{P}$ and a specific place $\bar{p} \in \hat{P}$ is given on the basis of Definition 2.7 of directed path.

Definition 3.8. ***(Synchronizing transition sequence)*** *Given a synchronized SM PN $\langle N, E, f\rangle$, let $\rho(\hat{P}, \bar{p}) = \langle p'_0 t'_1 p'_1 t'_2 \cdots t'_r p'_r\rangle$, with $\bar{p} = p'_r$, be a directed path in $N = (P, T, Pre, Post)$ that visits all places in $\hat{P} \subseteq P$ and ends in $\bar{p} \in \hat{P}$, with $\bar{p} \neq p_j$ for $j = 0, 1, \ldots, r-1$. Let σ be the firing sequence obtained by removing all places from $\rho(\hat{P}, \bar{p})$. Such a sequence is called a* synchronizing transition sequence *for $\hat{P}$ and $\bar{p}$ if*

C1) $\nexists t, t' \in \hat{P}^\bullet$ *such that* $t \in \sigma$, $t' \notin \sigma$, *and* $f(t) = f(t')$.

C2) $\forall p'_i, p'_k \in \rho(\hat{P}, \bar{p})$: *if* $p'_i = p'_k$ *and* $i < k$ *it holds that* $f(t'_j) \neq f(t'_k)$ *for* $j = 1, \ldots k-1$. ■

In simple words, condition C1) requires that there is no transition exiting $\hat{P}$ and sharing the same label of a transition in σ. Condition C2) requires that if a place is visited multiple

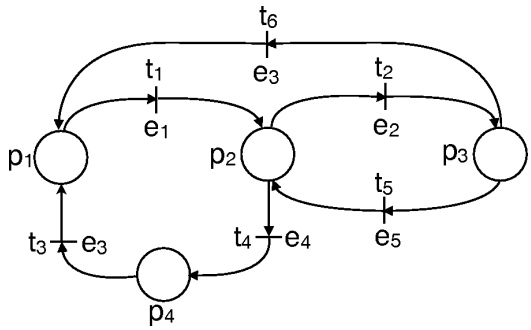


Figure 3.2: A strongly connected synchronized SM PN

times, its ingoing transition has not to share the same label of any of the transitions in the path.

A first result related to the existence of a SS for SM PNs with a single token can now be stated.

Proposition 3.9. *Consider a strongly connected synchronized SM PN* $\langle N, E, f\rangle$ *containing a single token. Let* σ *be a synchronizing transition sequence for* P *and* $\bar{p} \in P$*. Then* $\bar{w} = f^*(\sigma)$ *is a* 1*-SS for marking* $\bar{M}$ *that assigns the token to place* $\bar{p}$*, i.e.,*

$$\bar{M}:\ \bar{M}(p) \ = \ \begin{cases} 1 \ \ if \ \ p = \bar{p}, \\ 0 \ \ otherwise. \end{cases}$$

Proof. Let $\sigma = t'_1 \cdots t'_r$ be the synchronizing transition sequence found and $\rho(P, \bar{p}) = p'_0 t'_1 p'_1 t'_2 \cdots t'_r p'_r$, with $\bar{p} = p'_r$, be the corresponding path (not necessarily elementary).

Let $\bar{w}$ be the corresponding input event sequence, i.e., $\bar{w} = f^*(\sigma) = e'_1 \cdots e'_r$.

We first prove that after the occurrence of event e_1 the token can only be in a place p'_k such that $k \geq 1$. Assume, in fact, the token is initially in place p'_i and event e'_1 occurs. Two different cases have to be treated. If $i = 0$, then by definition of σ, the token is certainly driven to place p'_1. If $i \geq 1$, two further subcases are possible: a) no output transition of p'_i has label e_1, i.e., $T_{e_1} \cap p'^{\bullet}_i = \varnothing$, and the token will not not move; b) an output transition of p'_i has label e_1 and its firing moves the token to some place p'_j, with $j > i$.

The last result follows from conditions C1) and C2) of Definition 3.8. In fact, condition C1) assures that only transitions belonging to σ are receptive to e_1; thus the token can only be driven along the chosen path. Besides, condition C2) assures that the token cannot go back in the upstream path along the sequence.

By repeating this argument, we can show that after the application of event e_i, for $i = 2, \ldots, r$, the token can only be in a place p'_k such that $k \geq i$, hence this ensures that when all events in the input sequence $\bar{w}$ have been applied the token will be in place $\bar{p}$. □

Next algorithm shows how Proposition 3.9 can be effectively used to compute a 1-SS. In this, function $\tau : P \times T \to T$ (resp. $\pi : P \times T \to P$) returns the set of transitions (resp. places) visited by directed path ρ. Function $start : P \times T \to P$ determines the last place

which has been added to ρ. For instance, consider the path $\rho = \langle p_r t_r p_{r-1} t_{r-1} \cdots t_1 p_0 \rangle$. It holds that $\tau(\rho) = \{t_1, t_2, \cdots, t_r\}$, $\pi(\rho) = \{p_0, p_1, \ldots, p_r\}$ and $start(\rho) = p_r$.

Algorithm 3.10. ***(STS computation of a 1-SS on synchronized SM PNs)***
Input: *A synchronized SM PN $\langle N, E, f \rangle$ and a target place $\bar{p}$.*
Ouput: *a 1-SS w that drives the token to place $\bar{p}$.*

1. $\rho = \bar{p}$, $\mathcal{R} = \{\rho\}$;

2. $flag := false$;

3. ***while*** $flag = false \vee \mathcal{R} \neq \varnothing$

a. *pick* $\rho \in \mathcal{R} : |\rho| = \max_{\rho' \in \mathcal{R}} |\rho'|$;

b. $p := start(\rho)$;

c. $\mathcal{T} := {}^\bullet p \backslash (\tau(\rho) \cup \bar{p}^\bullet)$;

d. ***while*** $flag = false \vee \mathcal{T} \neq \varnothing$,

i. *pick* $t \in \mathcal{T}$, $\rho' := {}^\bullet t t \rho$;

ii. ***if*** *ρ' does not satisfiy C2),* ***then goto*** *step* ***3.d.v.***

iii. ***if*** $\pi(\rho) = P$,

– ***if*** *ρ' satisfies C1),* ***then*** $flag := true$.

else, ***goto*** *step* ***3.d.v.***

iv. $\mathcal{R} := \mathcal{R} \cup \{\rho'\}$

v. $\mathcal{T} := \mathcal{T} \backslash \{t\}$;

e. $\mathcal{R} := \mathcal{R} \backslash \{\rho\}$.

end while

end while

4. ***if*** $flag = false$,

a ***then*** *no STS exists;*

else

b *pick* $\rho \in \mathcal{R} : |\rho| = \max_{\rho' \in \mathcal{R}} |\rho'|$;

c *let σ be the firing sequence obtained removing all places from ρ;*

d $w := f^*(\sigma)$.

end if ■

The algorithm computes a synchronizing transition sequence and the corresponding SS. It starts from desired place $\bar{p}$ (step 1) and puts the path of zero length $\rho = \bar{p}$ into $\mathcal{R}$, which contains the set of path to be analyzed. The net is explored using a backward search until either a STS has been found, i.e., the $flag$ is true, or there are no more paths to analyze, i.e., $\mathcal{R} \neq \varnothing$ (step 3).

Once a path ρ is selected, we consider the set of transitions $\mathcal{T}$ inputting its start place p that i) have not already been visited in the path; ii) do not output from the final place $\bar{p}$ (step 3.c).

For all new paths ρ', obtained adding to ρ one transition in $\mathcal{T}$ and its input place (step 3.d.i), we do the following. First, we check condition C2) (step 3.d.ii), which must hold for all prefixes of the final path. If it does not hold, we discard the path going to step 3.d.v. Then we check if ρ' contains all places: in this case, if it satisfies condition C1) (step 3.d.iii) we stop the algorithm (flag=true), else we discard it, going to step 3.d.v. All new not discarded paths, are added to set $\mathcal{R}$ to be later explored (step 3.d.iv).

When all transitions in $\mathcal{T}$ have been evaluated, path ρ is then removed from $\mathcal{R}$ (step 3.e).

At step 4, if the flag is set to false, there is no 1-SS constructible via the STS approach. Otherwise the path of maximum length is contained in $\mathcal{R}$ and it defines an STS.

Paths are constructed via a depth-first-search, as ensured by the condition of step 3.a that always picks (one of) the longest path(s). We could implement a *breadth-first-search* by picking — at the same step — the shortest $\rho \in \mathcal{R}$, to ensure the shortest STS solution to be found.

Example 3.11. *Consider the strongly connected SM PN in Figure 3.2. The objective is to find a SS that leads the system to the marking* $[0\,0\,0\,1]^T$. *Let* $\rho = p_1t_1p_2t_2p_3t_5p_2t_4p_4$ *be the directed path that contains all the places and ends in* p_4. *Let* $\sigma = t_1t_2t_5t_4$ *be the synchronizing transition sequence for* P *and* p_4. *It holds that* $\bar{w} = f^*(\sigma) = e_1e_2e_5e_4$ *is the searched* 1*-SS.* ■

Note that condition C1) of Definition 3.8 is sufficient to assure the sequence to be a synchronizing one if ρ is an elementary path.

The conditions given by Proposition 3.9 for the existence of a SS are sufficient but not necessary.

Although one determines a SS by just analyzing the net structure — avoiding then the RG and the auxiliary graph construction and consistently reducing the complexity —, the conditions required are very restrictive.

In fact there are SM PNs for which those conditions do not hold but that still have a SS.

Example 3.12. *Consider again the strongly connected SM PN in Figure 3.2 with one token and suppose* $f(t_5) = f(t_2) = e_2$. *Let the goal be again to construct a* 1*-SS that leads the system to the marking* $[0\,0\,0\,1]^T$. *There clearly exists no synchronizing transition sequence with such a change of the labeling function, hence no* 1*-SS can be determined by Proposition 3.9. Despite this, one easily finds the* 1*-SS* $\bar{w} = e_1e_4e_2e_4$ *by the way of Algorithm 3.3.* ■

In simply words, when conditions required by Proposition 3.9 do not hold, one can always determine a SS using Algorithm 3.3, obviously with an increased complexity as previously shown in section 3.1.2.

3.2.1.2 k-SS on strongly connected state machines

We now consider the problem of determining a k-SS for nets with k tokens.

Proposition 3.13. *Consider a strongly connected synchronized SM PN $\langle N, E, f\rangle$ containing k tokens.*

Let σ be a synchronizing transition sequence for P and $\bar{p} \in P$ and $w = f^*(\sigma)$ a 1−SS. w^k is a k-SS that moves all k tokens to place $\bar{p}$, such that:

$$\bar{M} : \bar{M}(p) = \begin{cases} k \ if \ p = \bar{p}, \\ 0 \ otherwise. \end{cases}$$

Proof. Consider a first application of w, at least one token is driven to $\bar{p}$. Because of condition C2) and of the fact that the directed path does not pass through $\bar{p}$, none of the output transitions of this place is receptive to some event in w. Hence every application of w does not move the token from $\bar{p}$ and takes the k tokens at least one by one to place $\bar{p}$. □

Example 3.14. *Consider the SM PN of Example 3.11, where $w = e_1e_2e_5e_4$ is the 1-SS previously found. Let the PN have 2 tokens. It holds that $w^2 = e_1e_2e_5e_4e_1e_2e_5e_4$ is a 2-SS, leading the net to the desired final marking $\bar{M} = [0\,0\,0\,2]^T$.* ■

The previous propositions shows that having determined a synchronizing transition sequence, one readily obtains not only a 1-SS but a k−SS for an arbitrary k. However not all SSs can be obtained in this way.

Thus we consider the following problem: given any arbitrary 1-SS, constructed not from a synchronizing transition sequence but by using Algorithm 3.3, does Proposition 3.13 apply so that we can we use it to construct a k-SS? Unfortunately this is not case, as shown by next example.

Example 3.15. *Consider the SM PN of Example 3.11 and let $w = e_3e_1e_2e_5e_4$ be a 1-SS for p_4. Let the PN have 2 tokens. It is easy to see that w^2 is not a 2-SS, since only one token out of two would be driven to p_4.* ■

It is however possible to provide a sufficient condition for an arbitrary 1−SS to ensure that, concatenating it k times, a k−SS is obtained.

Proposition 3.16. *Consider a strongly connected synchronized SM PN $\langle N, E, f\rangle$ containing k tokens. Let w be a 1-SS for a target marking $\bar{M}$, such that $\bar{M}(p) = 1$ if $p = \bar{p}$, otherwise $\bar{M}(p) = 0$.*

If for all $t \in \bar{p}^\bullet$ it holds that $f(t) \notin w$, i.e., sequence w does not contain any symbol labeling an output transition of place $\bar{p}$, then w^k is a k-SS for a target marking $\bar{M}_k$, such that $\bar{M}_k(p) = k$ if $p = \bar{p}$, otherwise $\bar{M}_k(p) = 0$.

Proof. During the first application of w, at least one of the tokens is driven to $\bar{p}$. Any further application of w moves to $\bar{p}$ at least one of the tokens not in this place, and does

not move the tokens already in $\bar{p}$, as none of its output transitions is receptive to a event in w. Thus w^k takes the k tokens to place $\bar{p}$. □

Note that any SS constructed via the STS approach satisfies this sufficient condition by definition.

3.2.2 Complexity via synchronizing transition sequences analysis

We have shown in Proposition 3.9 a technique to compute a 1-SS on a strongly connected net based on synchronizing transitions sequences. Here we discuss the complexity of such a procedure.

Algorithm 3.10 computes a synchronizing transition sequence by using a backward depth-first search from place $\bar{p}$ and verifying the conditions of Definition 3.8 over the labeling function.

It is known that a depth first search requires $O(b^d)$ time [29], for explicit graphs traversed with repetition, having a branching factor b and a depth search of d.

Assume that a SM PN has a backward branching factor (the number of transitions inputting in a place) bounded by $\phi = \max_{p \in P} |{}^\bullet p|$. While exploring the net with possible repetitions of places, an upper bound for the depth search length is $q - 1$, where q is the number of net transitions. Thus a first very rough approximation of the needed time is given by $O(\phi^{q-1})$.

This time only depends on structural net parameters, does not grow with the number of tokens and is typically smaller than the time required by Algorithm 3.3.

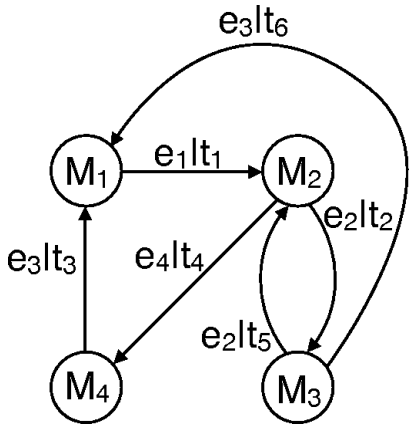
3.2.3 A modified reachability graph approach

As previously proven, under some particular conditions any SS for a net with one token can also be used as a building block to determine a SS for k tokens.

The STS approach constructs SSs with a depth-first search on the net structure and verifies certain conditions over the labeling function. However its conditions may prevent the approach to find a solution, that is why here a new approach is presented.

This different approach is provided so that the sole sequences satisfying these conditions, if any exists, are found.

The proposed approach is a modified implementation of the RG approach, based on an arc-pruning. Given a target place $\bar{p}$, events labeling arcs outputting $\bar{p}$ — disregarding self-loops — are not considered. These events belong to the set of *forbidden events*, which is denoted as $\mathcal{F}(\bar{p})$.



M_1	$[1\,0\,0\,0]^T$
M_2	$[0\,1\,0\,0]^T$
M_3	$[0\,0\,1\,0]^T$
M_4	$[0\,0\,0\,1]^T$

Figure 3.3: The RG (b) of the PN in Figure 3.2 with one token and $f(t_5) = e_2$.

On the basis of the given target place, we construct the so-called *modified reachability graph* (MRG) $\mathcal{G}_M$, which is the graph of the net with only 1 token obtained by removing all forbidden events from the RG $\mathcal{G}$ of the net. Note that there is a different MRG for any given target place.

Example 3.17. *Consider the synchronized PN of Figure 3.2 with $f(t_5) = e_2$ and containing one token. its RG is shown in Figure 3.3. For $\bar{p} = p_4$, it holds that $\mathcal{F}(p_4) = \{e_3\}$. Its MRG $\tilde{\mathcal{G}}_M$ is shown in Figure 3.4a disregarding dashed edges. Here, $M_i : M_i(p) = 1$ if $p = p_i$, 0 otherwise. The dashed self-loops are then added to make it completely specified. The corresponding auxiliary graph $\mathcal{A}(\tilde{\mathcal{G}}_M)$ is shown in Figure 3.4b.* ■

The MRG computation of 1-SS consists of three steps. First, we determine the set of forbidden events, which is used to construct the MRG and the corresponding auxiliary graph; second, we apply the algorithm for the RG computation. Finally we prove that the obtained sequence is a 1-SS for the complete model.

The so-constructed SS is 1-SS $\bar{w}$, which is not necessarily the shortest one. Such a sequence leads the net to a target marking $\bar{M} \in R(N, M_0)$, where place $\bar{p}$ is the only marked place.

Algorithm 3.18. ***(MRG computation of a 1-SS on synchronized SM PNs)***
Input: *a synchronized SM PN $\langle N, E, f\rangle$ and a target place $\bar{p}$.*
Ouput: *a 1-SS $\bar{w}$ for a marking $\bar{M}$ such that $\bar{M}(p) = 1$ if $p = \bar{p}$ and 0 otherwise.*

1. *Let $\mathcal{F}(\bar{p}) \subseteq E$ be the set of forbidden events such that $\mathcal{F}(\bar{p}) = \{e \in E : \exists t \in \bar{p}^\bullet \setminus {}^\bullet\bar{p} \wedge f(t) = e\}$.*
2. *Construct the RG $\mathcal{G}$ of the net for any initial marking $M_0 : M_0^T \cdot \vec{1} = 1$.*

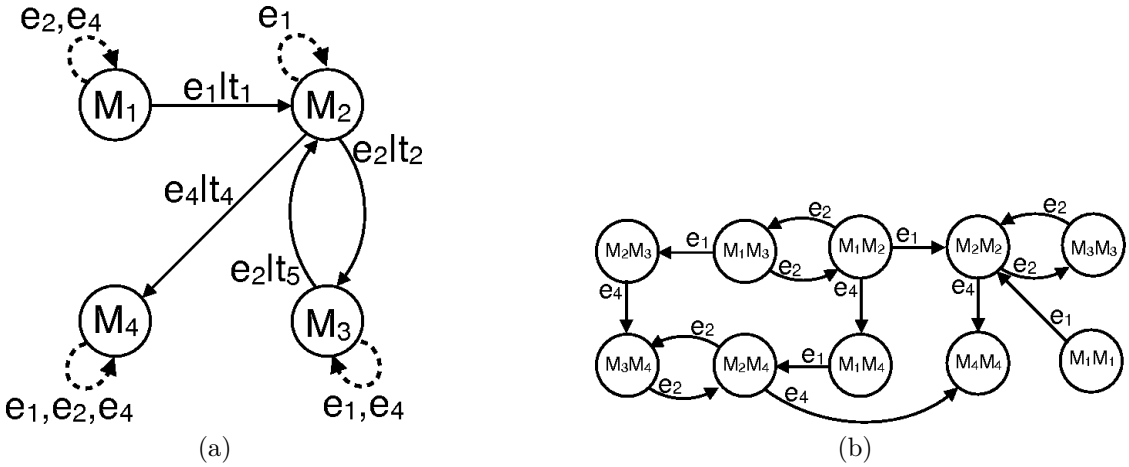


Figure 3.4: The completely specified MRG $\tilde{\mathcal{G}}_M$ for $\bar{p} = p_4$ (a) and its auxiliary graph $\mathcal{A}(\tilde{\mathcal{G}}_M)$ (b) of the PN in Figure 3.2 with one token and $f(t_5) = e_2$.

3. *Construct the MRG $\mathcal{G}_M$, by removing all events $e \in \mathcal{F}(\bar{p})$ from $\mathcal{G}$.*
4. *Let $\tilde{\mathcal{G}}_M$ be the completely specified MRG, obtained by applying Algorithm 3.2 to $\mathcal{G}_M$.*
5. *Construct the corresponding auxiliary graph $\mathcal{A}(\tilde{\mathcal{G}}_M)$.*
6. *A SS for marking $\bar{M}$, if such a sequence exists, is given by the direct application of Algorithm 2.32 to $\mathcal{A}(\tilde{\mathcal{G}}_M)$, having $\bar{M}$ as target.* ■

Note that Algorithm 3.3 for the RG computation of a SS is here applied to pruned graphs. The correctness of its results is proven by the following theorem.

Theorem 3.19. *Sequences determined by the way of Algorithm 3.18 are* 1*-SS for the considered synchronized SM PN.*

Proof. Let $\bar{w} = e_1 e_2 \cdots e_k$ be the sequence constructed by Algorithm 3.18, such that it drives the MRG through the sequence of markings M'_0, M'_1, M'_2, $\cdots$, M'_k. We have to prove that: i)$\bar{w}$ drives the given synchronized PN through exactly the same sequence of markings; ii) point i) holds for every marking $M'_0 \in \mathcal{M}_0$.

First, we prove point i). For any event $e_i \in \bar{w}$ every transition $t \in T_{e_i}$ fires in the net and is taken into account in $\mathcal{G}_M$. Hence, if in the latter the occurrence of event e_i at M'_i yields M'_{i+1}, the same marking will be reached on the PN.

Second, we prove point ii), by showing that at step 6, while applying Algorithm 2.32 to the pruned MRG, the current state uncertainty corresponds to the set of reachable markings of the net, which is necessary by Definition 3.7 of 1-SS. This proof is easily provided, since $\mathcal{G}_M$ has the same cardinality of $\mathcal{G}$, whose behavior is equivalent to the synchronized net itself. □

Example 3.20. *Consider the synchronized PN of Figure 3.2. The objective is to find a* 1*-SS such that the the single token is driven to place p_4. As previously shown in Exam-*

ple 3.12, no 1*-SS can be determined by Proposition 3.9, i.e., the STS approach does not find any solution.*

However, the execution of Algorithm 3.18 may give the 1*-SS* $\bar{w} = e_1e_4e_2e_4$*, as pointed out in the aforementioned example.* ■

As previously said the RG approach needs to construct the RG $\mathcal{G}$ of the synchronized PN and then its corresponding auxiliary graph $\mathcal{A}(\mathcal{G})$. Their cardinalities, as shown in Section 3.1.2, may significantly increase with an increasing number of tokens.

The MRG approach constructs the RG $\mathcal{G}$ of the net for $k = 1$ and, for a given place $\bar{p}$, the MRG $\mathcal{G}_M$ and its corresponding auxiliary graph $\mathcal{A}(\mathcal{G}_M)$, whose cardinality is exactly the same of $\mathcal{A}(\mathcal{G})$ but with a smaller branching factor. However, for a different number of tokens, none of the two graphs has to be reconstructed, since determining a 1-SS using Algorithm 3.18 allows to readily determine a k-SS for an arbitrary large k. That is because for such sequences the sufficient condition provided by the following proposition holds.

Proposition 3.21. *Consider a strongly connected synchronized SM PN* $\langle N, E, f\rangle$ *containing* k *tokens. Let* $\bar{w}$ *be a* 1*-SS constructed by the way of Algorithm 3.18 for a target marking* $\bar{M}$*, such that* $\bar{M}(p) = 1$ *if* $p = \bar{p}$*, otherwise* $\bar{M}(p) = 0$*. The input sequence* $\bar{w}^k$ *is a* k*-SS for a target marking* $\bar{M}_k$*, such that* $\bar{M}_k(p) = k$ *if* $p = \bar{p}$*, otherwise* $\bar{M}_k(p) = 0$*.*

Proof. Every sequence $\bar{w}$ constructed by Algorithm 3.18 satisfies the following condition: $(\forall t \in \bar{p}^\bullet \backslash {}^\bullet\bar{p})\ f(t) \notin w$. In other words sequence $\bar{w}$ does not contain any symbol labeling a transition that outputs place $\bar{p}$ and inputs any place $p \neq \bar{p}$. Proposition 3.16 provides a slightly more restrictive version of this proposition. The requirement is that no transition $t \in \bar{p}^\bullet$ is labelled by an event $e \in \bar{w}$. Instead in this case here we disregard transition labeling self-loops. However, the same proof provided for Proposition 3.16 applies also in this case. □

3.2.4 Complexity via a modified reachability graph analysis

The MRG approach is basically a modified version of the RG approach. In fact both the RG and MRG approach perform the same greedy algorithm on slightly different graphs. The MRG approach introduces the set of forbidden events to decrease the branching factor of the analyzed reachability graph. Given a target place $\bar{p}$ and its corresponding set of forbidden events $\mathcal{F}(\bar{p})$.

The computational time does not change with the number of tokens k, as previously shown in Section 3.2.3, since any k-SS is constructed starting from the SS for the 1-token case. The analyzed graph is of size

$$|\mathcal{G}(m,k)|_{k=1} = m.$$

The following lemma can now be stated.

Lemma 3.22. *Consider a strongly connected SM PN with* k *tokens. Let* m *be the number of its places and* E *be its input alphabet. Let* $\bar{p}$ *be the target place and* $\mathcal{F}(\bar{p})$ *the*

corresponding set of forbidden events. For such a net, Algorithm 3.18 requires a time:

$$\begin{cases} O(m^3 + |E'|m^2) \\ E' = E \backslash \mathcal{F}(\bar{p}) \end{cases}$$

■

3.3 Non strongly connected state machines Petri nets

Consider now connected — but not necessarily strongly connected — state machines. It can be shown how the existence of SS depends on the interconnection between ergodic and transient components.

Proposition 3.23. *Consider a synchronized SM PN $\langle N, E, f \rangle$ with μ transient components and η ergodic components. If $\eta > 1$ there exists no SS for such a net.*

Proof. Let the net have two ergodic components ER' and ER''. Consider two initial markings M_0' and M_0'' both with k tokens such that M_0' (resp., M_0'') assigns all tokens to the component ER' (resp., ER''). Clearly there exists no marking $\bar{M}$ reachable from both M_0' and M_0'', hence no SS exists according to Definition 3.7. □

Later it will be shown that an additional information about the initial tokens distribution allows to construct a SS even in nets with more than one ergodic components.

3.3.1 SS for particular structure of state machine Petri nets

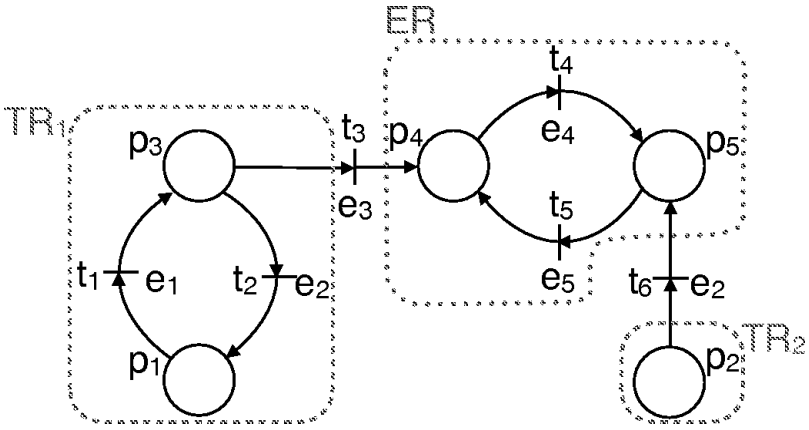


Figure 3.5: A not strongly connected SM PN with two components TR and one ER.

The following result considers a special class of nets with a single ergodic component and where each transient component is directly connected to the ergodic one. They are based on the STS approach.

Proposition 3.24. *Consider a synchronized not strongly connected SM PN* $N = (P, T, Pre, Post)$ *with a single component* ER *and* μ *transient components* $TR_1, TR_2, \ldots, TR_\mu$. *Let the net have only one token.*

Let $\sigma = \sigma_1 t'_1 \ldots \sigma_\mu t'_\mu \sigma'$, *where:* σ_i *is a synchronizing transition sequence for the set of all places belonging to* TR_i *and place* $p_i \in TR_i$; t'_i *is a transition from* p_i *to some* $p'_i \in ER$; σ' *is a synchronizing transition sequence for the set of all places belonging to* ER *and place* $\bar{p} \in ER$.

The input sequence $w = f^*(\sigma)$ *is a* 1*-SS that moves the unique token to place* $\bar{p}$.

Proof. By definition of synchronizing transition sequence, the input sequences corresponding to each couple (σ_i, t'_i) drive the token to the ergodic component. The application of the input sequence corresponding to σ' matches with the already solved problem of Proposition 3.9 wrt to the subnet induced by the ergodic component. □

The above result can be extended for the case of k tokens.

Proposition 3.25. *Consider a synchronized not strongly connected SM PN* $N = (P, T, Pre, Post)$ *with a single component* ER *and* μ *transient components* $TR_1, TR_2, \ldots, TR_\mu$. *Let the net have* k *tokens.*

Let $\sigma = \sigma_1 t'_1 \ldots \sigma_\mu t'_\mu \sigma'$ *be the transition sequence determined in Proposition 3.24 where:* σ_i *is a synchronizing transition sequence for the set of all places belonging to* TR_i *and place* $p_i \in TR_i$; t'_i *is a transition from* p_i *to some* $p'_i \in ER$; σ' *is a synchronizing transition sequence for the set of all places belonging to* ER *and place* $\bar{p} \in ER$.

The input sequence $w = f^*(\sigma_1 t'_1)^k \ldots f^*(\sigma_\mu t'_\mu)^k f^*(\sigma')^k$ *is a* k*-SS that moves all the tokens to place* $\bar{p}$.

Proof. Consider the input sequence $f^*(\sigma_i t'_i)$ for component TR_i. For the same reasoning of Proposition 3.24, this input sequence will drive at least one of the token initially in TR_i to ER. Thus, any other further application of this sequences will drive all tokens initially in TR_i (whose number is less than or equal to k) to ER. After the application of the input $f^*(\sigma_1 t'_1)^k \ldots f^*(\sigma_\mu t'_\mu)^k$ all k tokens will be in the component ER. Finally the input sequence $f^*(\sigma')^k$ will move all tokens to place $\bar{p}$. □

Example 3.26. *Consider the SM PN in Figure 3.5 with three tokens. One wants to find a SS that drives those tokens to place* p_4, *then reaching the marking* $M = [0\,0\,0\,3\,0]^T$. *Let it be* $(\sigma_1, t'_1) = (t_1, t_3)$ *and* $(\sigma_2, t'_2) = (\varepsilon, t_6)$. *Let* $\sigma' = t_5$ *be the synchronizing transition sequence from each of the places of the component ER to place* p_4, *having* $^\bullet t_4 = p_4$. *It holds that* $w = f^*(t_1 t_3)^3 f(t_6)^3 f(t_5)^3 = \{e_1 e_3\}^3 \{e_2\}^3 \{e_5\}^3 = e_1 e_3 e_1 e_3 e_1 e_3 e_2 e_2 e_2 e_5 e_5 e_5$ *is a SS.* ■

3.3.2 SS for more general state machine Petri nets

It is now proposed an algorithm to determine sequences for non strongly connected state machines having a single ergodic component where the interconnection between transient

components can be arbitrary.

It is first stated the following result.

Proposition 3.27. *Consider a SM PN $N = (P, T, Pre, Post)$ with a single component ER and let $\mathcal{C}(N)$ be its condensed graph. For each node v_i of $\mathcal{C}(N)$ associated with a transient component TR_i (with $i > 0$), let l_i be the length of the longest path from v_i to node v_0 associated with the ergodic component ER. Then if there is an edge (v_i, v_j) in $\mathcal{C}(N)$ it holds $l_i > l_j$.*

Proof. First observe that $\mathcal{C}(N)$ is acyclic by construction and the node v_0 is reachable from any other node, hence $l_j \in \mathbb{N}$ is well defined for each node v_j (with $j > 0$). By definition, if (v_i, v_j) is an edge of $\mathcal{C}(N)$, then $l_i \geq l_j + 1$. □

The following algorithm for the one-token case allows to obtain a SS, such that a place $\bar{p}$ in the single ergodic component is marked.

Algorithm 3.28. ***(Computing a SS leading to $\bar{p} \in ER$)***

Input: *A synchronized SM PN $\langle N, E, f \rangle$ containing 1 tokens, with μ components TR and 1 component ER. A target place $\bar{p}$.*
Ouput: *A SS $\bar{w}$ for marking $\bar{M}$, such that $\bar{M}(p) = 1$ if $p = \bar{p}$, 0 otherwise.*

1. *Let $\mathcal{C}(N)$ be the condensed graph of N and associate ER with node v_0.*
2. *Label every other node v_i of $\mathcal{C}(N)$ with l_i, where l_i is the length of the longest path from v_i to v_0.*
3. *Let Σ_k be the set of nodes such that $\Sigma_k = \{v_i : l_i = k\}$, thus for construction $\Sigma_0 = \{v_0\}$.*
4. *Let $w = \varepsilon$.*
5. ***For*** *$k = l_{max}, l_{max} - 1, \ldots 1$,*
   - 5.1. ***for*** *all $v_i \in \Sigma_k$,*
     - 5.1.1. *pick any transition t' connecting v_i to v_j, being $v_j \in \Sigma_{k'}$ and $k' < k$;*
     - 5.1.2. *consider the strongly connected subnet associated with node v_i. Determine a 1−SS w' for place p', where $p' \in t'^{\bullet}$;*
     - 5.1.3. *let $w = ww'$.*

     End for

   End for
6. *Consider the strongly connected subnet associated with node v_0. Let w' be a 1−SS for place $\bar{p}$.*
7. *Let $\bar{w} = ww'$.* ■

The algorithm starts taking into account the farthest nodes from ER. By definition of condensed graph, transient nodes with the same label value are not connected. Hence at step 5.1.2 the application of each couple (w', t'), step by step, drives the token always nearer to ER until it reaches it.

We have remarked that a net with more than one ergodic component cannot have a SS, at least according to Definition 3.23. However, the knowledge of the initial tokens distribution among the net components may lead to other interesting characterizations, provided of course the initial state uncertainty is redefined according to this new information.

3.4 Synchronizing sequence on synchronized Petri nets containing state machine subnets

In the following we discuss some results on synchronized PNs which do not belong to the class of SM PNs, but which do contain SM PNs.

This class of PNs has been proved to be a powerful framework for modeling shared resources among concurrent processes and providing analysis of deadlock avoidance and resolution, hence to give solutions to the so-called *resource allocation problem* (RAP) [73]. Systems of this kind are often called *resource allocation systems* (RAS) [28, 80] and PNs have been considerably used when dealing with RAP [41, 42, 79].

There exists an extensive literature for PN models while modeling RAS. In particular we may cite the class of S^3PR nets [41], S^4PR nets [97, 124], S^*PR nets [43], $NS-RAP$ [42], $ERCN$–merged nets [133] or PR nets [65].

In this setting, resources may be modeled by places and their instance by tokens. The advancing of tokens along the SM PNs represents the sequentiality of the processes. Any arc from a resource place to a transition (resp. from any transition to a resource place) represents the obtainment (resp. the release) of some resources by a process.

We now show how — under certain conditions — our approach can be further extended to this more general setting.

Proposition 3.29. *Consider a synchronized PN $\langle N, E, f\rangle$. Let $P = P_s \cup P_z$ and $T = T_s \cup T_z$, such that $N_s = (P_s, T_s, Pre_s, Post_s)$ is a strongly connected SM PN subnet, where Pre_s and $Post_s$ are the restrictions of Pre and $Post$ to $P_s \times T_s$.*

Let $\bar{w}$ be a SS that drives the subnet N_s to a target marking $\bar{M}_s$. This sequence $\bar{w}$ drives N to a target marking $\bar{M}$ such that:

$$\bar{M}(p) = \bar{M}_s(p) \;\; if \;\; p \in P_s,$$

if the two following conditions hold:

i) $\{T_z^\bullet \cup {}^\bullet T_z\} \cap P_s = \varnothing$;

ii) $(\forall e \in \bar{w})\ T_e \cap P_z^\bullet \cap T_s = \varnothing$.

Proof. Condition i) states that no transition $t \in T_z$ is connected to any place $p \in P_s$. This ensures that the firing of a transition in T_z cannot affect the marking of places in P_s. Hence, given the special structure of N_s, the following condition holds for any initial marking M_0: $(\forall M \in R(N, M_0))\ \sum_{p \in P_s} M(p) = \sum_{p \in P_s} M_0(p)$, i.e., the token count in the SM PN component remains constant.

Let $\bar{w} = e_1 e_2 \cdots e_k$ be a SS for subnet N_s that yields a known marking $\bar{M}_s$ from any reachable marking M of the subnet. To prove the result, it is sufficient to show that at each step $i = 1, \ldots, k$ the same sequence, applied to N from any marking M', with $M'(p) = M(p)$ if $p \in P_s$, produces exactly the same transition firings that it produces in N_s.

In fact, when a input symbol $e_i \in \bar{w}$ is applied:

- all transitions that can fire in N_s can also fire in N, because the additional places P_z in N cannot disable these transitions since they do not belong to ${}^\bullet T_s$;
- no transition in $P_z^\bullet$ can fire, because no transition in $P_z^\bullet$ has label e_i. □

Such a result can be further generalized to nets containing more than one SM subnets.

Proposition 3.30. *Consider a synchronized PN $\langle N, E, f \rangle$. Let $P_s \cup P_z = P$ and $T_s \cup T_z = T$, where $P_s = \dot{\bigcup}_{i=1}^{n} P_{s,i}$ and $T_s = \dot{\bigcup}_{i=1}^{n} T_{s,i}$ (here $\dot{\cup}$ denotes the union of disjoint subsets). These sets are such that for $i = 1, 2, \ldots n$ $N_{s,i} = (P_{s,i}, T_{s,i}, Pre_{s,i}, Post_{s,i})$ is a strongly connected SM PN subnet. $Pre_{s,i}$ and $Post_{s,i}$ are the restrictions to Pre and $Post$ to $P_{s,i} \times T_{s,i}$.*

For every subnet $N_{s,i}$, let $\bar{w}_i$ be a SS that drives the subnet $N_{s,i}$ to a target marking $\bar{M}_{s,i}$. The sequence $\bar{w} = \bar{w}_1 \bar{w}_2 \ldots \bar{w}_n$ drives N to a target marking $\bar{M}$ such that:

$$\bar{M}(p) = \bar{M}'_{s,i} \text{ with } \bar{M}_{s,i} \xrightarrow{\bar{w}_{i+1} \bar{w}_{i+2} \cdots \bar{w}_n} \bar{M}'_{s,i} \quad if \ \ p \in P_{s,i},$$

if the two following conditions hold:

i) $\{{}^\bullet T_z \cup T_z^\bullet\} \cap P_s = \varnothing$;

ii) $(\forall e \in \bar{w}_i)\ T_e \cap P_z^\bullet \bigcap_{j=1}^{i} T_{s,j} = \varnothing$.

Proof. The proof follows along the same lines of the proof of Proposition 3.29 with just an additional consideration. First we observe that condition ii) in Proposition 3.30 is a generalization of condition ii) in Proposition 3.29: the condition now must hold not only for the transitions of net i but also for those of all nets j with $j < i$. In fact, the overall SS is composed by concatenation of the SSs for each state machine subnet. When we apply the SS w_i to the net, we assume that the markings of all subnets j, for $j < i$ are known but may change, as some transitions in the already synchronized subnets may be receptive to an event $e \in \bar{w}_i$. However, condition ii) ensures that the enabling condition of these transitions does not depend on the places in P_z, whose marking is unknown, and the marking reached after the application of event e is computable. □

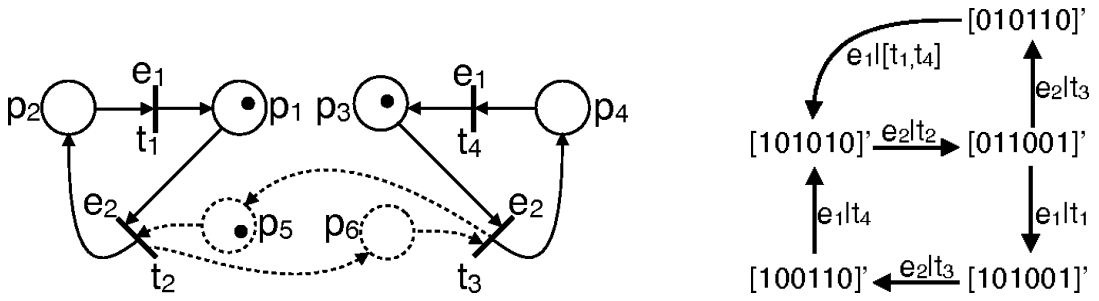


Figure 3.6: A marked synchronized PN containing SM subnets (a) and its RG

Sequence $\bar{w}$ determined in the previous proposition is a SS for the subnet N_s. It also drives the complete model N to a state where the marking of places in P_s is known, while in general nothing can be said about the marking of places in P_z.

Example 3.31. *Consider the net in Figure 3.6a. Let* $P_{s,1} = \{p_1, p_2\}$, $P_{s,2} = \{p_3, p_4\}$, $P_z = \{p_5, p_6\}$, $T_{s,1} = \{t_1, t_2\}$, $T_{s,2} = \{t_3, t_4\}$ *and then* $T_z = \varnothing$. N_s *is then the net depicted in Figure 3.6a, without taking into account dashed places and arcs. Let* $\bar{w}_1$ *and* $\bar{w}_2$ *be SSs that drives respectively* $N_{s,1} = (P_{s,1}, T_{s,1}, Pre_{s,1}, Post_{s,1})$ *to* $\bar{M}_{s,1} = [0\,1]^T$ *and* $N_{s,2} = (P_{s,2}, T_{s,2}, Pre_{s,2}, Post_{s,2})$ *to* $\bar{M}_{s,2} = [0\,1]^T$. *By separately analyzing the two subnets,* $\bar{w}_1 = \bar{w}_2 = e_1$ *are obtained.* $\bar{w} = \bar{w}_1\bar{w}_2$ *respects conditions i) and ii) of Proposition 3.30 and is therefore a SS for* N_s, *i.e., it drives the net to a marking* $\bar{M}$ *that is either* $[1\,0\,1\,0\,1\,0]^T$ *or* $[1\,0\,1\,0\,0\,1]^T$, *as can be seen by its RG in Figure 3.6b.* □

Proposition 3.30 and Proposition 3.29 apply to nets containing SM PNs so that they provide a SS that leads from any reachable marking to a marking where only the token content of places of the SM subnetsis known. This concept of partial synchronization will be proposed again for the the case of unbounded PNs in Chapter 4.

However, it can be proved that the marking of places not belonging to the SM subnets can be reconstructed, if the additional places introduce constraints on the firing rules of the net which belongs to certain classes of specifications.

This is the case of the *generalized mutual exclusion constraints* (GMECs) [55], which are conditions that limit a weighted sum of tokens contained in a subset of places.

Definition 3.32. ***(GMEC)*** *Let* $\langle N, M_0 \rangle$ *be a marked net with a set of places* P. *A single generalized mutual exclusion constraint* $(\vec{h}, k)$ *defines a set of legal markings*

$$\mathcal{M}(\vec{h}, k) = \{M \in \mathbb{N}^{|P|} : \vec{h}^T \cdot \leq k\},$$

where $\vec{h}$ *is a weight vector, and* $k \in \mathbb{N}^+$ [1]. ■

A GMEC is a natural way to express the concurrent use of a finite number of resources which have to be shared among different processes. A single GMEC can be easily implemented by a *monitor* [55]. In the area of the manufacturing design an important class of PNs, providing an important model for RAS, is represented by nets composed by SMs

[1] Here, $\mathbb{N}^+ = \{1, 2, 3, 4, \ldots\}$.

and monitors, where monitors are places whose marking represents the number of current available units of a resource and whose outgoing (resp. ingoing) transitions represent the acquisition (resp. the release) of units of the resource.

Definition 3.33. ***(Monitor)*** *Given $\langle N, M_0\rangle$ be a marked net with $N = (P, T, Pre, Post)$, and a GMEC $(\vec{h}, k)$, the monitor that enforces this constraint is a new place r to be added to N. The resulting marked PN is denoted $\langle N^r, M_0^r\rangle$, with $N = (P \cup \{r\}, T, Pre^r, Post^r)$. Let C be the incidence matrix of N. Then N^r will have the incidence matrix:*

$$C^r = \begin{bmatrix} C \\ -\vec{h}^T \cdot C \end{bmatrix}.$$

It is assumed that there are no self-loops containing r in N^r, hence Pre^r and $Post^r$ may uniquely determined by C^r. It is also assumed that the initial marking of M_0 of the marked PN satisfies the constraint $(\vec{h}, k)$. ■

While constructing a SS, according to its definition, the marking of all monitors is unknown, but we are given their upper bound, since the number of resources acquirable by the processes are bounded. However every GMEC entails a P-invariant, i.e., the marking of monitors can be inferred by the marking of some places of the subnets. Here, we deal with the more general case of nets containing SM subnets and a set of monitors.

Proposition 3.34. *Consider a bounded synchronized PN $\langle N, E, f\rangle$. Let $P_s \cup P_r = P$, where $P_s = \biguplus_{i=1}^{n} P_{s,i}$, $P_r = \{r_1, r_2, \dots r_z\}$ and $T = \biguplus_{i=1}^{n} T_{s,i}$ (here $\biguplus$ denotes the union of disjoint subsets). These sets are such that for $i = 1, 2, \dots n$ $N_{s,i} = (P_{s,i}, T_{s,i}, Pre_{s,i}, Post_{s,i})$ is a strongly connected SM PN subnet. Every place $r_i \in P_r$ is a monitor satisfying the constraint $(\vec{h}_i, k_i)$. $Pre_{s,i}$ and $Post_{s,i}$ are the restrictions to Pre and $Post$ to $P_{s,i} \times T_{s,i}$.*

For every subnet $N_{s,i}$, let $\bar{w}_i$ be a SS that drives the subnet $N_{s,i}$ to a target marking $\bar{M}_{s,i}$. The sequence $\bar{w} = \bar{w}_1\bar{w}_2 \dots \bar{w}_n$ drives N to a target marking $\bar{M}$ such that:

$$\bar{M} \in \mathbb{N}^{|P|}: \quad \bar{M}(p) = \begin{cases} k_i - \vec{h}^T \cdot \bar{M}, & if \;\; p = r_i \\ \bar{M}'_{s,i}(p) \; with \; \bar{M}_{s,i} \xrightarrow{\bar{w}_{i+1}\bar{w}_{i+2}\cdots\bar{w}_n} \bar{M}'_{s,i} & if \;\; p \in P_{s,i}. \end{cases}$$

if the following condition holds:

i) $(\forall e \in \bar{w}_i)\; T_e \cap P_r^{\bullet} \bigcap_{j=1}^{i} T_{s,j} = \varnothing.$

Proof. The proof follows on the proof of Proposition 3.30. We observe that condition ii) of Proposition 3.30 here corresponds to condition i) and that condition ii) here is not requested, since $T = T_s$ thus it would always be satisfied. The marking of every SM subnet is computed by applying each SS and then updating them with reference to the further applied subsequences. Finally the marking of all monitors is computed by the aid of the set of constraints enforced by the monitor itself. □

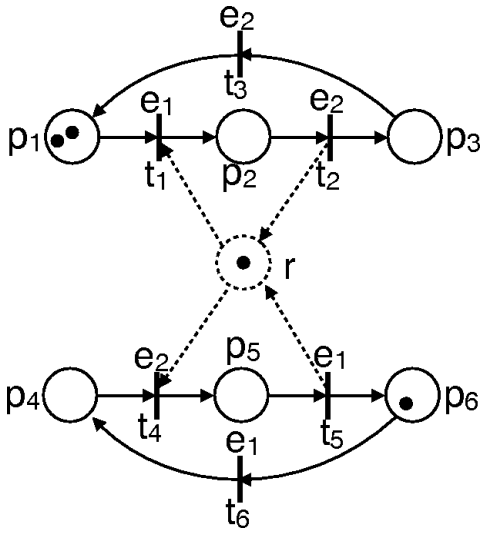


Figure 3.7: A marked synchronized PN with a monitor

Example 3.35. *Consider the marked synchronized PN in Figure 3.7. Let* $P_{s,1} = \{p_1, p_2, p_3\}$, $P_{s,2} = \{p_4, p_5, p_6\}$, $P_r = \{r\}$, $T_{s,1} = \{t_1, t_2, t_3\}$ *and* $T_{s,2} = \{t_4, t_5, t_6\}$. N_s *is then the net depicted in Figure 3.7, without taking into account the dashed monitor and dashed arcs. Let* $(\vec{h}, k)$ *be the monitor with* $\vec{h} = [0\,1\,0\,0\,1\,0]^T$ *and* $k = 1$, *i.e.,* $M(p_2) + M(p_5) \leq 1$.

Let $\bar{w}_1$ *and* $\bar{w}_2$ *be the SSs that drives respectively* $N_{s,1} = (P_{s,1}, T_{s,1}, Pre_{s,1}, Post_{s,1})$ *to* $\bar{M}_{s,1} = [2\,0\,0]^T$ *and* $N_{s,2} = (P_{s,2}, T_{s,2}, Pre_{s,2}, Post_{s,2})$ *to* $\bar{M}_{s,2} = [1\,0\,0]^T$. *By separately analyzing the two subnets,* $\bar{w}_1 = e_2e_2e_2$ *and* $\bar{w}_2 = e_1e_1$ *are obtained.* $\bar{w} = \bar{w}_1\bar{w}_2$ *respects condition i) of Proposition 3.34, hence it is a SS which drives the net to marking* $\bar{M} = [2\,0\,0\,1\,0\,0\,1]^T$, *where the marking of the monitor is easily computed by the way of the inequality its constraints entail.* □

3.5 A Matlab toolbox for synchronizing sequences construction

In this section, the main MATLAB functions for for synchronization of system modeled by automata and synchronized PNs are briefly illustrated. They can be downloaded on the web [100].

In the following, we assume that an automaton is represented by an cell matrix that has as many rows/column as the automaton's cardinality. Consider row i and column j, the cell contains a line vector with the indices of the input events that cause the transition from state x_i to state x_j.

For instance, the automaton in Figure 2.1 is represented by the following cell matrix:

$$\Lambda = \{[1], [\,][2]; [2], [1][\,]; [1], [2][\,]\}.$$

Note that the output events are not taken into account since not of interest for the

synchronization scope.

A marked synchronized PN is represented by the two matrices (Pre and $Post$), one column vector (M_0) and one row vector (E_{in}): the pre and post incidence matrices, the initial marking and the labeling function with as much element as the number of transitions. In particular, for the latter, the value of element i represents the index of the event associated to transition t_i.

For example, the marked synchronized PN in Figure 2.4a is represented as follows:

$$Pre = \begin{pmatrix} 2, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{pmatrix} \quad Post = \begin{pmatrix} 0, & 1, & 1 \\ 1, & 0, & 0 \\ 0, & 1, & 0 \end{pmatrix} \quad M = [2; 0; 0] \quad E_{in} = [1, 2, 1]$$

3.5.1 Main functions for SS construction with automata

aux_graph.m: computes the auxiliary graph of an automaton. Self-loops are omitted.

Its input is:

- the structure of the automaton Λ.

Its output is:

- a cell matrix a_graph that contains for each row: i) an unordered pair of state of (x_i, x_j) identifying the label of graph node, ii) the number of arcs outgoing the node, iii) the triple (x'_i, x'_j, i), identifying the reached node and the input event causing the transition.

Consider the auxiliary graph in Figure 3.1b. Line 2 of matrix a_graph, representing node (M_0, M_1), would be :

$$a\_graph\{2, :\} = \{[0, 1], 2, [1, 1, 1], [0, 2, 0]\}$$

check_scc.m: checks if an automaton is strongly connected.

Its input is:

- the structure of the automaton Λ.

It outputs a boolean value which is equal to 1, if the graph associated with the automaton is strongly connected, 0 otherwise.

randomFSM.m: randomly computes an automaton.

Its inputs are:

- the cardinality of the set of states;
- the cardinality of the input alphabet.

Its output is:

- the structure of the automaton Λ.

show_AUX_GRAPH.m: provides a textual representation of the *aux_graph.m* output data function. The auxiliary graph can be constructed step-by-step starting from the initial node following all input events' application and corresponding reached nodes.

synch_seq.m computes the SS of an automaton for a given final state

Its input are:

- the structure of the automaton Λ;
- the corresponding auxiliary graph;
- a desired final state $\bar{x}$.

Its output is

- a row vector with the indices of the input events characterizing the SS.

s_reached.m simulates a the behavior of a given automaton while applying an input event.

Its inputs are:

- the structure of the automaton Λ;
- an initial state x_0;
- an input event.

It outputs the new reached state.

3.5.2 Main functions for SS construction with synchronized PNs

CSU.m: builds the current state uncertainty of a synchronized PN, by applying Algorithm 2.16 to the initial state uncertainty.

Its inputs are:

- a synchronized PN defined by its Pre and $Post$ matrices and its labeling function E_{in};
- the initial marking or the set of markings defining the initial state uncertainty $MM0$;
- an input events' sequence;
- a boolean variable $show$, which if is equal to 1 provides a textual representation of the output data.

Its output is

- a matrix that has as many row as the length of the input sequence plus one. Row i contains respectively i) an identifier number of step; ii) the markings where the net may be after the application of event $e_i \in w$; iii) the firing vector associated with this event.

CG_synch.m builds the coverability/reachability graph (CG/RG) of a marked synchronized Petri net. It follows from of Algorithm 4.5.

Its inputs are:

- a marked synchronized PN defined by its Pre and $Post$ matrices, its initial marking and its labeling function E_{in};
- a variable (woc), which is used for unbounded PNs while introducing the ω value This is done by analyzing, for each node M of the tree, all of its ancestors that are covered by M. A place takes an ω value if there exists a set of marking $\mathcal{M}'$ covered by M. Three cases are possible:

  $woc = 1$**:** only the first encountered node $M' \in \mathcal{M}'$ is taken into account (this case is the one implemented by Algorithm 4.5);

  $woc = 2$**:** all marking $M' \in \mathcal{M}'$, are taken into account all at once and $M(p) = \omega$ if $\exists M' \in \mathcal{M}' : M \geqneq M'$ and $M(p) > M'(p)$.

  $woc = 3$**:** all marking $M' \in \mathcal{M}'$, are taken into account and the marking M is updated recursively from the nearest encountered ancestor to the farthest one following the previous case rule.

Its outputs are

- the coverability/reachability tree and graph of the synchronized PN. Both matrices have the same structure. Row i contains all information about of node i: i) the marking; ii) the outgoing arcs specified by reached node, input event and firing vector that label them. Both graph has as many rows as the number of nodes per graph plus one additional row containing some information of the net in terms of number of places, transitions and events.

enumeratingSM.m gives an estimate of the number of m-places deterministic synchronized SMs over a k-events alphabet. Note that this estimate is a lower bound, since it takes into account only totally synchronized PNs. For more details the reader shall refer to [115]

makeFSM_s.m builds the completely specified automaton corresponding to the reachability graph of a synchronized PN.

Its input is:

- the reachability graph of a synchronized PN.

Its outputs are:

- a cell matrix corresponding to the structure of the automaton equivalent to the RG;
- a matrix containing all the reachable markings associated with each of this states.

randomSM_PN.m randomly computes a strongly connected and deterministic SM PN.

Its inputs are:

- the cardinality of the set of places;
- the cardinality of the set of transitions;
- the cardinality of the set set of input events;

Its outputs are:

- the (Pre and $Post$) matrices;
- the labeling function specified by the matrix E_{in};
- the number of tries before successfully exiting.

showSTS.m provides a textual representation of the $STS.m$ output data function.

STS.m computes the synchronizing transition sequence (STS) for a given State Machine PN and a desired final place.

Its inputs are:

- the (Pre and $Post$) matrices;
- the labeling function specified by the matrix E_{in};
- the desired place.

Its output is:

- a matrix cell with as many row as the STS found.

3.6 Comparisons among the proposed approaches for bounded synchronized Petri nets

This section has two objectives. First, we compare the three algorithms we have presented for SS computation on state machine Petri nets (reachability graph based approaches versus path based approach) by applying them to a series of randomly generated nets and analyzing their performance. Second, we show the effectiveness of our approach by applying it to a manufacturing system modeled by a PN which is not a SM.

The model data and MATLAB programs can be downloaded from [100].

All simulations have been run on a mini Mac intel core Duo 2, 2.53 GHz processor, with 4 GB 1067 MhZ DDR3 RAM.

3.6.1 Numerical results for randomly generated PNs

In order to evaluate the quality of the three proposed approaches, a second experiment has been performed. This subsection takes into account randomly generated models, whose construction is later presented in detail in Section 3.5.

Randomly generated models have been previously adopted as a validation method for synchronizing sequence construction also by Roman [108].

For selected values of m places, q transitions and k tokens, we randomly generate 100 synchronized SM PNs which are a) deterministic and b) strongly connected, with:

i) $m = 2 \div 7$ places;

ii) $q = m \div 15$ transitions;

iii) $k = 1, 2$ tokens.

q \ m	2	3	4	5	6
1	4	12	32	80	192
2	16	324	8192	250000	8957952
3	64	8748	2097152	7.8125e+8	4.1794e+11
4	256	236196	5.3687e+8	2.4414e+12	1,9499e+16
5	1024	6.3772e+6	1.3743e+11	7,6293e+15	9,0976e+20
6	4096	1.7218e+8	3.5184e+13	2,3841e+19	4,2446e+25

Table 3.1: The number of deterministic synchronized SM PNs over a $k-$event alphabet and a p places.

To do so, we resort to function "*randomSM_ PN.m*". Every computation stops either when such a SM PN is found or when no deterministic and strongly connected SM PN exists for the chosen setting of number of places, transitions and input events.

A good enumeration can suggest methods for random generation, since this process clearly has a stop criterium given by the number (or an upper bound) of synchronized SMs for which it holds a) and b).

In automata framework the enumeration problem for arbitrary input and output alphabet size has been solved. Such results are easily applicable to SMs, since, as previously shown, SMs are isomorphic to its reachability graph in the 1−token case.

This number of generable deterministic SMs, for a given number of m places and q events, is denoted $S(m,q)$. Totally synchronized SMs are considered for the sake of simplicity but more general labeling function can be easily considered.

The following exact formula for one event input alphabet holds:

$$S(m,1) = \sum_{m_1+2m_2+3m_3+\ldots=n} \left(\Pi_{i\geq 1} \left(\sum_{j|i} j m_j \right)^{m_i} \frac{i^{-m_i}}{m_i!} \right),$$

which is demonstrated to be asymptotically equal to:

$$A m^{-\frac{1}{2}} \tau^m,$$

where $A \doteq 0.45$ and $\tau \doteq 2.94$.

A pretty good estimate of the number of SMs of $m-$places over a $q-event$ alphabet is given by the following formula, known as the *Domaratzki-Kisman* theorem [37]:

$$S(m,q) = S(m,1) n^{(q-1)m} \simeq m 2^{m-1} m^{(q-1)m}.$$

Table 3.1 gives $S(m,q)$ for $2 \leq m \leq 6$ and $1 \leq q \leq 6$.

For more details in model enumeration, the reader shall refer to [115].

In all cases the input alphabet has cardinality f randomly chosen in $\frac{q}{m} \div q$. Note that $\frac{q}{m}$ is the minimal alphabet cardinality to ensure the determinism for a SM having m places and q transitions. For each net a place is randomly selected and we use Algorithm 3.3 (denoted RG), Algorithm 3.10 (denoted by STS) and Algorithm 3.18 (denoted by MRG) to determine a SS such that this place is marked. The algorithms are compared by means of three performance indexes:

$\mathcal{N}_{RG}$, $\mathcal{N}_{STS}$, $\mathcal{N}_{MRG}$: number of times the algorithm successfully terminates returning a SS;

$\hat{\mathcal{T}}_{RG}$, $\hat{\mathcal{T}}_{STS}$, $\hat{\mathcal{T}}_{MRG}$: average time required to compute the sequence;

$\hat{L}_{RG}$, $\hat{L}_{STS}$, $\hat{L}_{MRG}$: average length of the sequences.

Finally the performance of the two approaches is evaluated by comparing 2 approaches at time and computing the ratio of their indexes. First it is evaluated the effectiveness of the three approaches in terms of number of successful terminations, by comparing the STS and MRG approach to the RG approach. The RG approach, if the net is synchronizable, always finds a solution, so it is here the chosen as a basis for comparison. Then, computational time and length of the found sequences are compared. Here results are referred and related to the STS approach, since it is expected to be faster than the others.

Results are shown in Table 3.3 to Table 3.7 for nets with one token and in Table 3.8 and Table 3.9 for the two tokens' case. Black cells denote parameter values for which no strongly connected SM can be generated, i.e., for $m > q$.

Note that the tables showing $\mathcal{N}_{STS}/\mathcal{N}_{RG}$, $\mathcal{N}_{MRG}/\mathcal{N}_{RG}$,$\hat{\mathcal{T}}_{STS}/\hat{\mathcal{T}}_{MRG}$ and $\hat{L}_{STS}/\hat{L}_{MRG}$ do not depend on the number of tokens and thus they are shown only for $k = 1$.

Table 3.2 shows the ratio $\mathcal{N}_{STS}/\mathcal{N}_{RG}$ between the number of times a SS has been found using the STS and the RG approach. In the previous sections we have mentioned that while the RG approach always determines a SS if any exists, the STS approach may fail to do so. Hence the value in the table should be contained in the interval $[0, 1]$. We can observe, however, that the average ratio of those indexes is about 0.78 hence confirming that the STS approach can find a solution in most cases and thus this result is not too restrictive.

Table 3.3 shows the ratio $\mathcal{N}_{MRG}/\mathcal{N}_{RG}$ between the number of times a SS has been found using the MRG and the RG approach. In the previous sections we have mentioned that the MRG approach is based on a greedy computation like the RG approach, but applied to a pruned graph. Hence the value also in this table should be contained in the interval $[0, 1]$. However, results of MRG approach are even better of the STS approach, since the average ratio of those indexes is about 0.87. The additional constraints, introduced to modify the RG approach, are preventing the approach to find a solution for an even smaller number of nets.

Table 3.4 shows the ratio $\hat{\mathcal{T}}_{STS}/\hat{\mathcal{T}}_{RG}$ between the average execution time to compute a SS using the STS and the RG approach for nets with one token. Here we expect the STS approach to be more efficient, as discussed in Section 3.2.2, and this is confirmed from

q \ m	2	3	4	5	6	7
3	0.900	0.900				
4	0.900	0.900	0.812			
5	0.900	0.780	0.717	0.690		
6	0.809	0.644	0.717	0.664	0.685	
7	0.850	0.809	0.682	0.664	0.707	0.703
8	0.823	0.809	0.644	0.644	0.767	0.688
9	0.876	0.717	0.780	0.734	0.682	0.721
10	0.850	0.734	0.700	0.682	0.652	0.701
11	0.888	0.823	0.717	0.823	0.767	0.654
12	0.888	0.809	0.750	0.700	0.767	0.745
13	0.900	0.876	0.765	0.734	0.799	0.786
14	0.888	0.823	0.795	0.644	0.826	0.812
15	0.900	0.850	0.780	0.809	0.799	0.803

Table 3.2: $\mathcal{N}_{STS}/\mathcal{N}_{RG}$ ($k = 1$)

q \ m	2	3	4	5	6	7
3	1.000	1.000				
4	1.000	1.000	0.925			
5	1.000	0.867	0.797	0.780		
6	0.899	0.716	0.797	0.737	0.760	
7	0.944	0.899	0.758	0.737	0.786	0.810
8	0.915	0.899	0.716	0.716	0.852	0.824
9	0.973	0.797	0.867	0.815	0.758	0.803
10	0.954	0.815	0.778	0.758	0.725	0.704
11	0.987	0.915	0.797	0.915	0.852	0.860
12	0.987	0.899	0.833	0.778	0.852	0.838
13	1.000	0.973	0.850	0.815	0.887	0.890
14	0.987	0.915	0.883	0.716	0.917	0.902
15	1.000	0.944	0.867	0.899	0.887	0.894

Table 3.3: $\mathcal{N}_{MRG}/\mathcal{N}_{RG}$ ($k = 1$)

the fact that in almost all cases the table entries are much smaller than one. Only in a few cases, for very large values of m and q, we have that the RG method is faster than the STS one. This, we believe, it is due to our implementation of the STS approach that uses a brute force depth-first search.

Table 3.5 shows the ratio $\hat{\mathcal{T}}_{STS}/\hat{\mathcal{T}}_{MRG}$ between the average execution time to compute a SS using the STS and the MRG approach for nets with one token. This table shows that the STS approach is still "faster" than the MRG approach in most of the analyzed settings but the gap is much less significant and in 10% of the cases it is even positive.

q \ m	2	3	4	5	6	7
3	0.226	0.237				
4	0.143	0.232	0.433			
5	0.242	0.212	0.202	0.602		
6	0.172	0.266	0.327	0.307	0.457	
7	0.204	0.239	0.378	0.327	0.216	0.626
8	0.159	0.269	0.506	0.416	0.292	0.161
9	0.133	0.367	0.627	0.535	0.458	0.223
10	0.184	0.485	0.548	0.6038	0.725	0.271
11	0.157	0.545	0.527	0.905	0.732	0.88
12	0.187	0.309	0.653	0.598	0.542	0.78
13	0.121	0.331	0.610	0.781	1.037	1.15
14	0.127	0.405	0.753	0.726	0.767	1.6
15	0.100	0.744	0.937	1.522	1.082	2.54

Table 3.4: $\hat{\mathcal{T}}_{STS}/\hat{\mathcal{T}}_{RG}$ ($k = 1$)

q \ m	2	3	4	5	6	7
3	0.788	0.825				
4	0.835	0.829	0.856			
5	0.818	0.860	0.878	0.880		
6	0.834	0.892	0.873	0.855	0.820	
7	0.834	0.882	0.890	0.860	0.814	0.805
8	0.826	0.872	0.926	0.879	0.832	0.854
9	0.835	0.910	0.906	0.915	0.900	0.925
10	0.848	0.905	0.934	0.932	0.821	0.865
11	0.838	0.898	1.003	0.961	0.862	0.908
12	0.837	0.941	0.984	0.944	0.876	0.897
13	0.845	0.920	0.979	1.024	0.872	0.905
14	0.843	0.930	0.977	0.945	1.030	1.058
15	0.833	0.980	0.986	1.064	1.030	0.997

Table 3.5: $\hat{\mathcal{T}}_{STS}/\hat{\mathcal{T}}_{MRG}$ ($k = 1$)

Table 3.8 shows the ratio $\hat{\mathcal{T}}_{STS}/\hat{\mathcal{T}}_{RG}$ between the execution time to compute a SS using the STS and the RG approach for nets with two tokens. Here we see the main advantage of the STS approach, and so the MRG approach, that can use a 1-SS to determine a k-SS, while the RG method had a complexity that grows polynomially with k (and exponentially with m), as discussed in Section 3.1.2. Here the advantage of the STS and MRG method is more noticeable for large values of m and q.

Table 3.6 shows the ratio $\hat{L}_{STS}/\hat{L}_{RG}$ between the average length of a SS computed using the STS and the RG approach. Table 3.7 show the ratio $\hat{L}_{STS}/\hat{L}_{MRG}$ between the average length of a SS computed using the STS and the MRG approach. Both values are computed

q \ m	2	3	4	5	6	7
3	1.000	0.835				
4	1.000	0.918	0.652			
5	1.000	0.910	0.858	0.800		
6	1.000	0.912	0.893	0.855	0.654	
7	1.000	0.832	1.000	0.850	0.534	0.547
8	1.000	1.000	0.776	0.709	0.902	0.612
9	1.000	1.100	0.856	0.915	0.790	0.574
10	1.000	0.835	0.954	0.912	0.901	0.501
11	1.000	1.000	0.843	0.961	0.802	0.881
12	1.000	0.911	1.000	0.754	0.746	0.592
13	1.000	0.770	0.949	1.000	0.862	0.76
14	1.000	1.100	0.897	0.875	0.813	0.856
15	1.000	1.000	1.060	0.924	0.700	0.692

Table 3.6: $\hat{L}_{STS}/\hat{L}_{RG}$ ($k = 1$)

q \ m	2	3	4	5	6	7
3	1.000	1.000				
4	1.000	1.000	0.980			
5	1.000	0.963	0.781	0.840		
6	1.000	0.889	0.971	0.868	1.210	
7	1.000	1.023	1.032	0.769	1.625	0.920
8	1.000	1.200	1.333	0.926	0.653	0.845
9	1.000	1.167	0.830	1.160	0.556	0.654
10	1.000	1.875	1.193	0.952	0.400	0.489
11	1.000	1.600	1.127	1.125	0.224	0.354
12	1.000	1.500	1.137	1.260	0.300	0.402
13	1.000	1.786	0.429	0.976	0.274	0.354
14	1.000	1.222	1.167	1.079	0.554	0.521
15	1.000	1.185	1.215	1.429	0.431	0.478

Table 3.7: $\hat{L}_{STS}/\hat{L}_{MRG}$ ($k = 1$)

for nets with one token. This index is probably less significant than the previous ones, although one may argue that the shortest the SS the less expensive is the synchronization (in terms of costs or of time required). Here we can see that in the case of one token the STS and MRG approach in most of the cases produce shorter SS than the RG approach. Also, the solution provided by the STS approach is the shortest one, since it computes SS via a depth first search, while RG and MRG approaches find a solution that may be not the best one. In fact, they chose a subsequence w that synchronizes at least two markings but does not pick those markings with any criterium. The evaluation could be based on some heuristics that at each step picks a couple of markings that either maximizes

q \ m	2	3	4	5	6	7
3	0.260	0.270				
4	0.120	0.164	0.287			
5	0.230	0.780	0.127	0.172		
6	0.169	0.145	0.207	0.084	10^{-3}	
7	0.170	0.148	0.232	0.084	10^{-3}	10^{-3}
8	0.133	0.217	0.316	0.194	0.091	10^{-3}
9	0.116	0.283	0.410	0.014	0.020	10^{-3}
10	0.160	0.353	0.357	0.282	10^{-3}	0.052
11	0.148	0.421	0.117	0.448	10^{-3}	0.050
12	0.158	0.246	0.390	0.270	0.200	10^{-3}
13	0.110	0.276	0.205	0.354	10^{-3}	10^{-3}
14	0.118	0.248	0.365	10^{-3}	0.295	10^{-4}
15	0.100	0.548	0.400	0.758	0.392	10^{-3}

Table 3.8: $\hat{\mathcal{T}}_{STS}/\hat{\mathcal{T}}_{RG}$ ($k = 2$)

q \ m	2	3	4	5	6	7
3	1.710	1.500				
4	1.710	1.600	1.500			
5	1.710	1.410	1.412	1.172		
6	1.710	1.600	1.420	1.425	1.392	
7	1.710	1.330	1.642	1.355	0.923	0.857
8	1.710	1.710	1.460	1.265	1.523	0.975
9	1.710	1.890	1.576	1.447	1.326	0.973
10	1.710	1.500	1.608	1.507	1.648	0.928
11	1.710	1.710	1.540	1.556	1.273	0.886
12	1.710	1.600	1.542	1.365	1.333	0.770
13	1.710	1.410	1.505	1.653	1.282	0.797
14	1.710	1.890	1.540	1.376	1.396	0.647
15	1.710	1.760	1.772	1.651	1.300	0.792

Table 3.9: $\hat{L}_{STS}/\hat{L}_{RG}$ ($k = 2$)

the number of synchronized markings or minimizes the length of such a subsequence. The situation, as shown in Table 3.9, is the opposite when the number of tokens grows. Consider for example the 2 tokens case. The k-SS obtained by STS and MRG approach is always k times longer than the corresponding 1-SS, while shorter solutions may be obtained by the RG approach.

On the base of these results, we can say that to compute a SS for strongly connected Petri nets it is convenient to first search for a STS based solution using Algorithm 3.10 and then, if this fails, to use Algorithm 3.3. This is summarized in the flowchart in Figure 3.8.

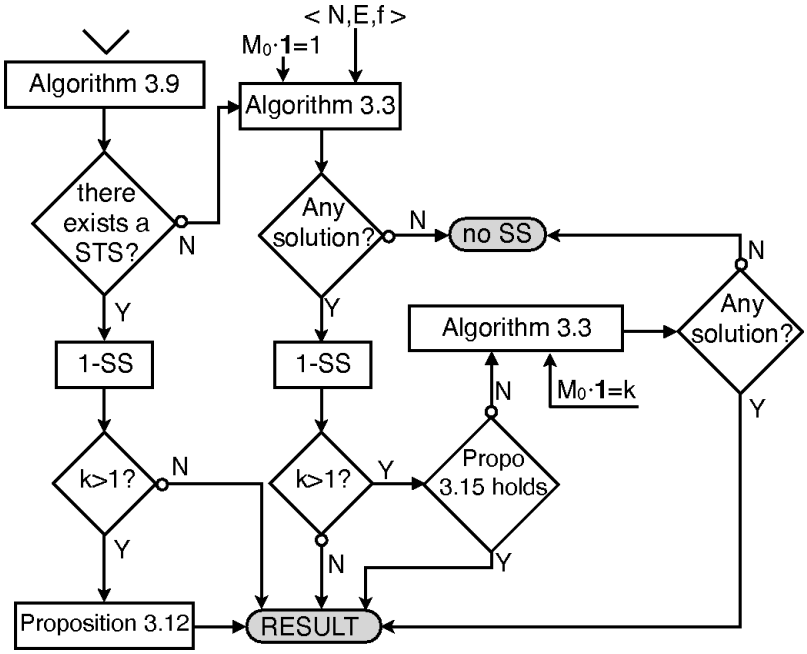


Figure 3.8: Suggested way to determine a SS for strongly connected SMs.

3.6.2 A manufacturing example

In the following, a second experiment has been performed: a family of manufacturing systems, represented by a parametric PN, is analyzed by applying the theoretical results provided in section 3.4 — for synchronized PNs composed by SM subnets — to construct a SS.

We consider a manufacturing plant consisting of two parallel and symmetric production lines, that produce two different kinds of final product. Each line i has a fixed number of k pallets, that limit the number of parts that can be under processing at a given time. A raw piece entering the system waits in the buffer (not modeled) until a pallet becomes available. Robot R_0 feeds the two lines alternatively, taking one part from the buffer and mounting it on an empty pallet.

In each line there are two workstations, which are composed by two machines and one robot. A workstation can process several pallets at a time.

For instance consider line 1. The pallet entering the system is deposed by R_0 on a conveyor belt and is moved to the first worstation where machines $M'_{1,1}$ and $M"_{1,1}$ perform their operation as requested. Robot R_{11} moves the pallet from the conveyor belt to the machines and viz.

Once the operations required on the first workstation have been completed, the pallet is put again onto the conveyor belt and moved to the second workstation where the processing is repeated. After processing, parts are unloaded from the pallets by robot $R_{1,2}$ and put on an AGV that moves them to the output buffer O_1 (not represented in Figure 3.10).

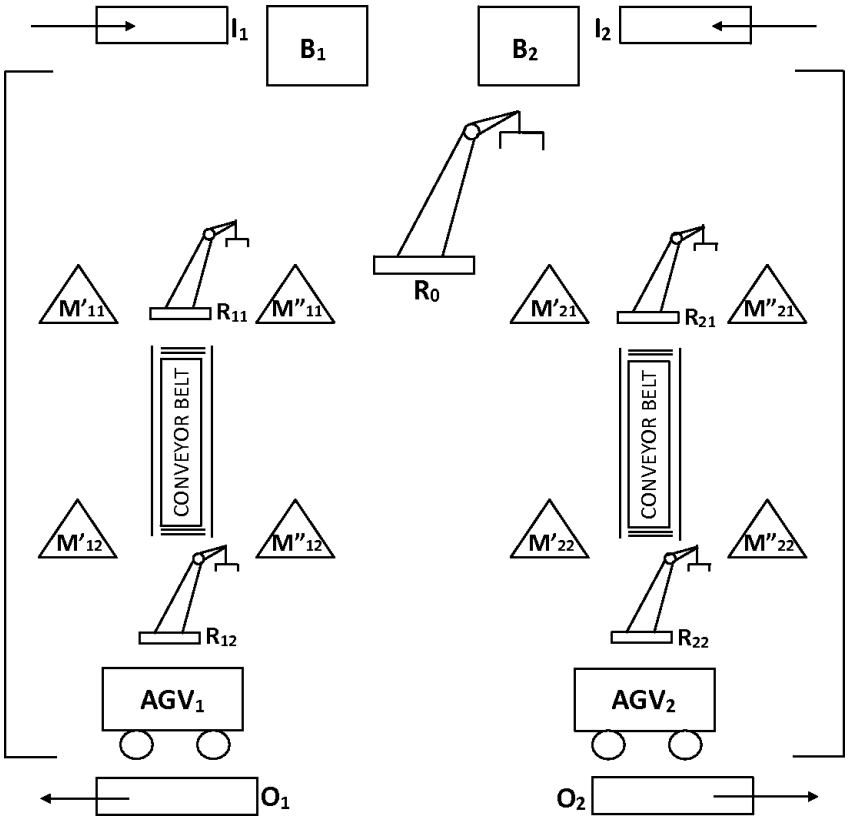


Figure 3.9: Layout of the manufacturing system.

The system's layout is shown in Figure 3.9: we say that such a plant has production length $l = 2$ because each line is composed by a series of two workstations. We can consider a parameterized family of plants of this type, assuming that the number of pallets k and length l of the production lines may vary. For this family of plants, we obtain the synchronized Petri net depicted in Figure 3.10.

This Petri net has $4 + 6l$ places and $2 + 10l$ transitions. The marking of places p_1 and p_2 represents number of the empty pallets in each line, the marking of place p_3 (resp. p_4) denotes that robot R_0 is ready to move a pallet to the left (resp. right) production line.

Consider machines M'_{ij} and $M"_{ij}$, which compose workstation j in line i. Transition $n'_{i,j}$ (resp. $n"_{i,j}$) represents the loading of a pallet from the conveyor belt queue to machine M'_{ij} (resp. $M"_{ij}$) while transition $t'_{i,j}$ (resp., $t"_{i,j}$) represents the unloading. Tokens in place p'_{ij} (resp. $p"_{ij}$) denote pallets loaded on machine M'_{ij} (resp. $M"_{ij}$).

The firing of transition $t_{i,j}$ denotes the transfer of a pallet to the next workstation.

The PN in Figure 3.10, without taking into account dashed places and arcs, is composed by two SMs. Hence, according to Proposition 3.30, $P_s = P_{s,1} \cup P_{s,2} = P \backslash \{p_3, p_4\}$, $P_{s,1} = \{p_1, p_{1,1}, p'_{1,1}, p''_{1,1}, \ldots p_{1,l}, p'_{1,l}, p''_{1,l}\}$, $P_{s,2} = \{p_2, p_{2,1}, p'_{2,1}, p''_{2,1}, \ldots p_{2,l}, p'_{2,l}, p''_{2,l}\}$ and $T_s = T$.

We look for a SS that from an arbitrary state can empty the system, thus moving all empty pallets to the input buffers. The obtained sequences are $\bar{w}_1 = \{e_4 e_5 e_6 e_7 e_2\}^l$, $\bar{w}_2 = \{e_8 e_9 e_{10} e_{11} e_3\}^l$ and $\bar{w} = \bar{w}_1 \bar{w}_2$. $\bar{w}$ meets conditions of Proposition 3.30 then is a SS for N_s.

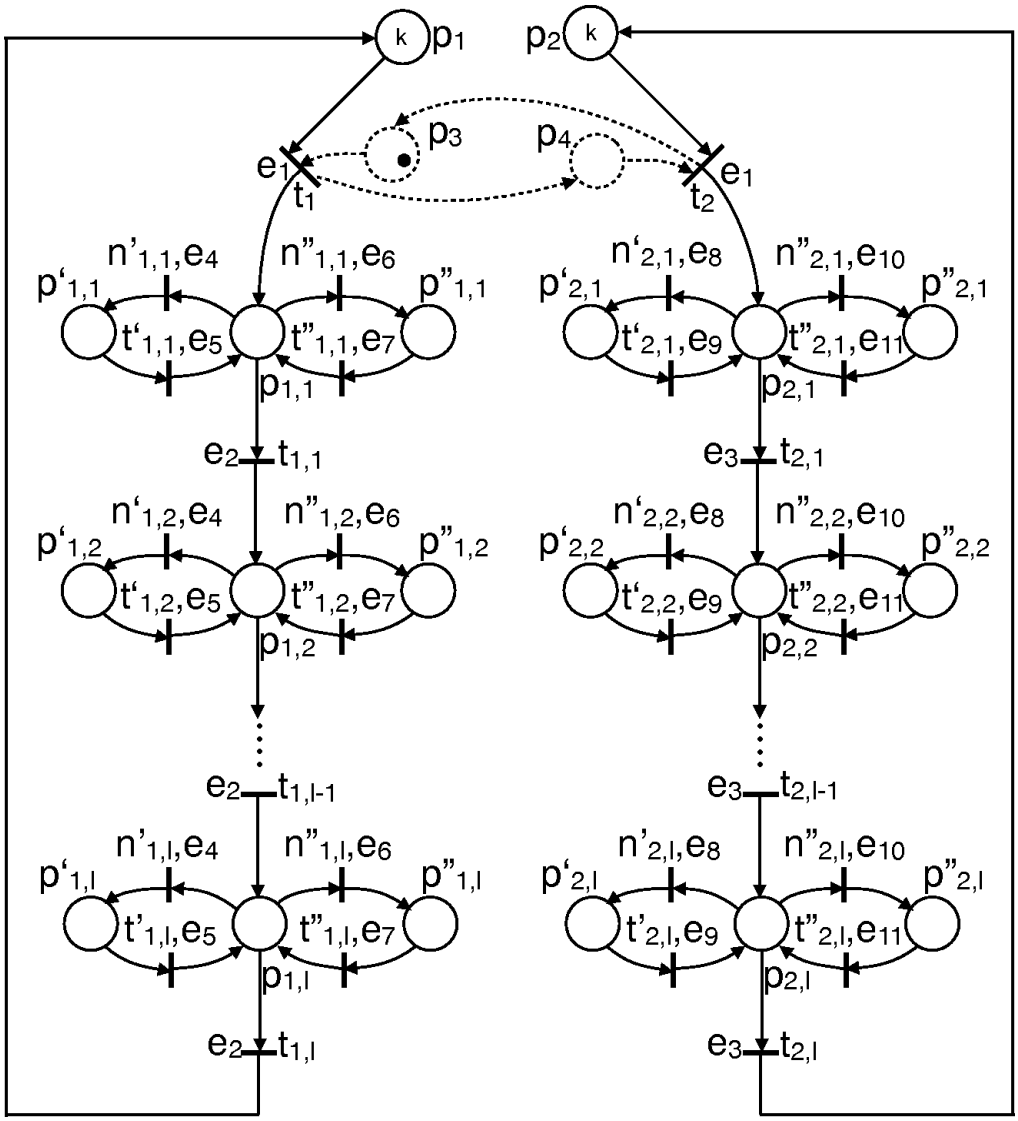


Figure 3.10: Petri net model of the actuators of the manufacturing system in Figure 3.9

Results obtained with using the STS, MRG and the RG approach are shown in Table 3.10, where required time ($\mathcal{T}_{STS}$, $\mathcal{T}_{MRG}$ and $\mathcal{T}_{RG}$) of the SS are summarized for different values of m and l. Note that here only one simulation per setting has been performed.

The table shows the cardinality of the reachability graph ($|\mathcal{G}|$) and of the auxiliary graph ($|\mathcal{A}(\mathcal{G})|$) for the RG approach. Note that Proposition 3.30 applies to the synchronized PN in Figure 3.10. Thus, the MRG and STS approach work on the underlying SM subnets. As a consequence, the MRG approach constructs a different RG $\mathcal{G}_M$ and a different modified auxiliary graph $\mathcal{A}(\tilde{\mathcal{G}}_M)$, whose cardinalities do not correspond. These are important parameters to understand the limits of the RG approach, while exhaustively enumerating the set space of the net.

Note that the table shows also non-numerical values where the corresponding result cannot be provided:

		RG			MRG			STS
k	l	$\|\mathcal{G}\|$	$\|\mathcal{A}(\tilde{\mathcal{G}})\|$	$\hat{\mathcal{T}}_{RG}[s]$	$\|\mathcal{G}_M\|$	$\|\mathcal{A}(\tilde{\mathcal{G}}_M)\|$	$\hat{\mathcal{T}}_{MRG}[s]$	$\hat{\mathcal{T}}_{STS}[s]$
1	1	32	528	3.43	4	10	0.42	0.26
1	2	98	4851	*o.t.*	7	28	0.46	0.43
1	3	200	20100	*o.t.*	10	55	0.92	2.81
1	4	338	57291	*o.t.*	13	91	1.57	31.13
2	1	200	20100	*o.t.*	4	10	0.42	0.26
2	2	1568	*o.t.*	*n.c.*	7	28	0.46	0.43
2	3	*o.t.*	*n.c.*	*n.c.*	10	55	0.92	2.81
2	4	*o.t.*	*n.c.*	*n.c.*	13	91	1.57	31.13
3	1	800	320400	*o.t.*	4	10	0.42	0.26
3	2	*o.t.*	*n.c.*	*n.c.*	7	28	0.46	0.43
3	3	*o.t.*	*n.c.*	*n.c.*	10	55	0.92	2.81
3	4	*o.t.*	*n.c.*	*n.c.*	13	91	1.57	31.13

Table 3.10: Time results for a manufacturing system.

i) *out of time* (o.t.), when the corresponding value has not been computed within 6 hours;

ii) *not computable* (n.c.), if the corresponding value cannot be computed: e.g., the RG is o.t. and the corresponding auxiliary graph cannot be evaluated.

The table denotes, for an increasing number of tokens, that the RG approach goes almost always o.t., due to a significant larger space state. On the contrary, the required time does not change with the STS and MRG approach, that always find a solution.

Chapter 4

Synchronizing sequences on unbounded Petri nets

Summary

In this chapter the problem of boundedness for the class of synchronized PNs is investigated.

First, a modified coverability graph construction is provided to yield a faithful representation of the PN behavior.

Then the RG approach given for the bounded case is further extended to this setting. The proposed approach suffers from the fact that no finite space representation can exhaustively answer to the reachability problem. However it is shown that this shortcoming can be solved for some particular semantics.

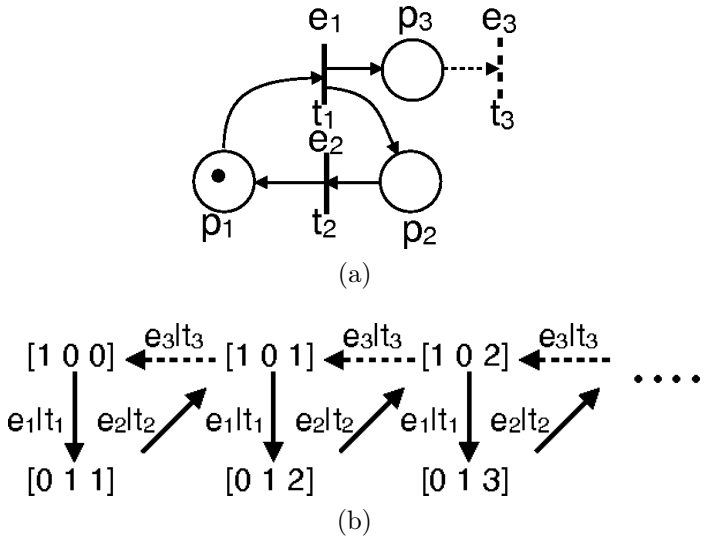


Figure 4.1: A synchronized PN (a) and the partial construction of its RG (b).

4.1 Problem statement

The same setting provided for bounded synchronized PNs can be extended with some minor changes to the class of unbounded PNs, i.e., nets with an infinite state space.

The next example shows what kind of problems may encountered while constructing a SS on unbounded synchronized PNs.

Example 4.1. *Consider the PN in Figure 4.1a (without the dashed transition t_3) with a initial marking $[1\,0\,0]^T$ and its infinite reachability graph in Figure 4.1b (without the dashed arcs labeled $e_3|t_3$). Here the set of bounded places and the set of unbounded places are respectively $P_b = \{p_1, p_2\}$ and $P_u = \{p_3\}$. The RG does not have an ergodic component, because all the nodes are transient components: such a case cannot occur in bounded nets. As previously said, there exists a SS only for those markings belonging the ergodic component, so the absence of an ergodic component implies that no SS exists regardless of the chosen target marking.*

Consider now the unbounded PN in Figure 4.1a (including dashed transition t_3) with an initial marking $[1\,0\,0]^T$ and its infinite reachability graph in Figure 4.1b (including the dashed arcs labeled $e_3|t_3$). Even in this case $P_b = \{p_1, p_2\}$ and $P_u = \{p_3\}$. Suppose we want to reach the target marking $\bar{M} = [1\,0\,0]^T$. For such a marked net, it holds that $\mathcal{M}_0 = R(N, M_0) = \{M \in \mathbb{N}^3 : M(p_1) + M(p_2) = 1, M(p_3) = k \in \mathbb{N}\}$. Obviously, the input sequence $w = e_2\{e_3\}^k$ drives the net to $\bar{M}$ from any marking $M = [1\,0\,u]^T$ and $M = [0\,1\,u]^T$ with $u \leq k$. However, since u can be arbitrarily large, properly speaking no SS to $\bar{M}$ exists for this net.

Finally, note that in both previously discussed cases it is always possible to reach a marking where the token content of places p_1 and p_2 is known. In fact, from any reachable marking the input sequence $w = e_2$ drives the net to $M(p_1) = 1$ and $M(p_2) = 0$. ■

The previous example shows that in an unbounded net one cannot find a SS that leads from any reachable marking to a marking where the token content of an unbounded place is known. In fact, the ω symbol, used to keep the graph finite, entails loss of information in terms of reachable markings and of firing sequences.

Note that this concept of partial synchronization of the state has been suggested in the context of sequential machines by Cheng *et al.* [22], where if a SS is prohibitively long or non existent, a selected subset of all flip-flops could be electrically reset.

This motivates the following extended definition of SS that only takes into account the set of bounded places P_b.

Definition 4.2. ***(SS on unbounded PNs)*** *Given an unbounded synchronized PN $\langle N, E, f\rangle$ and a starting marking M_0, $\bar{w}$ is called a SS for a desired marking $\bar{M}$ iff the net is driven, starting from any marking $M \in R(N, M_0)$, to a marking $\bar{M}'$ such that $\bar{M}' \uparrow_b = \bar{M} \uparrow_b$.* ■

Note that $\bar{M} \uparrow_b$ is the projection of the marking $\bar{M}$ onto the set of bounded places P_b.

The set of all SSs for a given marking $\bar{M}$ is denoted $\mathcal{SS}(N, M_0, \bar{M})$.

Thus, because of Definition 4.2, the synchronization process aims to guide the net from an unknown marking to one of the markings that corresponds to the desired one in terms of bounded places. Clearly the target turns from a target marking into a target set of markings.

Definition 4.3. ***(Synchronization target marking set)*** *Consider an unbounded synchronized PN $N = \langle N, E, f\rangle$ and a starting marking M_0. Given a desired marking $\bar{M}$, one defines the target marking set as $\bar{\mathcal{M}} = \{M : M \uparrow_b = \bar{M} \uparrow_b\}$.* ■

4.2 A modified coverability graph

In this section a *modified coverability graph* (MCG) construction is provided to yield a faithful representation of the unbounded PN behavior. Afterwards it is shown that this algorithmic construction may present some shortcomings that can be solved for some particular semantics.

4.2.1 An algorithmic construction of the MCG

In synchronized PNs, the occurrence of any input event sequence w at some marking M may cause the firing of a (set of) transition(s). From any marking $M' \geqneq M$, the same input event application can yield another evolution, i.e., a larger set of firing transitions or simply a disjoint one. In fact, obtaining an increased marking after the application of an input sequence is only a necessary condition, due to the non-necessarily monotonic evolution of the net. Such a condition must hold for every further occurrence of w.

The following definition gives a procedure to determine the increasing sequences presented in Definition 2.22.

Definition 4.4. ***(increasing input sequences)*** *Consider a synchronized PN $\langle N, E, f\rangle$ and let the current marking be M. Let M' and M'' be respectively the marking reached after a first and a second occurrence of sequence w, i.e., $M[w|\sigma\rangle M'[w|\sigma'\rangle M''$. Sequence w is called increasing input sequence if the following three conditions hold:*

(Test for increasing sequences)

C1) $\pi(\sigma) = \pi(\sigma')$;

C2) $M \lneqq M' \lneqq M''$;

C3) $\forall p : M'(p) > M(p)$ *and* $\forall t \in p^\bullet$, $M'(p) \geq Pre(p,t)$. ■

We recall that, given sequence $\sigma \in T^*$, $\pi(\sigma)$ is its *firing vector* as defined in section 2.1.2.

An algorithmic construction of the *modified coverability tree* (MCT) can now be given.

Algorithm 4.5. ***(MCT construction for synchronized PNs)***
Input: *a deterministic marked synchronized PN $\langle N, M_0, E, f\rangle$.*
Output: *a MCT $\mathcal{T}$.*

1. *Label the root node q_0 with the initial marking M_0 and tag it "new".*

2. ***While*** *a node tagged "new" exists,* ***do***

2.1. *Select a node q tagged "new".*

2.2. *Let M be the label of q.*

2.3. ***For*** *all $e \in E$:*

2.3.1. *Let $M' = M + \sum_{t \in \mathcal{E}_e(M)} (Post(\cdot,t) - Pre(\cdot,t))$ be the marking reached by firing all enabled $t \in T_e$.*

2.3.2. *Let $\hat{Q}$ be the set of nodes met on a backward path from q to q_0 whose label is $\hat{M} \lneqq M'$.*

2.3.3. ***For*** *all nodes $\hat{q} \in \hat{Q}$ labeled $\hat{M}$,*

2.3.3.1. *let w and σ be the input sequence and the corresponding firing sequence s.t. $\hat{M}[w|\sigma\rangle M'$.*

2.3.3.2. *Let apply again w from M', obtaining $M'[w|\sigma'\rangle M''$.*

2.3.3.3. ***If*** *the three following conditions hold:*

C1) $\pi(\sigma) = \pi(\sigma')$;

C2) $\hat{M} \lneqq M' \lneqq M''$;

C3) $\forall p : M'(p) > \hat{M}(p)$ *and* $\forall t \in p^\bullet$, $M'(p) \geq Pre(p,t)$;

then *let $M'(p) = \omega$.*

End for

2.3.4. *Add a new node q' and label it M'.*

2.3.5. *Add an arc labeled $e|\{\mathcal{E}_e(M)\}$ from q to q'.*

2.3.6. ***If*** *there exists already in the tree a node with label M',* ***then*** *tag node q' "duplicate",* ***else*** *tag it "new".*

2.4. *Untag node q.*

End while ■

Analogously to the RT construction of Algorithm 2.18, the MCG is obtained from the MCT fusing duplicate nodes with the untagged node with the same label.

Note that, this algorithm can also be used to compute the RT (and then the RG) skipping from step 2.2.2 to step 2.2.3.

Proposition 4.6. *Consider a marked unbounded synchronized PN $\langle N, M_0, E, f\rangle$ and its MCG $\mathcal{G}$. There exists a ω−marking M_ω in $\mathcal{G}$ such that $M_\omega(p) = \omega$ iff $p \in P_u$.*

Proof. Consider Definition 2.22 for increasing sequences. Condition C1) and condition C2) ensure that sequence w leads to a greater marking by the firing of the same transition sequence. For increasing sequences those two requirements must hold while applying w not once but an infinite number of times. Condition C3) guarantees exactly this monotonicity of the firing rule. In fact, while arbitrarily incrementing the marking of every place p such that $M'(p) > M(p)$, if C3) holds, sequence σ will still not change. □

The boundedness of the MCG is proven by the following proposition.

Proposition 4.7. *Consider a marked unbounded synchronized PN $\langle N, M_0, E, f\rangle$. Its MCG constructed via Algorithm 4.5 is a finite graph.*

Proof. In the framework of P/T nets, Karp and Miller [69] have given an algorithmic construction of a CT. Their algorithm is proven to compute the tree in a finite number of steps — i.e., the constructed tree has a finite number of vertices — even if the net has an infinite reachability set. The boundedness of the CT is guaranteed by the ω-symbol, which replaces the marking of a place whenever an increasing sequence is found. These increasing sequences, which in the RT represent paths of infinite length, are depicted in the CT as self-loops over ω-markings.

The very same proof can be given in the framework of synchronized PNs, since the Karp-Miller algorithm [69] and Algorithm 4.5 coincide except in the computation of increasing sequences (see Definition 2.22). In fact, reaching an increased marking after the occurrence of an input sequence — see condition **C2)** — is only a necessary condition for synchronized PNs. Additional conditions **C1)** and **C3)** introduce a finite number of markings before being satisfied, so that the identification of any increasing input sequence is only postponed. □

An example of MCG can be found in Figure 4.2, where Algorithm 4.5 is applied to the unbounded synchronized PN of Figure 4.1a.

Note that this is a simple way to construct the coverability marking set and more efficient techniques could be investigated as in the case of P/T nets for minimal coverability sets.

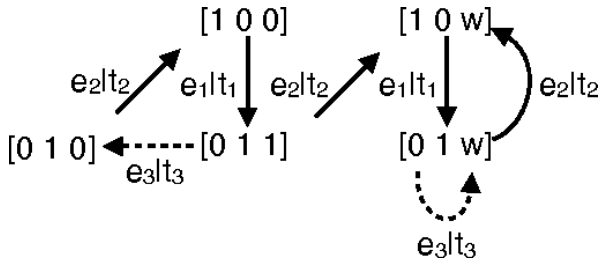


Figure 4.2: The MCG of the synchronized PN of Figure 4.1a.

However, to the best of our knowledge, no algorithmic construction of such a set exists so far.

4.2.2 Vanishing markings and vanishing sequences

The ω–symbol allows to have a set of ω–markings M_ω covering an infinite set of finite markings. It is clear that, for a given input event occurring at ω–marking M_ω, not all $M \in cov(M_\omega)$ enable the same firing sequence. Different firing sequences may lead then to markings that are not represented in the MCG. Thus, the so-constructed coverability marking set may not be, as for the case of P/T nets, a superset of the reachability set.

Definition 4.8. ***(Vanishing markings)*** *Consider an unbounded marked synchronized PN $\langle N, M_0, E, f\rangle$. For a MCG $\mathcal{G}$, constructed by the way of Algorithm 4.5, there may exist some marking M_v such that: i) $M_v \in R(N, M_0)$; ii) $\nexists M \in \mathcal{G} : M = M_v \wedge M \geq_\omega M_v$. M_v is called a* vanishing markings. ■

Vanishing markings are markings reachable only by the firing of vanishing sequences. As it could be easily seen in the following example, such sequences are fireable only from markings that have "not enough" tokens in the unbounded places.

Example 4.9. *Consider the PN in Figure 4.3a. Its partial RG construction and its MCG are respectively depicted in Figure 4.3b and in Figure 4.3c. Consider marking $M_1 = [1\,1\,1\,1\,0]^T$ of the RG, which is covered by ω–marking $M_1' = [\omega\,\omega\,1\,1\,0]^T$ of the MCG. Input event e_2 drives the net to vanishing marking $M_2 = [1\,0\,1\,0\,1]^T$ and the MCG to ω–marking $M_2'[\omega\,\omega\,0\,0\,2]^T$. M_2 does not cover M_2 because the marking of place p_2 is not enough to enable t_2, preventing it to fire.* ■

Vanishing markings cannot obviously be taken as a target for any SS, as shown in the following proposition.

Proposition 4.10. *Consider an unbounded synchronized PN $\langle N, E, f\rangle$. There exists no SS for a vanishing marking.*

Proof. The proposition is proved by *reductio ad absurdum*. Suppose there exists a SS of finite length for vanishing marking M_v. If this was possible, then one would have a sequence that, no matter the contents of the unbounded places, synchronizes the net

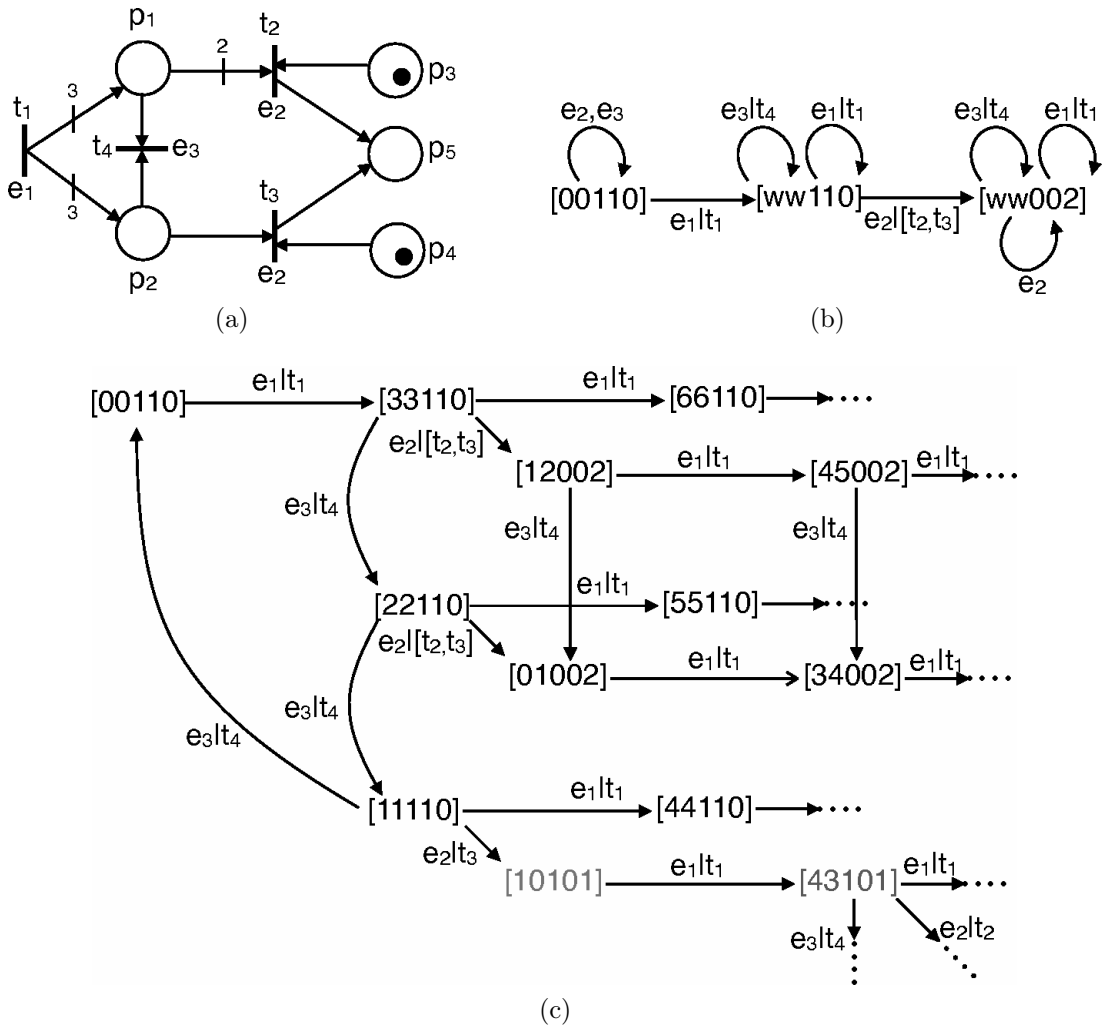


Figure 4.3: A synchronized PN with vanishing markings (a), its MCG (b) and RG (c).

to M_v. Obviously this is contradictory to Definition 4.8 and the supposition is thereby refuted. □

The proposed MCG lacks in representing vanishing markings. In fact, all transitions firing at an ω-marking may not be fired in the synchronized PN because the marking of unbounded places does not enable all of them.

This obviously does not occur in the case of totally synchronized PNs, i.e., nets where each event is associated with a different single transition, as defined in Section 2.1.2.

In the rest of the chapter, a less restrictive assumption is made so that a SS can be constructed for the case of unbounded nets.

Assumption 4.11. *Consider a synchronized PN* $\langle N, E, f \rangle$ *and let* $P_u \subset P$ *the set of its unbounded places. Every transition* $t \in P_u^\bullet$ *is such that* $\nexists t' \in T : t \neq t' \wedge f(t) = f(t')$.

The previous assumption ensures that any transition outputting an unbounded place is

associated with an input event which is not shared with any other transition.

Proposition 4.12. *There is no vanishing marking for unbounded synchronized PNs satisfying Assumption 4.11.*

Proof. Consider an unbounded synchronized PN $\langle N, E, f\rangle$ and its MCG $\mathcal{G}$, constructed by Algorithm 4.5. Consider some marking M'_ω of $\mathcal{G}$, reached by applying input event e from marking M_ω, i.e., $M_\omega[e|\sigma\rangle M'_\omega$. Assume that the current marking of the PN is any $M \in cov(M_\omega)$. The occurrence of event e will lead the net to a vanishing marking $M' \notin cov(M'_\omega)$, i.e., $M[e|\sigma'\rangle M'$, if the following condition holds: $(\exists p \in P_u, \exists t \in \sigma) \quad M(p) \ngeq Pre(p,t)$. In other words, vanishing marking M' is reached from M if while event e occurs any of the unbounded places is not sufficiently marked and a different firing sequence σ' is obtained, such that $\pi(\sigma') < \pi(\sigma)$.

If $\langle N, E, f\rangle$ satisfies Assumption 4.11, then event e will be associated with one transition so that either $\sigma' = \varepsilon$ or $\sigma' = \sigma$. Thus, the net cannot reach any vanishing marking. □

4.3 Computation of synchronizing sequences using the MCG

The RG approach (see Section 3.1.1) has been provided as an adaptation of the greedy algorithm [76] for SS construction in the case of bounded synchronized PNs.

The same algorithm can be easily thought as applicable to the class of unbounded PNs, provided, of course, a finite representation of its reachability set, i.e., an automaton whose behavior is equivalent to the unbounded net itself.

The developed approach for bounded nets — based on the analysis of the RG — can now be extended to unbounded nets satisfying Assumption 4.11 and whose behavior is represented by their MCG.

4.3.1 Potentially synchronizing sequences

The objective of this section is first to compute a SS for the MCG of a given net and then use this sequence to compute a SS for the net itself.

Graph-based approaches require a given graph $\mathcal{G}$ to be completely specified, condition not always true for a PN. In fact, from a marking not all transitions are necessarily enabled.

So, as in the bounded case, it is necessary to turn the MCG into a completely specified MCG, denoted $\tilde{\mathcal{G}}$. The new graph is obtained by applying Algorithm 3.2 to $\mathcal{G}$.

The SS construction could be done following these steps: i) computation of the MCG $\mathcal{G}$ of a given net; ii) computation of the completely specified MCG $\tilde{\mathcal{G}}$; iii) computation of the auxiliary graph $\mathcal{A}(\tilde{\mathcal{G}})$; iv) computation of a SS for $\tilde{\mathcal{G}}$; v) computation of a SS for the original net.

Steps i)-iii) have been discussed in previous sections so in the following we will focus on the last two steps of the procedure.

A sequence that synchronizes the MCG to a node corresponding to the desired marking is called a *potentially synchronizing sequence* (PSS).

The set of all PSSs for a given MCG is defined as follows.

Definition 4.13. ***(Set of PSS)*** *Given a MCG $\mathcal{G}$ of a synchronized PN constructed over a starting marking M_0, one defines the set of all PSSs for a target marking $\bar{M}$ as*

$$\mathcal{PSS}(\mathcal{G}, \bar{M}) = \{w : \forall M \in \mathcal{G},\ M \xrightarrow{w} \bar{M}' \wedge \bar{M}' \uparrow_b = \bar{M} \uparrow_b\}. \qquad \blacksquare$$

The following algorithm allows one to construct a PSS $\bar{w}$, which is not necessarily the shortest one. Such a sequence leads the MCG to a marking $M \in \bar{\mathcal{M}}$.

Algorithm 4.14. ***(Computing a PSS for a marking $\bar{M}$)***
Input: *An unbounded synchronized PN $N = \langle N, E, f\rangle$ satisfying Assumption 4.11, a starting marking M_0 and a bounded target marking $\bar{M}$.*
Ouput: *A PSS $\bar{w}$.*

1. *Let $\mathcal{G}$ and $\mathcal{A}(\tilde{\mathcal{G}})$ be respectively the MCG and the auxiliary graph of the completely specified MCG.*
2. *Let $\bar{\mathcal{M}}$ be the set of target markings for a given $\bar{M}$.*
3. *Let $\bar{w} = \varepsilon$, the empty initial input sequence.*
4. *Let $\phi(\bar{w}) = \{M : M \in V\}$, the initial current marking uncertainty.*
5. ***While*** *$\phi(\bar{w}) \notin \bar{\mathcal{M}}$,* ***do***
   - 5.1. *pick two markings $M_i, M_j \in \phi(\bar{w})$ such that the two following conditions hold: i) $M_i \neq M_j$, ii) $M_i \notin \bar{\mathcal{M}}$ or $M_i \notin \bar{\mathcal{M}}$ or $M_i, M_j \notin \bar{\mathcal{M}}$;*
   - 5.2. *find a shortest path in $\mathcal{A}(\tilde{\mathcal{G}})$ from (M_i, M_j) to $(\bar{M}', \bar{M}'')$, where $\bar{M}', \bar{M}'' \in \bar{\mathcal{M}}$.*
   - 5.3. ***If*** *no such a path exists, stop the computation, there exists no PSS for $\bar{M}$.*

     Else*, let w be the input sequence along this path,* ***do***
     - 5.3.1. *$\phi(\bar{w}w) = \{M' : \forall M \in \phi(\bar{w}), M \xrightarrow{w} M'\}$;*
     - 5.3.2. *$\bar{w} = \bar{w}w$.*

     end if

   end while $\blacksquare$

Note that the above algorithm performs a greedy-computation (see Algorithm 2.32) on the MCG of the given synchronized PN with two important differences. First, its stop criterium at step 5 is changed. In fact, the current state uncertainty is not required to be singleton, but included in the synchronization target marking set, according to Definition 4.2. Second, consider step 5.1. At this point two markings M_i and M_j of the

M_0	$[1\,0\,0]^T$
M_1	$[1\,1\,0]^T$
M_2	$[1\,\omega\,0]^T$
M_3	$[0\,\omega\,1]^T$

Table 4.1: Markings of the synchronized PN in Figure 4.4a

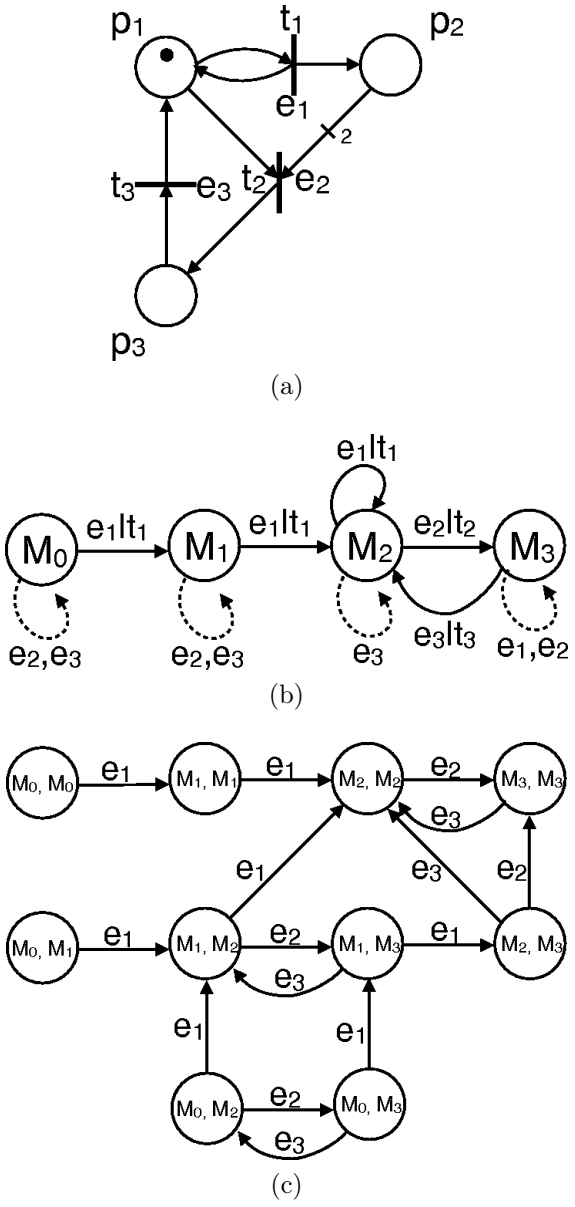


Figure 4.4: An unbounded synchronized PN (a), its completely specified MCG $\mathcal{G}$ (b) and auxiliary graph $\mathcal{A}(\tilde{\mathcal{G}})$ (c).

current state uncertainty are selected to be synchronized to the target by the application of input sequence w. Note that the two markings have not to simultaneously belong to the target marking set, accordingly to conditions i) and ii), otherwise the current state

uncertainty would not change. At step 5.2, since the target is no longer a marking but a set of markings, the algorithm searches the shortest path in the auxiliary graph from node (M_i, M_j) to a node $(\bar{M}', \bar{M}'')$, where $\bar{M}'$ and $\bar{M}''$ are not necessarily different and belong to the set of target markings.

Example 4.15. *Consider the marked synchronized PN in Figure 4.4a. Let* $P_u = \{p_2\}$ *and* $P_u^\bullet = \{t_2\}$*, where* $f(t_2) = e_2$*. The net clearly satisfies Assumption 4.11, since no other transition is associated with* e_2*. The reachable markings are provided by Table 4.1. Let* $\bar{M} = [0\,3\,1]^T$ *and then* $\bar{\mathcal{M}} = \{M_3\}$*. A possible execution of Algorithm 4.14 is described by the following steps. Let the initial marking uncertainty be* $\phi(\varepsilon) = \{M_0, M_1, M_2, M_3\}$*. If at step 5.1 markings* M_0 *and* M_1 *are selected, path* $(M_0, M_1) \xrightarrow{e_1 e_1 e_2} (M_3, M_3)$ *will be obtained. The corresponding current state uncertainty is updated to* $\phi(e_1 e_1 e_2) = \{M_3\}$*, so that the computation ends in finding* $w = e_1 e_1 e_2$ *as the searched PSS.* ■

4.3.2 Validating marking estimate

Each node of the MCG represents either one of the following: i) a marking M of the net, in such a case outgoing node transitions correspond to all transitions enabled on the net from that marking, i.e., $\mathcal{E}(M)$; ii) an ω–marking M_ω, covering an infinite set of markings of the net; in such a case all outgoing node transitions characterize the union of all sets of transitions enabled on the net from those markings, i.e., $\bigcup_{M \in cov(M_\omega)} \mathcal{E}(M)$.

On the basis of the above remark, it can be stated that any input sequence w synchronizing a PN — satisfying Assumption 4.11 — to a target marking $\bar{M}$, it also synchronizes the MCG to a marking $\bar{M}'$ such that $\bar{M} \in cov(\bar{M}')$. The opposite is not true, i.e., not every PSS is a SS.

However, for any PSS $\bar{w}$ there exists a minimal enabling marking M, i.e., if the current marking M' is greater than or equal to M, the occurrence of $\bar{w}$ at M' will drive the net to a marking in the target set.

Definition 4.16. ***(Validating marking set)*** *Consider an unbounded synchronized PN and its starting marking* M_0*, from which one constructs the MCG* $\mathcal{G}$*. It is given an input sequence* $\bar{w}$*, synchronizing* $\mathcal{G}$ *to the target* $\bar{M}$*. The set of all markings validating* $\bar{w}$*, denoted* $\mathcal{M}(\bar{w})$*, is the set of all markings* M *such that* $M \xrightarrow{w} \bar{M}'$ *and* $\bar{M}' \uparrow_b = \bar{M} \uparrow_b$ ■

If the initial marking uncertainty belongs to this set, an input sequence is clearly a SS.

4.3.2.1 Initial marking estimate

For any input event sequence computed by the way of the auxiliary graph one easily determines the underlying transition sequence, since the markings are known step by step.

Consider Algorithm 4.14, at each iteration i of step 5, a path $w_i = e_{i1}e_{i2}\dots e_{il}$ from node (M_{i1}, M_{i1}) to $(\bar{M}', \bar{M}'')$ is selected. Such a sequence identifies an interleaving of pairs of markings and input events as follows:

$$(M_{i1}, M'_{i1}) \xrightarrow{e_{i1}} (M_{i2}, M'_{i2}) \xrightarrow{e_{i2}} \dots \xrightarrow{e_{il}} (\bar{M}', \bar{M}''). \tag{4.1}$$

Consider the j–th event of the found sequence. Since the auxiliary graph has unordered pairs of markings, to compute the underlying transition one has to determine which of the following statements is true:

a) $M_{ij} \xrightarrow{e_{ij}} M_{ij+1}$ and $M'_{ij} \xrightarrow{e_{ij}} M'_{ij+1}$;

b) $M_{ij} \xrightarrow{e_{ij}} M'_{ij+1}$ and $M'_{ij} \xrightarrow{e_{ij}} M_{ij+1}$.

This uncertainty can be clearly solved by the MCG itself, being a deterministic oriented graph. Given the path w_i of Equation 4.1, the two underlying firing transition sequences can be easily determined as follows:

$$\sigma_i, \sigma'_i : \begin{cases} M_{i1}[\sigma_{i1}\rangle M_{i2}[\sigma_{i2}\rangle \dots M_{il}[\sigma_{il}\rangle \bar{M}' \\ M'_{i1}[\sigma'_{i1}\rangle M'_{i2}[\sigma'_{i2}\rangle \dots M'_{il}[\sigma'_{il}\rangle \bar{M}'', \end{cases} \tag{4.2}$$

where $\sigma_{ij} \in \Sigma_e(M_{ij})$, $\sigma'_{ij} \in \Sigma_e(M'_{ij})$, $\sigma_i = \sigma_{i1}\sigma_{i2}\dots\sigma_{il}$ and $\sigma'_i = \sigma'_{i1}\sigma'_{i2}\dots\sigma'_{il}$. We remind that all sequences in $\Sigma_e(M_{ij})$ are possible firing sequences at marking M_{ij} after the occurrence of event e (see Definition 2.12).

Obtained the underlying firing sequences, the initial marking estimate can be computed by using the underlying P/T net, i.e., the corresponding non-labeled net. The following algorithm, taken from [56], recursively calculates the minimal initial marking for a given sequence of transition.

Algorithm 4.17. ***(Computing minimal marking estimate)***
Input*: a PN $N = (P, T, Pre, Post)$, a marking M and a firing sequence σ.*
Output*: the minimal enabling marking $\bar{M} = \bar{M}(M, \sigma)$, s.t. $\bar{M}[\sigma\rangle \wedge (\forall p \in P)\, \bar{M}(p) \geq M(p)$.*

1. *Let $\bar{M}^0 = M$.*
2. *Let $\sigma = t_{i1}t_{i2}\dots t_{il}$.*
3. ***For*** *$j = 1, 2 \dots l$,* ***do***

   3.1. $\bar{M}^j = max\{\bar{M}^{j-1} + C \cdot y^j, Pre(\cdot, t_{ij})\} - C \cdot y^j$

   End for. ■

Assume $\sigma = t_{i1}t_{i2}\dots t_{il}$. The firing vector of the prefix sequence $t_{i1}t_{i2}\dots t_{ij}$ is denoted y^j, i.e., $y^j = \pi(t_{i1}t_{i2}\dots t_{ij})$.

M_0^{j-1} (resp. M_0^j) is the initial marking estimate before (resp. after) t_{ij} fires. Such an estimate is obtained by adding the minimum number of tokens to places, such that the prefix of σ_i considered so far is enabled.

Given a transition firing sequence σ, it is not difficult to see that the minimal initial marking $\bar{M}$ is unique. That marking is recursively obtained, when one initially sets $\bar{M}^0 = \vec{0}_{m\times 1}$.

Example 4.18. *Consider the PN in Figure 4.4a. It is given the firing sequence $\sigma = t_3t_2t_1$, for which one wants to construct the minimal marking that enables it. By the direct application of Algorithm 4.17 one obtains the following recursive updates:*

i) $\bar{M}_0^0 = \vec{0}_{3\times 1} = [0\,0\,0]^T$

ii) $\bar{M}_0^1 = max\{[0\,0\,0]^T + [0\,0\,0]^T, [0\,0\,1]^T\} - [0\,0\,0]^T = [0\,0\,1]^T$

iii) $\bar{M}_0^2 = max\{[0\,0\,1]^T + [1\,0\,-1]^T, [1\,2\,0]^T\} - [1\,0\,-1]^T = [0\,2\,1]^T$

iv) $\bar{M}_0^3 = max\{[0\,2\,1]^T + [0\,-2\,0]^T, [1\,0\,0]^T\} - [0\,-2\,0]^T = [1\,2\,1]^T$ ■

For any other initial marking M, Algorithm 4.17 verifies if M enables the given sequence σ and increases the marking of places that otherwise would not allow the firing of σ, obtaining $\bar{M}$.

Example 4.19. *Consider the MCG of the PN in Figure 4.4a and ω−marking $M_2 = [1\,\omega\,0]^T$. The minimal marking of the unique unbounded place p_2 that enables t_2 is given by the direct application of Algorithm 4.17. The following recursive updates are obtained:*

i) *Let* $M_2'(p) = \begin{cases} M_1(p) & if\ p \notin P_u \\ 0 & otherwise \end{cases}$. *Then* $\bar{M}_2^0 = M_2' = [1\,0\,0]^T$.

ii) $\bar{M}_2^1 = max\{[1\,0\,0]^T + [0\,0\,0]^T, [1\,2\,0]^T\} - [0\,0\,0]^T = [1\,2\,0]^T$. ■

4.3.2.2 Set of markings validating a PSS

A procedure to construct the set of markings validating a PSS is now proposed. It is shown that such a set belongs to the class of semi-cylindrical sets, whose definition is recalled in the following.

Definition 4.20. *[49] **(Cylindrical sets)** Given a subset $I \subset \{1,\dots,m\}$ and a vector $v \in \mathbb{N}^m$, the cylinder of basis (I,v) is the subset $C(I,v) = \{x \in \mathbb{N}^m : \forall i \in I,\ x_i = v_i \wedge \forall i \notin I,\ x_i \geq v_i\}$. One denotes by $SCyl(\mathbb{N}^m)$ the set of finite unions of cylinders, that are called semicylindrical subsets.* ■

Assume that Algorithm 4.14 has output a PSS $\bar{w}$. By the way of the MCG $\mathcal{G}$, the set of underlying transition sequences can be derived. Each sequence σ_i is associated with a path from any of the marking $M_i \in \mathcal{G}$ to some $\bar{M} \in \bar{\mathcal{M}}$.

The set of validating markings $\mathcal{M}(\bar{w})$ is constructed so that those underlying firing sequences are fireable. It is shown that it depends on the considered PSS and that is a subset of the covering set $CS(N, M_0)$.

Algorithm 4.21. ***(Validating marking set construction)***
Input: *An unbounded synchronized PN $\langle N, M_0, E, f\rangle$, its MCG $\mathcal{G}$ and a PSS $\bar{w}$ for a bounded target marking $\bar{M}$.*
Ouput: *the set $\mathcal{M}(\bar{w})$ of markings validating $\bar{w}$.*

1. *Let $|\mathcal{G}| = n$.*
2. *let $\mathcal{M}(\bar{w}) = \varnothing$.*
3. ***For*** *$i = 1, \ldots, n$, do*
   - *3.1. Let σ_i be the underlying firing sequence of $\bar{w}$, s.t.*
     - *3.1.1. $\bar{M} \in cov(M_{\omega,i})$;*
     - *3.1.2. σ_i is the path along $\mathcal{G}$, i.e., $M_i \xrightarrow{\bar{w}|\sigma_i} M_{\omega,i}$.*
   - *3.2. Let M_i' be s.t. $M_i'(p) = \begin{cases} M_i(p) \; if \; p \in P_b \\ 0 \qquad otherwise \end{cases}$*
   - *3.3. Let $\bar{M}_i = \bar{M}_i(M_i', \sigma_i)$, determined by Algorithm 4.17.*
   - *3.4. Let $\mathcal{M}(\bar{w}) = \mathcal{M}(\bar{w}) \bigcup C(I_b, \bar{M}_i)$.*

   End for ■

A possible execution of the above algorithm is given by the following example.

Example 4.22. *Consider the PSS $\bar{w} = e_1e_1e_2$ computed in Example 4.15. By the aid of the MCG, one finds that $\sigma_0 = \sigma_1 = \sigma_2 = t_1t_1t_2$, and $\sigma_3 = \varepsilon$. Let $M_0' = M_1' = M_2' = [1\,0\,0]^T$ and $M_3' = [0\,0\,1]^T$. By applying Algorithm 4.17 the following marking estimate are computed: $\bar{M}_0'(M_0', t_1t_1t_2) = \bar{M}_1'(M_1', t_1t_1t_2) = \bar{M}_2'(M_2', t_1t_1t_2) = [1\,0\,0]^T$ and $\bar{M}_3'(M_3', \varepsilon) = [0\,0\,1]^T$. Thus the set of markings for which w is SS is $\mathcal{M}(\bar{w}) = C(\{1,3\}, \bar{M}_0') \bigcup C(\{1,3\}, \bar{M}_3')$.* ■

The following theorem proves the correctness of the above algorithm.

Theorem 4.23. *Consider an unbounded synchronized PN $\langle N, E, f\rangle$ satisfying Assumption 4.11. Let $\mathcal{G}$ be its MCG for a given starting marking M_0. Let σ be the sequence in $\mathcal{G}$ along the path between markings M_ω and M_ω'. For all $M \in cov(M_\omega)$ s.t. $M \uparrow_u \geq \bar{M}_\omega(M_\omega, \sigma) \uparrow_u$ it holds that $M[\sigma\rangle M'$, where $M' \in cov(M_\omega')$.*

Proof. Increasing the marking of a place can change the set of enabled transitions and then, after the same input event application, the reached marking. Synchronized PNs

satisfying Assumption 4.11 have 1–to–1 mapping between transitions outputting an unbounded place and input events. Then, once sufficiently marked unbounded places, none but the expected transitions fire. □

An important result can be deduced by only looking at the set $\mathcal{M}(\bar{w})$.

Proposition 4.24. *Consider an unbounded synchronized PN $\langle N, E, f\rangle$ satisfying Assumption 4.11 and a starting marking M_0. It is given a PSS $\bar{w}$ for a target marking $\bar{M}$ for which the validating marking set $\mathcal{M}(\bar{w})$ is computed. If for every base $\bar{M}_i$ of $\mathcal{M}(\bar{w})$ it holds that $\bar{M}_i \uparrow_u = \vec{0}_{m_u}$ for $i = 1, \dots n$, then $\bar{w}$ is a SS.*

Proof. Consider each subset $C(I_b, \bar{M}_i)$ of $\mathcal{M}(\bar{w})$. If for every of those subsets it holds that $\bar{M}_i \uparrow_u = \vec{0}_{m_u}$, then $C(I_b, \bar{M}_i) = cov(\bar{M}_i)$. It is not difficult to see that $\mathcal{M}(\bar{w}) = CS(N, M_0)$, hence $R(N, M_0) \subseteq \mathcal{M}(\bar{w})$ and $\bar{w}$ synchronizes to $\bar{M}$. □

Note that verifying whether a given PSS $\bar{w}$ is a SS is clearly possible without any further computation if it holds that $\forall e \in \bar{w} : \nexists t \in P_u^\bullet \cap T_e$.

Example 4.25. *Consider the PN in Figure 4.4a. One wants to synchronize the net to marking $\bar{M} = [1\,3\,0]^T$ and determine the set of marking $\mathcal{M}$. A possible PSS is $\bar{w} = e_3$, whose underlying firing sequences are $\sigma_0 = \sigma_1 = \sigma_2 = \varepsilon$ and $\sigma_3 = t_3$. Finally $\mathcal{M} = C(\{1,3\}, \bar{M}'_0) \bigcup C(\{1,3\}, \bar{M}'_3)$, where $\bar{M}'_0 = [1\,0\,0]^T$ and $\bar{M}'_3 = [0\,0\,1]^T$. The condition of Proposition 4.24 holds and $\bar{w}$ is then a SS for $\bar{M}$.* ■

Chapter 5

Diagnosis of labeled Petri nets

Summary

In this part an approach to on-line diagnosis of discrete event systems based on labeled Petri nets is presented. The approach is based on the notion of basis markings and justifications and it can be applied both to bounded and unbounded Petri nets whose unobservable subnet is acyclic. Moreover it is shown that, in the case of bounded Petri nets, the most burdensome part of the procedure may be moved off-line, computing a particular graph called *Basis Reachability Graph*.

Finally the effectiveness of the proposed procedure is analyzed applying a MATLAB diagnosis toolbox we have developed to a manufacturing example taken from the literature.

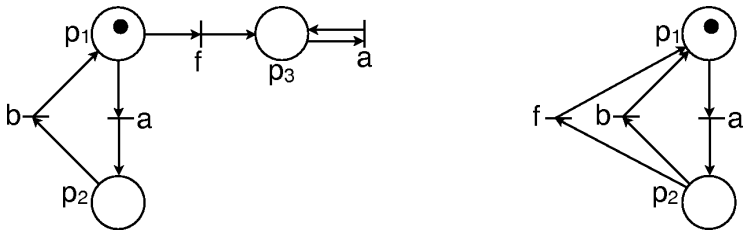


Figure 5.1: An example of a marked net with a permanent fault (a) and with an intermittent fault (b).

5.1 Discrete event diagnosis using labeled Petri nets

Failures are inevitable in today's complex industrial environment and they could arise from several sources such as design errors, equipment malfunctions, operator mistakes, and so on. As technology advances, as systems of increasing size and functionality are built, and as increasing demands on the performance of these systems are placed, then the complexity of these systems increases. Consequently (and unfortunately), the potential for systems to fail is enhanced, and no matter how safe the designs are, how improved the quality control techniques are, and how better trained the operators are, system failures become unavoidable [111].

In the diagnosis of discrete event systems faults may correspond to any discrete event. In out approach, a categorization of faults is given from the manner in which they are recovered after they occur. It can be distinguished between permanent and intermittent faults. A fault is *permanent* if the recovery event occurs only due to a repair/replacement of the fault that is controllable and observable. On the contrary, a fault is *intermittent* if the recovery event can occur either spontaneously or through repair/replacement; it tends to be uncontrollable and unobservable. Example is a loose wire that makes and breaks contact spontaneously. It is important to distinguish between these two types of faults. In fact, intermittent faults may spontaneously recover, making the system oscillate between non-faulty and faulty states. On the other hand, in the case of permanent faults, the system cannot spontaneously move from a fault state to a non-fault one [64].

Examples of permanent and intermittent faults in a given net system are presented in Figure 5.1.(a) and Figure 5.1.(b) respectively. In particular, the net system during the nominal behavior produces a cyclic sequence of "a" followed by "b". In Figure 5.1.(a) a permanent fault to the sensor that produces "b" is modeled: after the occurrence of the fault f only events a will be produced. On the contrary, Figure 5.1.(b) represents an intermittent fault to the sensor "b": after the occurrence of the fault event f the sensor that produces b may start working again.

5.1.1 Characterization of the set of consistent markings

To solve a diagnosis problem, it is essential to be able to compute the set of sequences and markings consistent with a given observation w. In this section a formalism that allows

one to characterize these sets without resorting to explicit enumeration is provided. The approach is based on the notions of minimal explanations and basis markings that are introduced in the following two subsections.

5.1.1.1 Minimal explanations and minimal e-vectors

In this subsection the notion of minimal explanation for unlabeled PNs is introduced and later it is extended to labeled PNs.

Definition 5.1. ***(explanations and e-vectors)*** *Given a marking M and an observable transition $t \in T_o$, let*

$$\Sigma(M,t) = \{\sigma \in T_u^* \mid M[\sigma\rangle M',\ M' \geq Pre(\cdot,t)\}$$

be the set of explanations *of t at M, and let*

$$Y(M,t) = \pi(\Sigma(M,t))$$

be the e-vectors *(or* explanation vectors*), i.e., the firing vectors associated with the explanations.* ■

Thus $\Sigma(M,t)$ is the set of unobservable sequences whose firing at M enables t. Among the above sequences select those whose firing vector is minimal. The firing vector of these sequences are called *minimal e-vectors.*

Definition 5.2. ***(minimal explanations and minimal e-vectors)*** *Given a marking M and a transition $t \in T_o$, let us define*[1]

$$\Sigma_{\min}(M,t) = \{\sigma \in \Sigma(M,t) \mid \nexists\ \sigma' \in \Sigma(M,t) : \pi(\sigma') \lneqq \pi(\sigma)\}$$

the set of minimal explanations *of t at M, and let us define*

$$Y_{\min}(M,t) = \pi(\Sigma_{\min}(M,t))$$

the corresponding set of minimal e-vectors. ■

Example 5.3. *Consider the PN in Figure 5.2 previously introduced in Example 2.29. It holds that $\Sigma(M_0,t_1) = \{\varepsilon\}$. Then $\Sigma(M_0,t_2) = \varnothing$. Finally, let $M = [\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ ]^T$, it holds that $\Sigma(M,t_5) = \{\varepsilon, \varepsilon_8, \varepsilon_8\varepsilon_9, \varepsilon_8\varepsilon_{11}, \varepsilon_8\varepsilon_9\varepsilon_{10}\}$, while $\Sigma_{min}(M,t_5) = \{\varepsilon\}$. It follows that $Y(M,t_5) = \{[0\ 0\ 0\ 0\ 0\ 0]^T, [1\ 0\ 0\ 0\ 0\ 0]^T, [1\ 1\ 0\ 0\ 0\ 0]^T, [1\ 0\ 0\ 1\ 0\ 0]^T, [1\ 1\ 1\ 0\ 0\ 0]^T\}$, and $Y_{min}(M,t_5) = \{[0\ 0\ 0\ 0\ 0\ 0]^T\}$.* ■

In [30] it was shown that, if the unobservable subnet is acyclic and backward conflict-free, then $|Y_{\min}(M,t)| = 1$.

[1] Given two vectors x and y, $x \lneqq y$ denotes that all components of x are less than or equal to all corresponding components of y and there exists at least one component of x that is strictly less than the corresponding component of y.

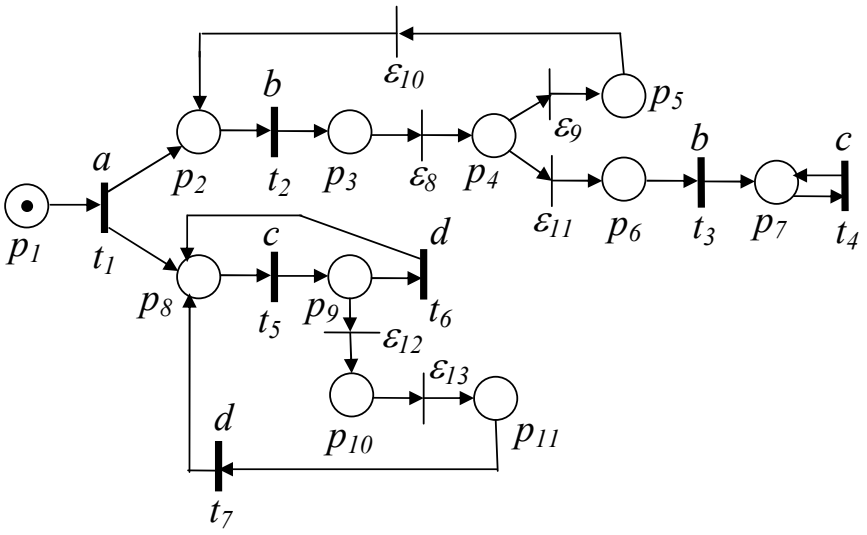


Figure 5.2: A marked labeled PN.

Different approaches can be used to compute $Y_{\min}(M,t)$, e.g., [11, 67]. Here, it is suggested an approach that terminates finding all vectors in $Y_{\min}(M,t)$ if applied to nets whose T_u-induced subnet is acyclic. It simply requires algebraic manipulations, and is inspired by the procedure proposed by [86] for the computation of minimal P-invariants. It can be briefly summarized by the following algorithm.

Algorithm 5.4. ***(Computation of*** $Y_{\min}(M,t)$***)***
Input: *a marking* M *and a transition* t *for a PN* $N = (P,T,Pre,Post)$.
Ouput: *the set of corresponding minimal e-vectors.*

1. *Let* $\Gamma := \left|\begin{array}{c|c} C_u^T & I_{n_u \times n_u} \\ \hline A & B \end{array}\right|$ *where* $A := (M - Pre(\cdot,t))^T$, $B := \vec{0}_{n_u}^T$.
2. ***While*** *A has negative entries, do*
  2.1. *choose an element* $A(i^*,j^*) < 0$*;*
  2.2. *let* $\mathcal{I}^+ = \{i \mid C_u^T(i,j^*) > 0\}$*;*
  2.3. *for all* $i \in \mathcal{I}^+$*, add to* $[A \mid B]$ *a new row*
    $[A(i^*,\cdot) + C_u^T(i,\cdot) \mid B(i^*,\cdot) + \vec{e}_i{}^T]$
    where $\vec{e}_i$ *is the* i*-th canonical basis vector.*
  2.4. *Remove the row* $[A(i^*,\cdot) \mid B(i^*,\cdot)]$ *from the table.*
3. *Remove from* B *any row that covers other rows.*
  End while
4. *Each row of* B *is a vector in* $Y_{\min}(M,t)$. ■

The above algorithm can be explained as follows.

Given a marking M and a transition t, Algorithm 5.4 computes the *minimal e-vectors* , i.e., the firing vectors of unobservable sequences whose firing at M is necessary to enable t.

At step 1 a row vector is defined, $A = A(1,\cdot)$, that has a number of columns equal to number of places of the net. This vector contains a negative element $A(1,j)$ if place p_j does not enable t at M. In particular, the absolute value $|A(1,j)|$ denotes the number of tokens missing from p_j to enable t at M. Finally, B is initially a null firing vector.

While A has negative entries, one of such entries is chosen and at step 2.2 it is checked if there exists an unobservable transition whose firing may increase the number of tokens in p_j: if so all possible such firings (of a single transition) computing the markings reached by each of these firings are considered. Vector B, in the right part of the table, denotes the corresponding firing vector. These new markings and the correspondent firing vectors will be the new rows of matrix A, while the previous row is removed.

Note that at step 2.3 it may be possible that the new row $[A(i^*,\cdot)+C_u^T(i,\cdot) \mid B(i^*,\cdot)+\vec{e}_i{}^T]$ is identical to a row already in the table: if such is the case it is not necessary to add it.

The *while* loop is repeated until all markings represented by matrix A have non negative components.

5.1.1.2 Basis markings and j-vectors

In this subsection, the definitions of basis markings and justifications, that are the crucial notions of the diagnosis approach presented in this thesis, are introduced.

In particular, given a sequence of observed events $w \in L^*$, a basis marking M_b is a marking reached from M_0 with the firing of the observed word w and of all unobservable transitions whose firing is strictly necessary to enable w. Such a sequence of unobservable transitions is called *justification.* Note that in general several sequences $\sigma_o \in T_o^*$ may correspond to the same w, i.e., there are several sequences of observable transitions such that $\mathcal{L}(\sigma_o) = w$ that may have actually fired. Moreover, in general, to any of such sequences σ_o a different sequence of unobservable transitions interleaved with it is necessary to make it firable at the initial marking. Thus the introduction of the following definition of pairs (sequence of transitions in T_o labeled w; corresponding justification) is needed.

Definition 5.5. ***(justifications and j-vectors)*** *Let $\langle N, M_0\rangle$ be a marked net with labeling function $\mathcal{L}: T \to L \cup \{\varepsilon\}$, where $N = (P,T,Pre,Post)$ and $T = T_o \cup T_u$. Let $w \in L^*$ be a given observation. Let*

$$\begin{aligned}\hat{\mathcal{J}}(w) \;= \{\; &(\sigma_o,\sigma_u),\ \sigma_o \in T_o^*,\ \mathcal{L}(\sigma_o) = w,\ \sigma_u \in T_u^* \mid \\ &[\exists \sigma \in \mathcal{S}(w) \;:\; \sigma_o = P_o(\sigma),\ \sigma_u = P_u(\sigma)] \wedge \\ &[\nexists\, \sigma' \in \mathcal{S}(w) :\ \sigma_o = P_o(\sigma'),\ \sigma'_u = P_u(\sigma') \wedge \\ &\qquad\qquad \pi(\sigma'_u) \not\leq \pi(\sigma_u)]\}\end{aligned}$$

be the set of pairs (sequence $\sigma_o \in T_o^*$ with $\mathcal{L}(\sigma_o) = w$, corresponding justification *of w). Moreover, let*

$$\begin{aligned}\hat{Y}_{\min}(M_0,w) = \{&(\sigma_o,y),\ \sigma_o \in T_o^*, \mathcal{L}(\sigma_o) = w, y \in \mathbb{N}^{n_u} \mid \\ &\exists (\sigma_o,\sigma_u) \in \hat{\mathcal{J}}(w) : \pi(\sigma_u) = y\}\end{aligned}$$

be the set of pairs (sequence $\sigma_o \in T_o^*$ with $\mathcal{L}(\sigma_o) = w$, corresponding j-vector*).* ■

In simple words, $\hat{\mathcal{J}}(w)$ is the set of pairs whose first element is the sequence $\sigma_o \in T_o^*$ labeled w and whose second element is the corresponding sequence of unobservable transitions interleaved with σ_o whose firing enables σ_o and whose firing vector is minimal. The firing vectors of these sequences are called *j-vectors*.

Example 5.6. *Consider the PN in Figure 5.2 previously introduced in Example 2.29.*

Assume $w = ab$. *In this case* $\hat{\mathcal{J}}(w) = \{(t_1t_2, \varepsilon)\}$ *and* $\hat{Y}_{min}(M_0, w) = \{(t_1t_2, \vec{0})\}$. *Now, consider* $w = acd$. *The set* $\hat{\mathcal{J}}(w) = \{(t_1t_5t_6, \varepsilon), (t_1t_5t_7, \varepsilon_{12}\varepsilon_{13})\}$ *and* $\hat{Y}_{min}(M_0, w) = \{(t_1t_5t_6, \vec{0}),$ $(t_1t_5t_7, [0\ 0\ 0\ 0\ 1\ 1]^T)\}$. ■

The main difference among minimal explanations and justifications is that the first ones are functions of a generic marking M and transition t, while justifications are functions of the initial marking M_0 and w. Moreover, as will be claimed in the following Proposition 5.9, in the case of acyclic unobservable subnets, justifications can be computed recursively summing up minimal explanations.

Definition 5.7. ***(basis markings)*** *Let* $\langle N, M_0\rangle$ *be a marked net with labeling function* $\mathcal{L} : T \to L \cup \{\varepsilon\}$, *where* $N = (P, T, Pre, Post)$ *and* $T = T_o \cup T_u$. *Let* w *be a given observation and* $(\sigma_o, \sigma_u) \in \hat{\mathcal{J}}(w)$ *be a generic pair (sequence of observable transitions labeled* w*; corresponding justification). The marking*

$$M_b = M_0 + C_u \cdot y + C_o \cdot y', \qquad y = \pi(\sigma_u),\ \ y' = \pi(\sigma_o),$$

i.e., the marking reached firing σ_o *interleaved with the justification* σ_u, *is called* basis marking *and* y *is called its* j-vector *(or* justification-vector*).* ■

Obviously, because in general more than one justification exists for a word w (the set $\hat{\mathcal{J}}(w)$ is generally not a singleton), the basis marking may be not unique as well.

Definition 5.8. ***(basis marking and relative j-vector)*** *Let* $\langle N, M_0\rangle$ *be a marked net with labeling function* $\mathcal{L} : T \to L \cup \{\varepsilon\}$, *where* $N = (P, T, Pre, Post)$ *and* $T = T_o \cup T_u$. *Let* $w \in L^*$ *be an observed word. Let*

$$\begin{aligned}\mathcal{M}(w) = \{(M, y) \mid\ & (\exists \sigma \in \mathcal{S}(w)\ :\ M_0[\sigma\rangle M)\ \wedge \\ & (\exists (\sigma_o, \sigma_u) \in \hat{\mathcal{J}}(w)\ :\ \sigma_o = P_o(\sigma), \\ & \qquad \sigma_u = P_u(\sigma),\ y = \pi(\sigma_u))\}\end{aligned}$$

be the set of pairs (basis marking, relative j-vector) that are consistent *with* $w \in L^*$. ■

Note that the set $\mathcal{M}(w)$ does not keep into account the sequences of observable transitions that may have actually fired. It only keeps track of the basis markings that can be reached and of the firing vectors relative to sequences of unobservable transitions that have fired to reach them. Indeed, this is the information really significant when performing diagnosis. The notion of $\mathcal{M}(w)$ is fundamental to provide a recursive way to compute the set of minimal explanations.

Proposition 5.9. *Given a marked net* $\langle N, M_0\rangle$ *with labeling function* $\mathcal{L} : T \to L \cup \{\varepsilon\}$, *where* $N = (P, T, Pre, Post)$ *and* $T = T_o \cup T_u$. *Assume that the unobservable subnet is acyclic. Let* $w = w'l$ *be a given observation.*

It holds:

$$\hat{Y}_{\min}(M_0, w'l) = \{(\sigma_o, y) \mid \begin{array}{l} \sigma_o = \sigma'_o t \wedge y = y' + e \ : \\ (\sigma'_o, y') \in \hat{Y}_{\min}(M_0, w'), \\ (t, e) \in \hat{Y}_{\min}(M'_b, l) \text{ and } \mathcal{L}(t) = l\}, \end{array}$$

where $M'_b = M_0 + C_u \cdot y' + C_o \cdot \pi(\sigma'_o)$ and $\hat{Y}_{\min}(M'_b, l)$ is the set of pairs (transition labeled l that may have fired at M'_b, corresponding j-vector) introduced in Definition 5.5.

Proof: *Let us prove this result by induction on the length of the observed string w.*

(Basis step) *For $w = \varepsilon$ the result trivially follows from Definitions 5.5 and 5.7.*

(Inductive step) *Assume the result is valid for w'. Let us prove it is also true for $w = w'l$ where $l = \mathcal{L}(t)$.*

In fact, if there exists a sequence $w = w'l \in L^*$, such that $M_0[\sigma_o\rangle\tilde{M}$ with $\mathcal{L}(\sigma_o) = w$ then there exist sequences σ' and σ'' such that

$$M_0[\sigma'\rangle M'[t\rangle M''[\sigma''\rangle\tilde{M}$$

where $\mathcal{L}(\sigma') = w'$, $\mathcal{L}(t) = l$ and $\sigma'' \in T_u^*$. By induction, there exists $(M'_b, y') \in \mathcal{M}(w')$ such that

$$M_0[\sigma'_a\rangle M'_b[\sigma'_b\rangle M'[t\rangle M''[\sigma''\rangle\tilde{M}$$

where $\mathcal{L}(\sigma'_a) = w'$, $\pi(\sigma'_a) = \pi(\sigma'_o) + y'$ and $\sigma'_b \in T_u^*$. Now there exists at least one minimal explanation[2] $\sigma'_c \in \hat{\Sigma}_{\min}(M'_b, l)$ such that $\pi(\sigma'_c) \leq \pi(\sigma'_b)$ and, since the T_u-induced subnet is acyclic the state equation gives necessary and sufficient conditions for the reachability, thus the marking reached is not dependent by the order of the firing of the unobservable transitions, thus

$$M_0[\sigma'_a\rangle M'_b[\sigma'_c\rangle M'_c[t\rangle M'_d[\sigma'_d\rangle M''[\sigma''\rangle\tilde{M} \tag{5.1}$$

where $\pi(\sigma'_c) + \pi(\sigma'_d) = \pi(\sigma'_b)$ and $(M'_d, \pi(\sigma'_c)) \in \mathcal{M}(w'l) = \mathcal{M}(w)$. From eq. (5.1) it holds

$$M'_b = M_0 + C \cdot \pi(\sigma'_a) = M_0 + C_u \cdot y' + C_o \cdot \pi(\sigma'_o),$$

$$M'_d = M'_b + C_u \cdot \pi(\sigma'_c) + C_o \cdot \vec{t} = M_0 + C_u \cdot (y' + \pi(\sigma'_c)) + C_o \cdot (\pi(\sigma'_o) + \vec{t}).$$

Thus $\sigma_o = \sigma'_o t$ and $y = y' + \pi(\sigma'_c)$, where $(\sigma_0, y) \in \hat{Y}_{min}(M_0, w'l)$. □

Example 5.10. *Consider the PN in Figure 5.2 previously introduced in Example 2.29.*

Assume $w = ab$. As shown in Example 5.6 $\hat{\mathcal{J}}(w) = \{(t_1 t_2, \varepsilon)\}$, thus the basis marking is $M_b = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$, and $\mathcal{M}(w) = \{(M_b, \vec{0})\}$.

Now, consider $w = acd$. As computed in Example 5.6, the set $\hat{\mathcal{J}}(w) = \{(t_1 t_5 t_6, \varepsilon)$, $(t_1 t_5 t_7, \varepsilon_{12}\varepsilon_{13})\}$. All the above j-vectors lead to the same basis marking $M_b = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$ thus $\mathcal{M}(w) = \{(M_b, \vec{0}), (M_b, [0\ 0\ 0\ 0\ 1\ 1]^T)\}$. ■

By Proposition 5.9, under the assumption of acyclicity of the unobservable subnet, the set $\mathcal{M}(w)$ can be easily constructed as follows.

[2] In fact, being the T_u–induced subnet acyclic, this is always true for all sequences σ'_c enabled at M and such that $\pi(\sigma'_c) = \pi(\sigma'_b)$ (in such case $\pi(\sigma'_d) = \vec{0}$).

Algorithm 5.11. ***(Computation of the basis markings and j-vectors)***
Input: *a labeled marked net* $\langle N, M_0 \rangle$ *with labeling function* $\mathcal{L} : T \to L \cup \{\varepsilon\}$.
Ouput: *the basis marking associated with a stream of observed labels.*

1. *Let* $w = \varepsilon$.
2. *Let* $\mathcal{M}(w) = \{(M_0, \vec{0})\}$.
3. *Wait until a new label* l *is observed.*
4. *Let* $w' = w$ *and* $w = w'l$.
5. *Let* $\mathcal{M}(w) = \varnothing$.
6. ***For all*** M' *such that* $(M', y') \in \mathcal{M}(w')$ *, do*
  6.1. for all $t \in T_l$, *do*
    6.1.1. for all $e \in Y_{\min}(M', t)$, ***do***
      6.1.1.1. *let* $M = M' + C_u \cdot e + C(\cdot, t)$,
      6.1.1.2. for all y' *such that* $(M', y') \in \mathcal{M}(w')$, *do*
        6.1.1.2.1. *let* $y = y' + e$,
        6.1.1.2.2. *let* $\mathcal{M}(w) = \mathcal{M}(w) \cup \{(M, y)\}$.
          End for
        End for
    End for
  End for
7. *Goto tep 3.* ■

In simple words, the above algorithm can be explained as follows. Assume that, after a certain word w' has been observed, a new observable t fires and its label $l = \mathcal{L}(t)$ is observed. Consider all basis markings at the observation w' and select among them those that may have allowed the firing of at least one transition $t \in T_l$, also taking into account that this may have required the firing of appropriate sequences of unobservable transitions. In particular, let us focus on the minimal explanations, and thus on the corresponding minimal e-vectors (step 6.1.1). Finally, update the set $\mathcal{M}(w't)$ including all pairs of new basis markings and j-vectors, taking into account that for each basis marking at w' it may correspond more than one j-vector.

Definition 5.12. ***(Set of basis markings at*** w***)*** *Let* $\langle N, M_0 \rangle$ *be a marked net where* $N = (P, T, Pre, Post)$ *and* $T = T_o \cup T_u$. *Assume that the unobservable subnet is acyclic. Let* $w \in T_o^*$ *be an observed word. Let*

$$\mathcal{M}_{basis}(w) = \{M \in \mathbb{N}^m \mid \exists y \in \mathbb{N}^{n_u} \text{ and } (M, y) \in \mathcal{M}(w)\}$$

be the set of basis markings at w. *Moreover, denote as*

$$\mathcal{M}_{basis} = \bigcup_{w \in T_o^*} \mathcal{M}_{basis}(w)$$

the set of all basis markings for any observation w. ■

Note that if the marked net is bounded then the set $\mathcal{M}_{basis}$ is *finite* being the set of basis markings a subset of the reachability set.

In the following theorem a result proved for unlabeled PNs [18] is extended to labeled PNs.

Theorem 5.13. *Consider a marked net $\langle N, M_0 \rangle$ whose unobservable subnet is acyclic. For any $w \in L^*$ it holds that*

$$\mathcal{C}(w) = \{M \in \mathbb{N}^m \mid M = M_b + C_u \cdot y \ : y \geq \vec{0} \ \text{ and } \ M_b \in \mathcal{M}_{basis}(w)\}.$$

Proof: *This proof has been given for unlabeled PNs in [18]. Also in the case of labeled PNs a formal proof can be given by induction on the length of observed string w, following the same arguments in the proof of Proposition 5.9.* □

The above result shows that the set $\mathcal{C}(w)$ can be characterized in linear algebraic terms given the set $\mathcal{M}_{basis}(w)$, thus not requiring exhaustive enumeration. This is the main advantage of the approach here presented.

5.1.2 Diagnosis using Petri nets

Assume that the set of unobservable transitions is partitioned into two subsets, namely $T_u = T_f \cup T_{reg}$ where T_f includes all fault transitions (modeling anomalous or fault behavior), while T_{reg} includes all transitions relative to unobservable but regular events. The set T_f is further partitioned into r different subsets T_f^i, where $i = 1, \ldots, r$, that model the different fault classes. Usually, fault transitions that belong the same fault class are transitions that represent similar physical faulty behavior.

The following definition introduces the notion of *diagnoser*.

Definition 5.14. ***(diagnoser)*** *A* diagnoser *is a function $\Delta : L^* \times \{T_f^1, T_f^2, \ldots, T_f^r\} \to \{0, 1, 2, 3\}$ that associates to each observation $w \in L^*$ and to each fault class T_f^i, $i = 1, \ldots, r$, a* diagnosis state.

- $\Delta(w, T_f^i) = 0$ *if for all $\sigma \in \mathcal{S}(w)$ and for all $t_f \in T_f^i$ it holds $t_f \notin \sigma$.*

  In such a case the ith fault cannot have occurred, because none of the firing sequences consistent with the observation contains fault transitions of class i.

- $\Delta(w, T_f^i) = 1$ *if:*

  (i) there exist $\sigma \in \mathcal{S}(w)$ and $t_f \in T_f^i$ such that $t_f \in \sigma$ but

  (ii) for all $(\sigma_o, \sigma_u) \in \hat{\mathcal{J}}(w)$ and for all $t_f \in T_f^i$ it holds that $t_f \notin \sigma_u$.

  In such a case a fault transition of class i may have occurred but is not contained in any justification of w.

- $\Delta(w, T_f^i) = 2$ *if there exist $(\sigma_o, \sigma_u), (\sigma_o', \sigma_u') \in \hat{\mathcal{J}}(w)$ such that*

  (i) there exists $t_f \in T_f^i$ such that $t_f \in \sigma_u$;

  (ii) for all $t_f \in T_f^i$, $t_f \notin \sigma_u'$.

  In such a case a fault transition of class i is contained in one (but not in all) justification of w.

- $\Delta(w, T_f^i) = 3$ *if for all* $\sigma \in \mathcal{S}(w)$ *there exists* $t_f \in T_f^i$ *such that* $t_f \in \sigma$.

  In such a case the ith fault must have occurred, because all firable sequences consistent with the observation contain at least one fault in T_f^i. ■

Note that assuming that certain transitions belong to the same fault class is not a restrictive assumption. On the contrary, it makes the presentation more general. If one is interested in reconstructing the occurrence of a particular transition t_f, with no ambiguity with other transitions, it is sufficient to define a fault class only containing t_f.

Example 5.15. *Consider the PN in Figure 5.2 previously introduced in Example 2.29. Let* $T_f = \{\varepsilon_{11}, \varepsilon_{12}\}$. *Assume that the two fault transitions belong to different fault classes, i.e.,* $T_f^1 = \{\varepsilon_{11}\}$ *and* $T_f^2 = \{\varepsilon_{12}\}$.

Let us observe $w = a$. *Then* $\Delta(w, T_f^1) = \Delta(w, T_f^2) = 0$, *being* $\hat{\mathcal{J}}(w) = \{(t_1, \varepsilon)\}$ *and* $\mathcal{S}(w) = \{t_1\}$. *In simple words no fault of both fault classes may have occurred.*

Let us observe $w = ab$. *Then* $\Delta(w, T_f^1) = 1$ *and* $\Delta(w, T_f^2) = 0$, *being* $\hat{\mathcal{J}}(w) = \{(t_1t_2, \varepsilon)\}$ *and* $\mathcal{S}(w) = \{t_1t_2, t_1t_2\varepsilon_8, t_1t_2\varepsilon_8\varepsilon_9, t_1t_2\varepsilon_8\varepsilon_9\varepsilon_{10}, t_1t_2\varepsilon_8\varepsilon_{11}\}$. *This means that a fault of the first fault class may have occurred (firing the sequence* $t_1t_2\varepsilon_8\varepsilon_{11}$*) but it is not contained in any justification of ab, while no fault of the second fault class can have occurred.*

Now, consider $w = abb$. *In this case* $\Delta(w, T_f^1) = 2$ *and* $\Delta(w, T_f^2) = 0$, *being* $\hat{\mathcal{J}}(w) = \{(t_1t_2t_2, \varepsilon_8\varepsilon_9\varepsilon_{10}), (t_1t_2t_3, \varepsilon_8\varepsilon_{11})\}$ *and* $\mathcal{S}(w) = \{t_1t_2\varepsilon_8\varepsilon_9\varepsilon_{10}t_2, t_1t_2\varepsilon_8\varepsilon_9\varepsilon_{10}t_2\varepsilon_8, t_1t_2\varepsilon_8\varepsilon_9\varepsilon_{10}t_2\varepsilon_8\varepsilon_9, t_1t_2\varepsilon_8\varepsilon_9\varepsilon_{10}t_2\varepsilon_8\varepsilon_9\varepsilon_{10}, t_1t_2\varepsilon_8\varepsilon_9\varepsilon_{10}t_2\varepsilon_8\varepsilon_{11}, t_1t_2\varepsilon_8\varepsilon_{11}t_3\}$. *This means that no fault of the second fault class can have occurred, while a fault of the first fault class may have occurred since one justification does not contain* ε_{11} *and one justification contains it.*

Finally, consider $w = abbccc$. *In this case* $\Delta(w, T_f^1) = 3$ *and* $\Delta(w, T_f^2) = 1$. *In fact since* $\hat{\mathcal{J}}(w) = \{(t_1t_2t_3t_5t_4t_4, \varepsilon_8\varepsilon_{11}), (t_1t_2t_3t_4t_5t_4, \varepsilon_8\varepsilon_{11}), (t_1t_2t_3t_4t_4t_5, \varepsilon_8\varepsilon_{11}), (t_1t_2t_3t_4t_4t_4, \varepsilon_8\varepsilon_{11})\}$ *a fault of the first fault class must have occurred, while a fault of the second fault class may have occurred (e.g.* $t_1t_2\varepsilon_8\varepsilon_{11}t_3t_4t_4t_5\varepsilon_{12}$*) but it is not contained in any justification of* w. ■

The following two results proved in [18] for unlabeled PNs still hold in the case of labeled PNs. In particular, the following proposition presents how the diagnosis states can be characterized analyzing basis markings and justifications.

Proposition 5.16. *Consider an observed word* $w \in L^*$.

- $\Delta(w, T_f^i) \in \{0, 1\}$ *iff for all* $(M, y) \in \mathcal{M}(w)$ *and for all* $t_f \in T_f^i$ *it holds* $y(t_f) = 0$.
- $\Delta(w, T_f^i) = 2$ *iff there exist* $(M, y) \in \mathcal{M}(w)$ *and* $(M', y') \in \mathcal{M}(w)$ *such that:*
  (i) there exists $t_f \in T_f^i$ *such that* $y(t_f) > 0$,
  (ii) for all $t_f \in T_f^i$, $y'(t_f) = 0$.
- $\Delta(w, T_f^i) = 3$ *iff for all* $(M, y) \in \mathcal{M}(w)$ *there exists* $t_f \in T_f^i$ *such that* $y(t_f) > 0$.

Proof: *By Definition 5.14,* $\Delta(w, T_f^i) = 0$ *iff no fault transition* $t_f \in T_f^i$ *is contained in any firing sequence that is consistent with* w*, while* $\Delta(w, T_f^i) = 1$ *iff no fault* $t_f \in T_f^i$ *is contained in any justification of* w *but there exists at leat one sequence that is consistent with* w *that contains a transition* $t_f \in T_f^i$*. Therefore, a necessary and sufficient condition to have* $\Delta(w, T_f^i) \in \{0, 1\}$ *is that for all j-vectors* y *at* w *and all* $t_f \in T_f^i$ *it is* $y(t_f) = 0$*, thus proving the first item.*

Analogously, $\Delta(w, T_f^i) = 2$ *if a transition* $t_f \in T_f^i$ *is contained in at least one (but not in all) justification of* w*. Thus, to have* $\Delta(w, T_f^i) = 2$ *it is necessary and sufficient that there exists at least one j-vector* y *that contains at least one transition* $t_f \in T_f^i$ *and one j-vector* y' *that does not contain transitions* $t_f \in T_f^i$*, thus proving the second item.*

Finally, given an observed word w *and a fault class* T_f^i *it holds* $\Delta(w, T_f^i) = 3$ *if all firable sequences consistent with* w *contain at least one fault transition* $t_f \in T_f^i$*. Thus, to have* $\Delta(w, T_f^i) = 3$ *it is necessary and sufficient that all justifications contain at least one transition* $t_f \in T_f^i$*. This proves the third item.* □

The following proposition shows how to distinguish between diagnosis states 0 and 1.

Proposition 5.17. *For a PN whose unobservable subnet is acyclic, let* $w \in L^*$ *be an observed word such that for all* $(M, y) \in \mathcal{M}(w)$ *it holds* $y(t_f) = 0 \ \forall \ t_f \in T_f^i$*. Consider the constraint set*

$$\mathcal{T}(M, T_f^i) \ = \ \begin{cases} M + C_u \cdot z \geq \vec{0}, \\ \sum\limits_{t_f \in T_f^i} z(t_f) > 0, \\ z \in \mathbb{N}^{n_u}. \end{cases} \tag{5.2}$$

- $\Delta(w, T_f^i) = 0$ *if* $\forall \ (M, y) \in \mathcal{M}(w)$ *the constraint set* (5.2) *is not feasible.*
- $\Delta(w, T_f^i) = 1$ *if* $\exists \ (M, y) \in \mathcal{M}(w)$ *such that the constraint set* (5.2) *is feasible.*

Proof: *Let* $w \in L^*$ *be an observed word such that* $\forall (M, y) \in \mathcal{M}(w)$ *it is* $y(t_f) = 0 \ \forall \ t_f \in T_f^i$*. By Definition 5.14 it immediately follows that:*

- $\Delta(w, T_f^i) = 0$ *if* $\forall (M, y) \in \mathcal{M}(w)$ *and* $\forall t_f \in T_f^i$ *there does not exist a sequence* $\sigma \in T_u^*$ *such that* $M[\sigma\rangle$ *and* $t_f \in \sigma$*;*
- $\Delta(w, T_f^i) = 1$ *if* $\exists$ *at least one* $(M, y) \in \mathcal{M}(w)$ *and a sequence* $\sigma \in T_u^*$ *such that for at least one* $t_f \in T_f^i$*,* $M[\sigma\rangle$ *and* $t_f \in \sigma$*.*

Now, as proved in [31] if a generic PN is acyclic *its state equation gives necessary and sufficient conditions for marking reachability. Therefore, if such a result is applied to the unobservable subnet, that is acyclic by assumption, it can be concluded that the set* $\mathcal{T}(M, T_f^i)$ *characterizes the reachability set of the unobservable net at marking* M*. Thus, due to this fact and the above two items, it can be concluded that there exists a sequence containing a transition* $t_f \in T_f^i$ *firable at* M *on the unobservable subnet if and only if* $\mathcal{T}(M, T_f^i)$ *is feasible.* □

On the basis of the above two results, if the unobservable subnet is acyclic, diagnosis may be carried out by simply looking at the set $\mathcal{M}(w)$ for any observed word w and, should the diagnosis state be either 0 or 1, by additionally evaluating whether the corresponding integer constraint set (5.2) admits a solution.

Example 5.18. *Consider the PN in Figure 5.2 where* $T_f^1 = \{\varepsilon_{11}\}$ *and* $T_f^2 = \{\varepsilon_{12}\}$.

Let $w = ab$. *In this case* $\mathcal{M}(w) = \{(M_b^1, \vec{0})\}$, *where* $M_b^1 = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$. *Being* $\mathcal{T}(M_b^1, T_f^i)$ *feasible only for* $i = 1$ *it holds* $\Delta(w, T_f^1) = 1$ *and* $\Delta(w, T_f^2) = 0$.

Let $w = abb$. *It is* $\mathcal{M}(w) = \{(M_b^1, [1\ 1\ 1\ 0\ 0\ 0]^T),\ (M_b^2, [1\ 0\ 0\ 1\ 0\ 0]^T)\}$, *where* $M_b^2 = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0]^T$. *It is* $\Delta(w, T_f^1) = 2$ *and* $\Delta(w, T_f^2) = 0$ *being both* $\mathcal{T}(M_b^1, T_f^2)$ *and* $\mathcal{T}(M_b^2, T_f^2)$ *not feasible.*

Let $w = abbccc$. *In this case* $\mathcal{M}(w) = \{(M_b^3, [1\ 1\ 1\ 0\ 0\ 0]^T), (M_b^4, [1\ 1\ 1\ 0\ 0\ 0]^T)\}$, *where* $M_b^3 = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0]^T$ *and* $M_b^4 = [0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0]^T$. *It is* $\Delta(w, T_f^1) = 3$ *and being* $\mathcal{T}(M_b^4, T_f^2)$ *feasible it holds* $\Delta(w, T_f^2) = 1$. ■

The approach described above requires to compute for each observed word w and for each fault class i a diagnosis state $\Delta(w, T_f^i)$. Let us conclude this section with a brief discussion on the definition of diagnosis states $\Delta = 1$ and $\Delta = 2$. Firstly, observe that both the diagnosis states correspond to *uncertain states* even if a higher degree of alarm is associated with $\Delta = 2$ with respect to $\Delta = 1$. Secondly, observe that an advantage in terms of computational complexity can be obtained by splitting the uncertain condition in two diagnosis states, namely $\Delta = 1$ and $\Delta = 2$. In fact, the diagnosis approach is based on the preliminary computation of the set $\mathcal{M}(w)$. If $\Delta = 2$ or $\Delta = 3$ no additional computation is required. On the contrary to distinguish among $\Delta = 0$ and $\Delta = 1$ an integer programming problem should be solved.

5.1.3 Basis Reachability Graph

Diagnosis approach described in the previous section can be applied both to bounded and unbounded PNs. The proposed approach is an on-line approach that for each new observed event updates the diagnosis state for each fault class computing the set of basis markings and j-vectors. Moreover if for the fault class T_f^i is necessary to distinguish between diagnosis states 0 and 1, it is also necessary to solve for each basis marking M_b the constraint set $\mathcal{T}(M_b, T_f^i)$.

In this section it is shown that if the considered marked net is bounded, the most burdensome part of the procedure can be moved off-line defining a graph called *Basis Reachability Graph* (BRG).

Definition 5.19. ***(Basis Reachability Graph)*** *The BRG is a deterministic graph that has as many nodes as the number of possible basis markings.*

To each node is associated a different basis marking M *and a row vector with as many entries as the number of fault classes. The entries of this vector may only take binary values:* 1 *if* $\mathcal{T}(M, T_f^i)$ *is feasible,* 0 *otherwise.*

Arcs are labeled with observable events in L and e-vectors. More precisely, an arc exists from a node containing the basis marking M to a node containing the basis marking M' if and only if there exists a transition t for which an explanation exists at M and the firing of t and one of its minimal explanations leads to M'. The arc going from M to M' is labeled $(\mathcal{L}(t), e)$, where $e \in Y_{\min}(M,t)$ and $M' = M + C_u \cdot e + C(\cdot,t)$. ■

Note that the number of nodes of the BRG is always finite being the set of basis markings a subset of the set of reachable markings, that is finite being the net bounded. Moreover, the row vector of binary values associated with the nodes of the BRG allows us to distinguish between the diagnosis state 1 or 0.

The main steps for the computation of the BRG in the case of labeled PNs are summarized in the following algorithm.

Algorithm 5.20. ***(Computation of the BRG)***
Input: *a labeled marked net $\langle N, M_0\rangle$ with labeling function $\mathcal{L}: T \to L \cup \{\varepsilon\}$.*
Ouput: *a diagnosis state associated with a stream of output labels.*

1. *Label the initial node (M_0, x_0) where $\forall i = 1, \ldots, r$,*

$$x_0(T_f^i) = \begin{cases} 1 & \text{if } \mathcal{T}(M_0, T_f^i) \text{ is feasible,} \\ 0 & \text{otherwise.} \end{cases}$$

Assign no tag to it.
2. ***While*** *nodes with no tag exist,*
select a node with no tag and do
2.1. *let M be the marking in the node (M, x),*
2.2. ***for all*** *$l \in L$*
2.2.1. ***for all*** *$t : L(t) = l \wedge Y_{\min}(M,t) \neq \emptyset$, do*
- *for all $e \in Y_{\min}(M,t)$, do*
  - *let $M' = M + C_u \cdot e + C(\cdot,t)$,*
  - ***if*** *$\nexists$ a node (M, x) with $M = M'$, do*
    - *add a new node to the graph containing (M', x') where $\forall i = 1, \ldots, r$,*

$$x'(T_f^i) = \begin{cases} 1 & \textbf{if } \mathcal{T}(M', T_f^i) \text{ is feasible,} \\ 0 & \text{otherwise.} \end{cases}$$

    and arc (l, e) from (M, x) to (M', x')
  - ***else***
    - *add arc (l, e) from (M, x) to (M', x')* ***if*** *it does not exist yet*

2.3. *tag the node "old".*
3. *Remove all tags.* ■

The algorithm constructs the BRG starting from the initial node to which it corresponds the initial marking and a binary vector defining which classes of fault may occur at M_0. Now, consider all the labels $l \in L$ such that there exists a transition t with $L(t) = l$ for which a minimal explanation at M_0 exists. For each of these transitions compute the marking resulting from firing t at $M_0 + C_u \cdot e$, for any $e \in Y_{\min}(M_0, t)$. If a pair (marking, binary vector) not contained in the previous nodes is obtained, a new node is added to the

M_0	[	1	0	0	0	0	0	0	0	0	0	]^T
M_1	[	0	1	0	0	0	0	0	1	0	0	]^T
M_2	[	0	1	0	0	0	0	0	0	1	0	0	]^T
M_3	[	0	0	1	0	0	0	0	1	0	0	0	]^T
M_4	[	0	0	1	0	0	0	0	0	1	0	0	]^T
M_5	[	0	0	0	0	0	0	1	1	0	0	0	]^T
M_6	[	0	0	0	0	0	0	1	0	1	0	0	]^T

Table 5.1: The markings of the BRG in Figure 5.3.

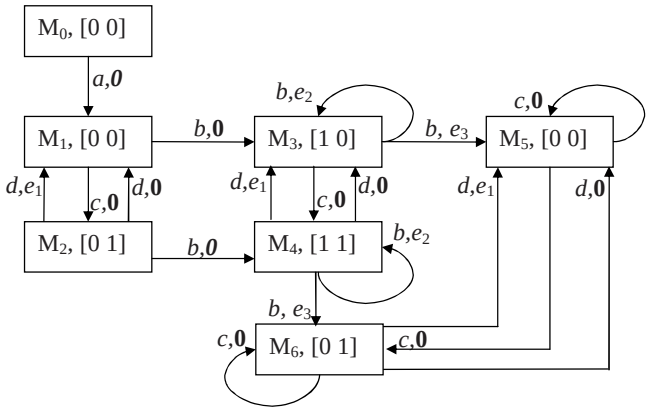


Figure 5.3: The BRG of the PN in Figure 5.2.

graph. The arc going from the initial node to the new node is labeled (l, e). The procedure is iterated until all basis markings have been considered. Note that the approach here presented always requires to enumerate a state space that is a subset (usually a strict subset) of the reachability space. However, as in general for diagnosis approaches, the combinatory explosion cannot be avoided.

Example 5.21. *Consider again the PN in Figure 5.2, where* $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, $T_u = \{\varepsilon_8, \varepsilon_9, \varepsilon_{10}, \varepsilon_{11}, \varepsilon_{12}, \varepsilon_{13}\}$, $T_f^1 = \{\varepsilon_{11}\}$ *and* $T_f^2 = \{\varepsilon_{12}\}$. *The labeling function is defined as follows:* $\mathcal{L}(t_1) = a$, $\mathcal{L}(t_2) = \mathcal{L}(t_3) = b$, $\mathcal{L}(t_4) = \mathcal{L}(t_5) = c$, $\mathcal{L}(t_6) = \mathcal{L}(t_7) = d$.

The BRG is shown in Figure 5.3. The notation used in in this figure is detailed in Tables 5.1 and 5.2. Each node contains a different basis marking and a binary row vector of dimension two, being two the number of fault classes. As an example, the binary vector $[0\ 0]$ *is associated with* M_0 *because* $\mathcal{T}(M_0, T_f^i)$ *is not feasible for* $i = 1$ *and* $i = 2$. *From node* M_0 *to node* M_1 *there is one arc labeled* a *with the null vector as minimal explanation. The node containing the basis marking* M_2 *has binary vector* $[0\ 1]$, *because* $\mathcal{T}(M_2, T_f^i)$ *is feasible only for* $i = 2$. *Node* $(M_2, [0\ 1])$ *has two output arcs both labeled with* d *and both directed to node* $(M_1, [0\ 0])$ *with two different minimal explanations* $\vec{0}$ *and* e_1, *respectively, plus another output arc* $(b, \vec{0})$ *directed to node* $(M_4, [1\ 1])$. ■

The following algorithm summarizes the main steps of the on-line diagnosis carried out by looking at the BRG.

Algorithm 5.22. ***(Diagnosis using the BRG)***
Input: *the BRG of a labeled PN.*
Ouput: *a diagnosis state associated with a stream of output labels.*

1. *Let* $w = \varepsilon$.
2. *Let* $\mathcal{M}(w) = \{(M_0, \vec{0})\}$.
3. *Wait until a new observable transition fires.*
Let l *be the observed event.*
4. *Let* $w' = w$ *and* $w = w'l$.
5. *Let* $\mathcal{M}(w) = \varnothing$, ***[Computation of*** $\mathcal{M}(w)$***]***
6. ***For all*** *nodes containing* M' *:* $(M', y') \in \mathcal{M}(w')$, ***do***
6.1. ***for all*** *arcs exiting from the node with* M', ***do***
6.1.1. *let* M *be the marking of the output node*
and e *be the minimal e-vector on the edge from* M' *to* M,
6.1.2. ***for all*** y' *such that* $(M', y') \in \mathcal{M}(w')$, ***do***
6.1.2.1. *let* $y = y' + e$,
6.1.2.2. *let* $\mathcal{M}(w) = \mathcal{M}(w) \cup \{(M, y)\}$,
7. ***for all*** $i = 1, \ldots, r$, ***do*** ***[Computation of the diagnosis state]***
7.1. ***if*** $\forall\ (M, y) \in \mathcal{M}(w) \wedge \forall t_f \in T_f^i$ *it is* $y(t_f) = 0$, *do*
7.1.1. ***if*** $\forall\ (M, y) \in \mathcal{M}(w)$ *it holds* $x(i) = 0$,
where x *is the binary vector in node* M, ***do***
7.1.1.1. *let* $\Delta(w, T_f^i) = 0$,
7.1.2. ***else***
7.1.2.1. *let* $\Delta(w, T_f^i) = 1$,
7.2. ***if*** $\exists\ (M, y) \in \mathcal{M}(w)$ *and* $(M', y') \in \mathcal{M}(w)$ *s.t.:*
(i) $\exists t_f \in T_f^i$ *such that* $y(t_f) > 0$,
(ii) $\forall t_f \in T_f^i$, $y'(t_f) = 0$, ***do***
7.2.1. *let* $\Delta(w, T_f^i) = 2$,
7.3. ***if*** $\forall\ (M, y) \in \mathcal{M}(w)\ \exists t_f \in T_f^i : y(t_f) > 0$, ***do***
7.3.1. *let* $\Delta(w, T_f^i) = 3$.
8. ***Goto*** *step 3.* ■

Steps 1 to 6 of Algorithm 5.22 enable us to compute the set $\mathcal{M}(w)$. When no event is observed, namely $w = \varepsilon$, then $\mathcal{M}(w) = \{(M_0, \vec{0})\}$. Now, assume that a label l is observed. All couples (M, y) such that an arc labeled l exits from the initial node and ends in a node containing the basis marking M are included in the set $\mathcal{M}(l)$. The corresponding value of y is equal to the e-vector in the arc going from M_0 to M, being $\vec{0}$ the j-vector relative to M_0. In general, if w' is the actual observation, and a new event labeled l fires, one has to consider all couples $(M', y') \in \mathcal{M}(w')$ and all nodes that can be reached from M' with an arc labeled l. Let M be the basis marking of the generic resulting node. Include in $\mathcal{M}(w) = \mathcal{M}(w'l)$ all couples (M, y), where for any M, y is equal to the sum of y' plus the e-vector labeling the arc from M' to M.

Step 7 of Algorithm 5.22 computes the diagnosis state. Consider the generic ith fault class. If $\forall (M, y) \in \mathcal{M}(w)$ and $\forall t_f \in T_f^i$ it holds $y(t_f) = 0$, the ith entry of all the binary row vectors associated with the basis markings M has to be checked, such that

	ε_8	ε_9	ε_{10}	ε_{11}	ε_{12}	ε_{13}
e_1	0	0	0	0	1	1
e_2	1	1	1	0	0	0
e_3	1	0	0	1	0	0

Table 5.2: The e-vectors of the BRG in Figure 5.3.

$(M, y) \in \mathcal{M}(w)$. If these entries are all equal to 0, it holds $\Delta(w, T_f^i) = 0$, otherwise it holds $\Delta(w, T_f^i) = 1$. On the other hand, if there exists at least one pair $(M, y) \in \mathcal{M}(w)$ with $y(t_f) > 0$ for any $t_f \in T_f^i$, and there exists at least one pair $(M', y') \in \mathcal{M}(w)$ with $y(t_f) = 0$ for all $t_f \in T_f^i$, then $\Delta(w, T_f^i) = 2$. Finally, if for all pairs $(M, y) \in \mathcal{M}(w)$, $y(t_f) > 0$ for any $t_f \in T_f^i$, then $\Delta(w, T_f^i) = 3$.

The following example shows how to perform diagnosis on-line simply looking at the BRG.

Example 5.23. *Consider the PN in Figure 5.2 and its BRG in Figure 5.3. Let* $w = \varepsilon$. *By looking at the BRG it holds that* $\Delta(\varepsilon, T_f^1) = \Delta(\varepsilon, T_f^2) = 0$ *being both entries of the row vector associated with* M_0 *equal to* 0.

Now, consider $w = ab$. *In such a case* $\mathcal{M}(w) = \{(M_3, \vec{0})\}$. *It holds* $\Delta(ab, T_f^1) = 1$ *and* $\Delta(ab, T_f^2) = 0$ *being the row vector in the node equal to* $[1\ 0]$.

Finally, for $w = abbc$ *it holds* $\Delta(abbc, T_f^1) = 2$ *and* $\Delta(abbc, T_f^2) = 1$. *In fact* $\mathcal{M}(w) = \{(M_4, y_1), (M_5, y_2), (M_6, y_3)\}$, *where* $y_1 = e_2$, $y_2 = y_3 = e_3$, *and the row vectors associated with* M_4 *and* M_5 *are respectively* $[1\ 1]$, $[0\ 0]$ *and* $[0\ 1]$. ■

Let us conclude this section observing that the BRG is a graph containing all information necessary for the construction of an observer. In the case of bounded PNs a modified version of the BRG is used to build the diagnoser that it is used to study the diagnosability of the system [16]. Note that, if an automaton has a number N of states, in the worst case (that depends on the labeling of events) the cardinality of the set of nodes of its observer is $2^N - 1$ [19]. On the contrary, the number of nodes of the BRG is equal to the number of basis markings that is at most equal to the number of reachable markings.

5.2 A Matlab toolbox for diagnosis of Petri nets

The tool is currently supported by the Distributed Supervisory Control of Large Plants (DISC) Software Platform. The DISC Project has been supported by the European Commission (see `http://www.disc-project.eu/software_platform.html`).

Note also that the developed tool provides other features whose theoretical results are not here presented. In particular, the problem of diagnosability can be investigated for the class of labeled PNs. An interesting comparisons with the other existing techniques can be found in [15], where a PN model is taken from the manufacturing area. This benchmark shows how the use of basis marking can greatly outperform tools based on an exhaustive enumeration of the state space.

5.2.1 The tool

In this section it is briefly illustrated a MATLAB function (BRG.m) that, given a bounded labeled PN, builds the basis reachability graph. This function, together with other MATLAB functions for diagnosis of labeled PNs, can be downloaded on the web [99].

The inputs of the MATLAB function BRG.m are:

- the structure of the net, i.e., the matrices Pre and $Post$;
- the initial marking M_0;
- a cell array F that has as many rows as the number of fault classes: each row contains the indices of the transitions that belong to the corresponding class;
- a cell array L that has as many rows as the cardinality of the considered alphabet: each row contains the indices of the observable transitions having the corresponding label;
- a cell array E that contains in each row a character (or a string of characters) defining a label of the considered alphabet. The cell array E is ordered according to L.

As an example, for the PN in Figure 5.2 introduced in the Example 2.29 the cell arrays F, L and E are:

$$F = \{[11];[12]\},\quad L = \{[1];[2\ \ 3];[4\ \ 5];[6\ \ 7]\},\quad E = \{['a'];['b'];['c'];['d']\}.$$

The output of the MATLAB function BRG.m is a cell array T that univocally identifies the resulting BRG. It has as many rows as the number of nodes of the BRG. A different row is associated with each node and contains the following information:

- an identifier number of the node;
- the transpose of the basis marking M_b^i associated with the node;
- a vector with as many columns as the number of fault classes: the jth element is equal to $x_i(T_f^j)$ evaluated at M_b^i. Thus, $x_i(T_f^j) = 0$ if $\mathcal{T}(M_b^i, T_f^j)$ is not feasible, 1 otherwise;
- the indices of the transitions enabled at the node;
- the identifier number of the nodes that are reached firing an enabled transition and the corresponding j-vector.

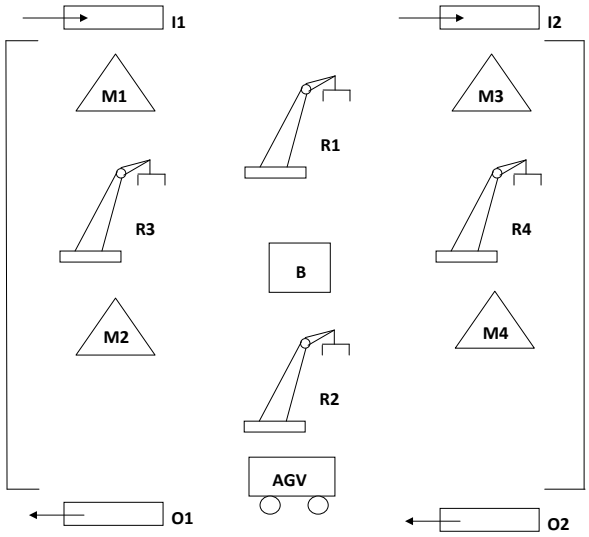


Figure 5.4: Layout of the manufacturing system.

5.2.2 A manufacturing example

In this section the diagnosis approach proposed in Section 5.1.2 is applied to an example modeling a manufacturing system. The considered net is similar to the one described in [138]. The automated manufacturing system layout is shown in Figure 5.4 and the corresponding PN is depicted in Figure 5.5, where thick transitions represent observable events and thin transitions represent unobservable events.

The plant consists of four machines (M1 to M4), four robots (R1 to R4), one AGV system (AGV), one buffer of finite capacity (B), two inputs of parts to be processed (I1 and I2) and two outputs for the processed parts (O1 and O2). The two production lines produce two different kinds of final product.

This PN has 46 places and 39 transitions. The marking of place p_{41} (γ) represents the number of free slots of the buffer, while α (the marking of place p_1) and β (the marking of place p_{16}) represent the number of parts of type 1 and 2, respectively. Places from 42 to 46 represent the faulty behavior (in the sense that these places are marked only if a fault has occurred).

The set of observable transitions T_o is composed by transitions from t_1 to t_{13}, the set of unobservable but regular transitions T_{reg} is composed by transitions from ε_{14} to ε_{35} and the set of fault transitions is partitioned into three fault classes: $T_f^1 = \{\varepsilon_{36}\}$, $T_f^2 = \{\varepsilon_{37}\}$ and $T_f^3 = \{\varepsilon_{38}, \varepsilon_{39}\}$. The first fault class models a fault in the robot R3 that moves a part from the output buffer of machine M1 to the input buffer of machine M2, rather than putting it in the buffer B. Analogously, the second fault class models a fault in the robot R4 that moves a part from the output buffer of machine M3 to the input buffer of machine M4, rather than putting it in the buffer B. Finally, the third fault class models a fault in the AGV. In particular, when the AGV is working correctly, a processed part exits the system and a new one is admitted. If a fault in the AGV occurs parts do not exit the production lines, and are not replaced by new input parts. However, in faulty

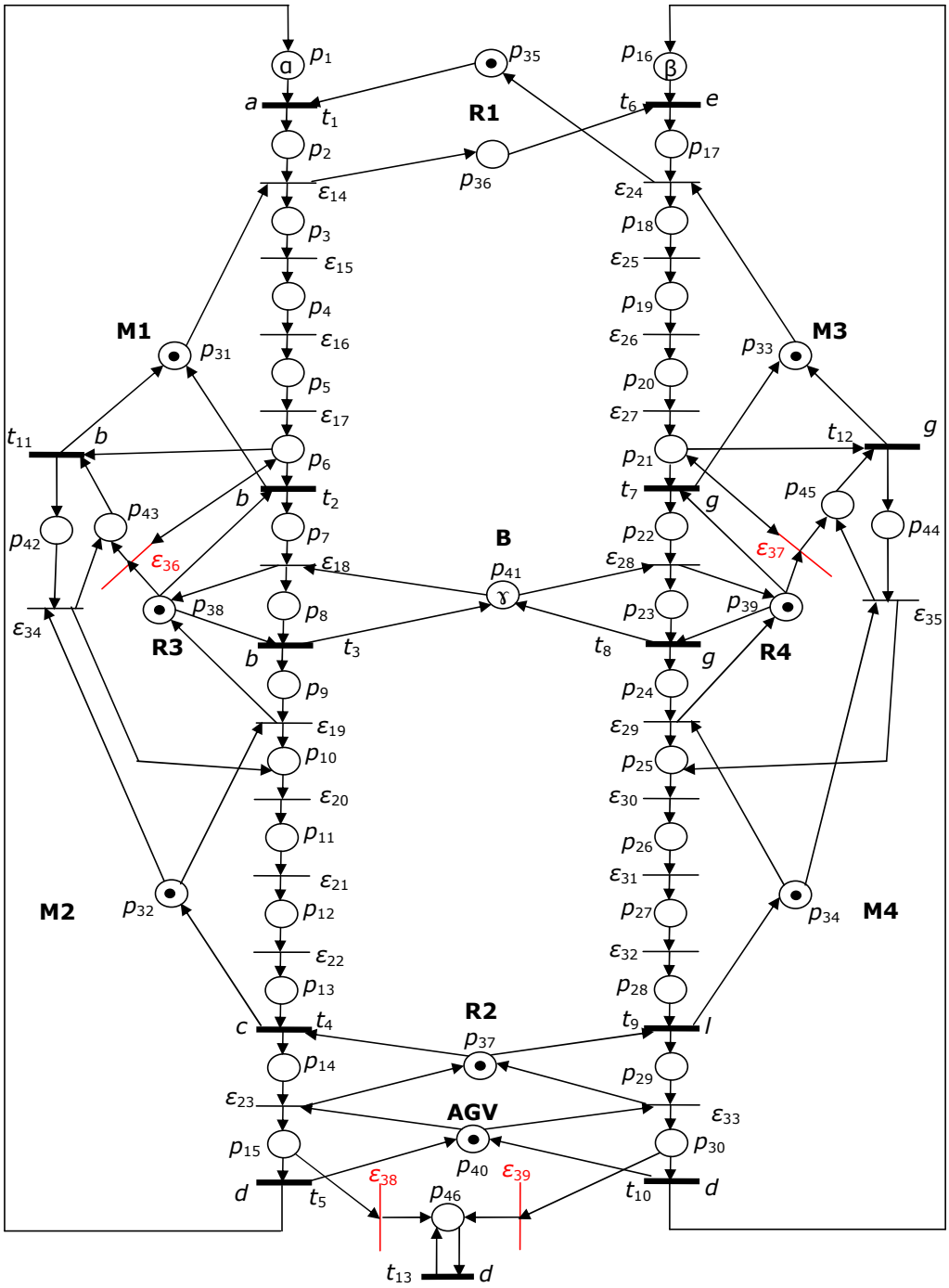


Figure 5.5: Petri net model of the manufacturing system in Figure 5.4.

behavior the sensors associated with the AGV may indefinitely produce the same signal they provide when a part regularly exits the production line.

Assume that each robot is equipped of a sensor that observes each time that the robot picks up a part. In particular, for R1 $\mathcal{L}(t_1) = a$ and $\mathcal{L}(t_6) = e$, for R2 $\mathcal{L}(t_4) = c$ and $\mathcal{L}(t_9) = l$, for R3 $\mathcal{L}(t_2) = \mathcal{L}(t_3) = \mathcal{L}(t_{11}) = b$, for R4 $\mathcal{L}(t_7) = \mathcal{L}(t_8) = \mathcal{L}(t_{12}) = g$. Moreover assume it is possible to observe each time that a part is moved by the AGV $\mathcal{L}(t_5) = \mathcal{L}(t_{10}) = \mathcal{L}(t_{13}) = d$.

In this section the results of the computation of the BRG for several initial states are presented. In particular, the cardinality of the number of nodes of the BRG, denoted as $|BRG|$, are summarized in Table 5.3 for different values of α and β.

The table also shows the cardinality of the reachability set R , i.e., $|R|$. This is an extremely important parameter to appreciate the advantage of using basis markings rather than exhaustively enumerating the set of reachable states, as it typically occurs in the automata based approaches. The value of $|R|$ has been computed using the PN tool TINA (Time PNs Analyzer) (see TINA website: http://homepages.laas.fr/bernard/tina).

— Column 1: $(\alpha + \beta)$ represents the total number of parts to be processed by the two production lines.

— Columns 2 and 3: α and β represent the number of parts to be processed in the first and second production line, respectively.

— Column 4 shows the number of nodes $|R|$ of the reachability graph.

— Column 5 shows the number of nodes $|BRG|$ of the BRG.

Table 5.3 shows how the state space of the reachability graph highly increases with the number of pallets circulating in the first and in the second production line (places p_1 and p_{16}). In particular, it increases exponentially. Note that, also in the case of the BRG the number of nodes increases exponentially, but much more slowly with respect to the cardinality of R.

Since robot R1 always starts taking one part from the first production line, all cases in which α is equal to zero present only one node corresponding to M_0 both in the BRG and in the reachability graph. Moreover all cases in which β is equal to zero present 9 nodes in the BRG and 27 nodes in the reachability graph corresponding respectively to the number of basis markings and consistent markings that can be reached when only one part is introduced in the first production line. Finally, since the first production line is perfectly symmetric to the second one the number of nodes both in the BRG and in the reachability graph does not change exchanging α with β. This is shown in Table 5.3.

For the considered PN, on the basis of the above simulations, it can be concluded that the diagnosis approach here presented is suitable from a computational point of view. In fact, thanks to the basis markings the reachability space can be described in a more compact manner.

Finally, remark that, although in this thesis the problem of diagnosability is not addressed, the manufacturing system illustrated in this section is diagnosable for any value of α, β and γ. The diagnosability of this system has been tested using the MATLAB tool in [99].

$\alpha+\beta$	α	β	$\|R\|$	$\|BRG\|$
2	2	0	27	9
2	1	1	1,640	170
2	0	2	1	1
3	3	0	27	9
3	2	1	10,260	604
3	1	2	10,260	604
3	0	3	1	1
4	4	0	27	9
4	3	1	35,098	1,343
4	2	2	62,210	2,128
4	1	3	35,098	1,343
5	5	0	27	9
5	4	1	78,404	2,294
5	3	2	205,761	4,691
5	2	3	205,761	4,691
5	1	4	78,404	2,294
6	6	0	27	9
6	5	1	131,614	3,325
6	4	2	448,306	7,963
6	3	3	655,472	10,250
6	2	4	448,306	7,963
6	1	5	131,614	3,325
7	7	0	27	9
7	6	1	186,808	4,373
7	5	2	741,035	11,503
7	4	3	1,383,391	17,273
7	3	4	1,383,391	17,273
7	2	5	741,035	11,503
7	1	6	186,808	4,373

Table 5.3: Numerical results in the case of $\gamma = 8$.

Chapter 6

Concluding remarks

In this last part of the thesis conclusion for diagnosis and testing of Petri nets are drawn.

Concluding remarks for testing Petri nets

The first part of this thesis is dedicated to the computation of synchronizing sequences in the Petri net (PN) framework. It has been shown how the classic automata approach can be applied with minor changes to the class of bounded synchronized PNs. According to this approach (*cf.* RG approach), one exploits the *reachability graph* of the net — which describes the state-space of the net and depends on the initial marking — and its corresponding *auxiliary graph* of cardinality $n \cdot (n+1)$, with n reachable markings. Clearly, n is exponential in the number of tokens the net contains.

Afterwards, we have considered a special class of synchronized PNs, called *state machine* (SM) PNs, characterized by the fact that each transition has a single input and a single output arc. For this class, we have proposed a novel approach which allows to determine a SS for nets containing 1 token (*cf.* 1-SS). This SS can be used as a building block to construct a SS for the same net in the k-token case (*cf.* k-SS). Such a sequence is called *synchronizing transition sequence* (STS). The STS approach constructs a SS with a depth-first search on the net structure — thus avoiding the state-space explosion problem encountered in the RG-approach — and verifies certain conditions over the labeling function. However, not all SSs can be obtained in this way, because of the conditions the STS approach requires.

It is then provided another approach, which is a modified implementation of the RG approach, based on an arc-pruning. In fact, for a given target place, a *modified reachability graph* (MRG) is constructed, which is the RG of the net with only 1 token and is obtained by removing all transition associated with any event in a set of *forbidden events*. Any 1-SS constructed by the way of the MRG approach leads to readily determine a k-SS for an arbitrary large k without any further computation. The MRG approach is also proven

to to be much less restrictive than the STS approach, covering then larger number of cases where a SS can be found.

A set of experimental results is also given with a tool for synchronization that we have developed. First, the three presented approaches are compared for randomly generated SM PNs. Furthermore they are applied to a family of manufacturing plants, which are not SM PNs.

Finally, our results are further extended to the case of unbounded PNs. To the best of our knowledge, the problem of determining a SS is here treated for the very first time on unbounded systems. It is proposed a simple approach, that is a generalization of the RG approach. To do so, a *modified coverability graph* construction is provided to yield a faithful representation of the PN behavior. Obviously, such an approach suffers from the fact that no finite space representation can exhaustively answer to the reachability problem. So results are restricted to bounded places.

There are several open lines for interesting future works. First of all improve the outcomes of the proposed approaches for state machine PNs. In this framework, for the multiple tokens case, the length of the SS grows polynomially with the number of token. For instance, some heuristics could be introduced to provide a lower upper bound.

We plan to extend our approach to other structures of synchronized PNs such as the marked graphs, for which there can not be any conflict but concurrency is allowed, and the free-choice PNs, where there can be concurrency and conflict, but not at the same time.

A possible case study of interest is extending the synchronized PN definition, including the always occurring event $\hat{\varepsilon}$, which is the neutral element of the monoid E^*. The always occurring event is not an input event. In fact, any transition t associated with $\hat{\varepsilon}$ is fired as soon as it is enabled by the current marking, since this event is "always occurring". In this case, the PN may present "unstable markings", i.e., markings where the net can not be obviously synchronized. In fact, when the PN reaches any of these markings, the current marking instantaneously changes under the occurrence of $\hat{\varepsilon}$. When this condition holds, a challenging problem would be to determine the minimal number of controllable transitions so that there still exists a SS.

Another work we intend to do is further generalizing synchronized PNs to take also the time into consideration.

We have also planned to deal with a more general model such as "the interpreted PNs". This input-output model combines the two models presented in this thesis, synchronized PNs and labeled PNs. In this nets an enabled transition is fired if its associated input event occurs and it produces an either observable or non-observable output event, depending on the labeling. In this framework, we want to generalize our approaches for first constructing *homing sequences* and finally taking into consideration the state verification problem, the conformance testing problem and the machine identification problem.

Concluding remarks for diagnosis of Petri nets

The second part of the thesis has been dedicated to the diagnosis of Petri nets (PNs). For the provided approach — differently to the synchronization problem — as most of the approaches dealing with diagnosis of DES, it is assumed that the faulty behavior is completely known, thus a fault model is available.

The provided diagnosis approach takes into account labeled PNs and is based on the notion of *basis markings* and *justifications*, that allow to characterize the set of markings that are consistent with the actual observation, and the set of unobservable transitions whose firing enable it. This ensures to avoid an exhaustive enumeration of the reachability set of the net both for bounded and unbounded labeled PNs whose unobservable subnet is acyclic. Here, four diagnosis states are defined, each one corresponding to a different degree of alarm.

However, in the case of bounded PNs the procedure can be simplified, by computing the so-called *basis reachability graph*. This may move off-line the most burdensome part of the computation.

Finally, a tool for the diagnosis of labeled bounded PNs has been presented and the simulations results using a net system taken from the manufacturing domain have been shown.

Bibliography

[1] R. Adler, L. Goodwyn, and Benjamin Weiss. Equivalence of topological markov shifts. *Israel Journal of Mathematics*, 27:49–63, 1977. 10.1007/BF02761605. [cited at p. 3]

[2] A.V. Aho, A.T. Dahbura, D. Lee, and M.U. Uyar. An optimization technique for protocol conformance test generation based on iuo sequences and rural chinese postman tours. *IEEE Trans. Commun.*, 39:1604—1615, Nov. 1991. [cited at p. 30]

[3] D.S. Ananichev and M. V. Volkov. Synchronizing monotonic automata. *Lecture Notes in Computer Science*, pages 111–121, 2003. [cited at p. 3]

[4] F. Basile, P. Chiacchio, and G. De Tommasi. An efficient approach for online diagnosis of discrete event systems. *IEEE Trans. Automatic Control*, 54(4):748–759, 2009. [cited at p. 36, 37]

[5] S. Baviehi and E.K.P. Chong. Automated fault diagnosis using a discrete event systems. In *1994 IEEE Int Symposium on Intelligent Control*, Ohio, USA, 1994. [cited at p. 4, 5]

[6] M. Behnam, I. Shin, T. Nolte, and M. Nolin. Sirap: A synchronization protocol for hierarchical resource sharing in real-time open systems. In *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 279–288. ACM, October 2007. [cited at p. 3]

[7] Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, and E. Shapiro. Dna molecule provides a computing machine with both data and fuel. *Proc. National Acad. Sci. USA*, 100(5):2191–2196, 2003. [cited at p. 3]

[8] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, and E. Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430 – 434, 2001. [cited at p. 3]

[9] A. Benveniste, E. Fabre, S. Haar, and C. Jard. Diagnosis of asynchronous discrete event systems: A net unfolding approach. *IEEE Trans. Automatic Control*, 48(5):714–727, 2003. [cited at p. 36, 37]

[10] G. Bochmann and G.-V. Jourdan. Testing k-safe Petri nets. In *Proceedings of the 21st IFIP WG 6.1 International Conference on Testing of Software and Communication Systems and 9th International FATES Workshop*, TESTCOM '09/FATES '09, pages 33–48, Berlin, Heidelberg, 2009. Springer-Verlag. [cited at p. 31]

[11] R.K. Boel and G. Jiroveanu. Distributed contextual diagnosis for very large systems. In *Proc. IFAC WODES'04: 7th Work. on Discrete Event Systems (Reims, France)*, pages 343–348, September 2004. [cited at p. 96]

[12] R.K. Boel and J.H. Van Schuppen. Decentralized failure diagnosis for discrete-event systems with costly communication between diagnosers. In *Proc. WODES'02: 6th Work. on Discrete Event Systems (Zaragoza, Spain)*, pages 175–181, October 2002. [cited at p. 35]

[13] P. Buchholz and P. Kemper. Hierarchical reachability graph generation for Petri nets. In *Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 660*, page 2002, 1997. [cited at p. 42]

[14] M.P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. Diagnosability analysis of unbounded Petri nets. In *Proc. 48th IEEE Conf. on Decision and Control*, Shanghai, China, dec 2009. [cited at p. 4, 37]

[15] M.P. Cabasino, A. Giua, L. Marcias, and C. Seatzu. A comparison among tools for the diagnosability of discrete event systems. In *Automation Science and Engineering (CASE), 2012 IEEE International Conference on*, pages 218–223, 2012. [cited at p. 108]

[16] M.P. Cabasino, A. Giua, and C. Seatzu. Diagnosability of bounded Petri nets. In *Proc. 48th IEEE Conf. on Decision and Control*, Shanghai, China, dec 2009. [cited at p. 4, 37, 108]

[17] M.P. Cabasino, A. Giua, and C. Seatzu. Diagnosis of discrete event systems using labeled Petri nets. In *Proc. 2nd IFAC Workshop on Dependable Control of Discrete Systems (Bari, Italy)*, June 2009. [cited at p. 27, 37]

[18] M.P. Cabasino, A. Giua, and C. Seatzu. Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9):1531–1539, 2010. [cited at p. 37, 100, 101, 102]

[19] C.G. Cassandras and S. Lafortune. *Introduction to discrete event systems, Second Edition.* Springer, 2007. [cited at p. 8, 108]

[20] W. Y. L. Chan, C. T. Vuong, and M. R. Otp. An improved protocol test generation procedure based on uios. In *Symposium proceedings on Communications architectures & protocols*, SIGCOMM '89, pages 283–294, New York, NY, USA, 1989. ACM. [cited at p. 30]

[21] M.-S. Chen, Y. Choi, and A. Kershenbaum. Approaches utilizing segment overlap to minimize test sequences. In *Proceedings of the IFIP WG6.1 Tenth International Symposium on Protocol Specification, Testing and Verification X*, pages 85–98, Amsterdam, The Netherlands, The Netherlands, 1990. North-Holland Publishing Co. [cited at p. 30]

[22] K.-T. Cheng and V.D. Agrawal. Initializability consideration in sequential machine synthesis. *IEEE Transactions on Computers*, 41:374–379, 1992. [cited at p. 79]

[23] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In *Proceedings of the 11th International Conference on Application and Theory of Petri Nets, 1990, Paris, France*, pages 387–410, 1990. NewsletterInfo: 36. [cited at p. 9]

[24] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Trans. Comput.*, 42(11):1343–1360, November 1993. [cited at p. 9]

[25] H. Cho, S.-W. Jeong, F. Somenzi, and C. Pixley. Multiple observation time single reference test generation using synchronizing sequences. In *Design Automation, 1993, with the European Event in ASIC Design. Proceedings. [4th] European Conference on*, pages 494 –498, feb 1993. [cited at p. 3]

[26] W. Chun and P. D. Amer. Improvements on uio sequence generation and partial uio sequences. In *Proceedings of the IFIP TC6/WG6.1 Twelth International Symposium on Protocol Specification, Testing and Verification XII*, pages 245–260, Amsterdam, The Netherlands, The Netherlands, 1992. North-Holland Publishing Co. [cited at p. 30]

[27] S.L. Chung. Diagnosing pn-based models with partial observable transitions. *International Journal of Computer Integrated Manufacturing*, 12(2):158–169, 2005. [cited at p. 37]

[28] J.M. Colom. The resource allocation problem in flexible manufacturing systems. In WilM.P. Aalst and Eike Best, editors, *Applications and Theory of Petri Nets 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 23–35. Springer Berlin Heidelberg, 2003. [cited at p. 56]

[29] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2 edition, 2001. [cited at p. 49]

[30] D. Corona, A. Giua, and C. Seatzu. Marking estimation of Petri nets with silent transitions. In *Proc. IEEE 43rd Int. Conf. on Decision and Control (Atlantis, The Bahamas)*, December 2004. [cited at p. 95]

[31] D. Corona, A. Giua, and C. Seatzu. Marking estimation of Petri nets with silent transitions. *IEEE Trans. Automatic Control*, 52(9):1695–1699, September 2007. [cited at p. 103]

[32] R. David and H. Alla. *Discrete, Continuous and Hybrid Petri Nets*. Springer-Verlag, 2004. [cited at p. 14, 16, 17, 24]

[33] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Discrete Events Dynamic Systems*, 10(1):33–86, January 2000. [cited at p. 5, 35]

[34] R. Devillers and L. Van Begin. Boundedness undecidability for synchronized nets. *Information Processing Letters*, 99(5):208 – 214, 2006. [cited at p. 24]

[35] E. Dijkstra. *A Discipline of Programming*. Prentice Hall, Englewood Cliffs, N.J., 1976. [cited at p. 133]

[36] Z.J. Ding, C.J. Jiang, and M.C. Zhou. Deadlock checking for one-place unbounded Petri nets based on modified reachability trees. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(3):881 –883, june 2008. [cited at p. 22]

[37] M. Domaratzki, D. Kisman, and J. Shallit. On the number of distinct languages accepted by finite automata with n states. *J. Autom. Lang. Comb.*, 7(4):469–486, September 2002. [cited at p. 66]

[38] M. Dotoli, M.P. Fanti, A. Mangini, and W. Ukovich. Identification of the unobservable behaviour of industrial automation systems by Petri nets. *Control Engineering Practice*, 2010. (In press). [cited at p. 37]

[39] M. Dotoli, M.P. Fanti, and A.M. Mangini. Fault detection of discrete event systems using Petri nets and integer linear programming. In *Proc. of 17th IFAC World Congress*, Seoul, Korea, July 2008. [cited at p. 36, 37]

[40] D. Eppstein. Reset sequences for monotonic automata. *SIAM J. Computing*, 19:500–510, 1990. [cited at p. 3, 29, 32]

[41] J. Ezpeleta, J.M. Colom, and J. Martinez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *Robotics and Automation, IEEE Transactions on*, 11(2):173–184, 1995. [cited at p. 56]

[42] J. Ezpeleta and L. Recalde. A deadlock avoidance approach for non-sequential resource allocation systems. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 34(1):93–101, 2004. [cited at p. 56]

[43] J. Ezpeleta, F. Tricas, F. Garcia-Valles, and J.M. Colom. A banker's solution for deadlock avoidance in fms with flexible routing and multiresource states. *Robotics and Automation, IEEE Transactions on*, 18(4):621–625, 2002. [cited at p. 56]

[44] A. Farooq, H. Hejiao, and W. Xiaolong. A technique for reachability graph generation for the Petri net models of parallel processes. *International Journal of Electrical and Electronics Engineering*, 3:5:298–302, 2009. [cited at p. 42]

[45] A. Finkel. The minimal coverability graph for Petri nets. In *Papers from the 12th International Conference on Applications and Theory of Petri Nets: Advances in Petri Nets 1993*, pages 210–243, London, UK, UK, 1993. Springer-Verlag. [cited at p. 22]

[46] G.-V. Jourdan and G.v. Bochmann. On testing 1-safe Petri nets. In *Theoretical Aspects of Software Engineering, 2009. TASE 2009. Third IEEE International Symposium on*, pages 275 –281, 29-31 2009. [cited at p. 31]

[47] G. Gaderer, P. Loschmidt, and T. Sauter. Improving fault tolerance in high-precision clock synchronization. *Industrial Informatics, IEEE Transactions on*, 6(2):206 –215, may 2010. [cited at p. 3]

[48] E. Garcia, A. Correcher, F. Morant, E. Quiles, and R. Blasco. Modular fault diagnosis based on discrete event systems. *Discrete Event Dynamic Systems*, 15(3):237–256, 2005. [cited at p. 4, 5]

[49] S. Gaubert and A. Giua. Petri net languages and infinite subsets of $\mathcal{N}^m$. *J. of Computer and System Sciences*, 59(3):373–391, april 1999. [cited at p. 9, 26, 89]

[50] G. Geeraerts, J.-F. Raskin, and L. Van Begin. On the efficient computation of the minimal coverability set for Petri nets. *International Journal of Foundations of Computer Science*, 21(02):135–165, 2010. [cited at p. 22]

[51] S. Genc and S. Lafortune. Distributed diagnosis of place-bordered Petri nets. *IEEE Trans. on Automation Science and Engineering*, 4(2):206–219, 2007. [cited at p. 36]

[52] M. Ghazel, A. Togueni, and M. Bigang. A monitoring approach for discrete events systems based on a time Petri net model. In *Proc. of 16th IFAC World Congress*, Prague, Czech Republic, July 2005. [cited at p. 36]

[53] A. Gill. State-identification experiments in finite automata. *Information and Control*, 4:132 – 154, 1961. [cited at p. 30]

[54] A. Gill. *Introduction to the theory of finite-state machines.* McGraw-Hill, New York, 1962. [cited at p. 30]

[55] A. Giua, F. DiCesare, and M. Silva. Generalized mutual exclusion contraints on nets with uncontrollable transitions. In *Systems, Man and Cybernetics, 1992., IEEE International Conference on*, pages 974 –979 vol.2, oct 1992. [cited at p. 58]

[56] A. Giua and C. Seatzu. Observability of place/transition nets. *Automatic Control, IEEE Transactions on*, 47(9):1424 – 1437, sep 2002. [cited at p. 88]

[57] A. Giua, C. Seatzu, and D. Corona. Marking estimation of Petri nets with silent transitions. *Automatic Control, IEEE Transactions on*, 52(9):1695 –1699, sept. 2007. [cited at p. 31]

[58] S. M. Gobershtein. Check words for the states of a finite automaton. *Cybernetics and Systems Analysis*, 10:54–58, 1974. 10.1007/BF01069018. [cited at p. 30]

[59] C.N. Hadjicostis and G.C. Veghese. Monitoring discrete event systems using Petri net embeddings. *Lecture Notes in Computer Science*, 1639:188–207, 1999. [cited at p. 36]

[60] F. Hennie. *Finite-State Models for Logical Machines.* New York: John Wiley, 2 edition, 1968. [cited at p. 29, 32]

[61] F. C. Hennie. Fault detecting experiments for sequential circuits. In *Switching Circuit Theory and Logical Design, 1964 Proceedings of the Fifth Annual Symposium on*, pages 95 –110, nov. 1964. [cited at p. 30]

[62] R. M. Hierons. Using a minimal number of resets when testing from a finite state machine. *Inf. Process. Lett.*, 90:287–292, June 2004. [cited at p. 3]

[63] E.P. Hsieh. Checking experiments ror sequential machines. *IEEE Transactions on Computers*, 20:1152–1166, 1971. [cited at p. 30]

[64] Z. Huang, V. Chandra, S. Jiang, and R. Kumar. Modeling discrete event systems with faults using a rules based modeling formalism. *Mathematical and Computer Modelling of Dynamical Systems*, 9(3):233–254, 2003. [cited at p. 94]

[65] M. D. Jeng, X. Xie, and M. Y. Peng. Process nets with resources for manufacturing modeling and their analysis. *Robotics and Automation, IEEE Transactions on*, 18(6):875–889, 2002. [cited at p. 56]

[66] K. Jensen. Coloured Petri nets: A high level language for system design and analysis. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 342–416. Springer Berlin Heidelberg, 1991. [cited at p. 9]

[67] G. Jiroveanu and R.K. Boel. Contextual analysis of Petri nets for distributed applications. In *16th Int. Symp. on Mathematical Theory of Networks and Systems (Leuven, Belgium)*, July 2004. [cited at p. 96]

[68] H. Jürgensen. Synchronization. *Information and Computation*, 206:1033–1044, September 2008. [cited at p. 3]

[69] R. M. Karp and R. E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147 – 195, 1969. [cited at p. 22, 23, 81]

[70] Z. Kohavi. *Switching and Finite Automata Theory.* The McGraw-Hill College, 2 edition, 1978. [cited at p. 29, 30, 32]

[71] S. Lai, D. Nessi, M.P. Cabasino, A. Giua, and C. Seatzu. A comparison between two diagnostic tools based on automata and Petri nets. In *Proc. WODES'08: 9th Work. on Discrete Event Systems (Göteborg, Sweden)*, pages 144–149, May 2008. [cited at p. 37]

[72] C. Lakos. The challenge of object orientation for the analysis of concurrent systems. In Javier Esparza and Charles Lakos, editors, *Application and Theory of Petri Nets 2002*, volume 2360 of *Lecture Notes in Computer Science*, pages 59–67. Springer Berlin Heidelberg, 2002. [cited at p. 10]

[73] K. Lautenbach and P. S. Thiagarajan. Analysis of a resource allocation problem using Petri nets. In J.C. Syre, editor, *Proc. of the 1st European Conference on Parallel and Distributed Processing*, pages 260–266, Toulouse, 1979. Cepadues Editions. [cited at p. 56]

[74] D. Lee and K. Sabnani. Reverse-engineering of communication protocols. In *Network Protocols, 1993. Proceedings., 1993 International Conference on*, pages 208 –216, oct 1993. [cited at p. 31]

[75] D. Lee and M. Yannakakis. Testing finite-state machine: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994. [cited at p. 30]

[76] D. Lee and M. Yannakakis. Principles and methods of testing finite state machine – a survey. *Proceedings of the IEEE*, 84(8):1090–1123, August 1996. [cited at p. 28, 39, 84]

[77] D. Lefebvre and C. Delherm. Diagnosis of DES with Petri net models. *IEEE Trans. on Automation Science and Engineering*, 4(1):114–118, 2007. [cited at p. 36]

[78] L. Li and C.N. Hadjicostis. Minimum initial marking estimation in labeled Petri nets. In *American Control Conference, 2009. ACC '09.*, pages 5000 –5005, june 2009. [cited at p. 31]

[79] Z. Li and M. C. 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. [cited at p. 56]

[80] Z. W. Li and M. C. Zhou. *Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach.* Springer Publishing Company, Incorporated, 1st edition, 2009. [cited at p. 56]

[81] F. Lin. Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems*, 4(2):197–212, 1994. [cited at p. 34]

[82] F. Lin, J. Markee, and B. Rado. Design and test of mixed signal circuits: a discrete event approach. In *Proc. 32rd IEEE Conf. on Decision and Control*, pages 246–251, dec 1993. [cited at p. 34]

[83] Y. Lu and I. Pomeranz. Synchronization of large sequential circuits by partial reset. In *VLSI Test Symposium, 1996., Proceedings of 14th*, pages 93 –98, apr-1 may 1996. [cited at p. 3]

[84] J. Lunze and J. Schroder. Sensor and actuator fault diagnosis of systems with discrete inputs and outputs. *IEEE Transactions on Systems, Man, and CyberneticsóPart B: Cybernetics*, 34(3):1096–1107, April 2004. [cited at p. 4, 5]

[85] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley series in parallel Computing, John Wiley & sons, 1995. [cited at p. 10]

[86] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In Girault, C. and Reisig, W., editors, *Informatik-Fachberichte 52: Application and Theory of Petri Nets: Selected Papers from the First and Second European Workshop on Application and Theory of Petri Nets, Strasbourg, September, 1980, Bad Honnef, September, 1981*, pages 301–310. Springer-Verlag, 1982. [cited at p. 96]

[87] P. M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, University of California, Irvine, 1974. AAI7511026. [cited at p. 10]

[88] S. Micha. A strong connectivity algorithm and its applications to data flow analysis. *Computers and Mathematics with Applications*, 7(1):67–72, 1981. [cited at p. 133]

[89] D.L. Mills. Internet time synchronization: the network time protocol. *Communications, IEEE Transactions on*, 39(10):1482 –1493, oct 1991. [cited at p. 3]

[90] P.E. Miyagi and L.A.M. Riascos. Modeling and analysis of fault-tolerant systems for machining operations based on Petri nets. *Control Engineering Practice*, 14(4):397–408, 2010. [cited at p. 36]

[91] M. Moalla, J. Pulou, and J. Sifakis. Synchronized Petri nets : A model for the description of non-autonomous systems. In J. Winkowski, editor, *Mathematical Foundations of Computer Science 1978*, volume 64 of *Lecture Notes in Computer Science*, pages 374–384. Springer Berlin / Heidelberg, 1978. [cited at p. 9, 17, 18]

[92] E. F. Moore. Gedanken-experiments on sequential machines. *Automata Studies, Annals of Mathematical Studies*, 34:129 – 153, 1956. [cited at p. 28, 29, 30]

[93] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541 –580, apr 1989. [cited at p. 14, 15, 31]

[94] B. K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *SFCS '86: Proc. of the 27th Annual Symposium on Foundations of Computer Science*, pages 132–142, Washington, DC, USA, 1986. IEEE Computer Society. [cited at p. 2, 3]

[95] B. K. Natarajan. Some paradigms for the automated design of parts feeders. *The International Journal of Robotics Research*, 8 (6):89–109, 1989. [cited at p. 3]

[96] T. Nolte, Insik Shin, M. Behnam, and M. Sjodin. A synchronization protocol for temporal isolation of software components in vehicular systems. *Industrial Informatics, IEEE Transactions on*, 5(4):375 –387, nov. 2009. [cited at p. 3]

[97] Jonghun Park and Spyros A. Reveliotis. Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings. *IEEE Transactions on Automatic Control*, 46:1572–1583, 2000. [cited at p. 56]

[98] C.A. Petri. *Kommunikation mit Automaten*. PhD thesis, University of Bonn, 1962. [cited at p. 9]

[99] M. Pocci. The MATLAB toolbox is available at the following web address: `http://www.diee.unica.it/giua/TESI/09_Marco.Pocci/PN_DIAG.zip`, 2009. [cited at p. 4, 109, 112]

[100] M. Pocci. The MATLAB toolbox is available at the following web address: `http://www.lsis.org/poccim/PN_SYNCHRO.zip`. September, 2012. [cited at p. 60, 65]

[101] J. Prock. A new tecnique for fault detection using Petri nets. *Automatica*, 27(2):239–245, 1991. [cited at p. 35]

[102] J. Provost, J.-M. Roussel, and J.-M. Faure. Test sequence construction from SFC specification. In *Proceedings of the 2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'09)*, pages 341–346, Bari, Italie, Jun 2009. [cited at p. 31]

[103] J. Provost, J.-M. Roussel, and J.-M. Faure. Translating Grafcet specifications into Mealy machines for conformance test purposes. *Control Engineering Practice*, Dec 2010. [cited at p. 31]

[104] J. Provost, J.-M. Roussel, and J.-M. Faure. Testing Programmable Logic Controllers from Finite State Machines specification. In *3rd International Workshop on Dependable Control of Discrete Systems - DCDS 2011*, Saarbrücken, Allemagne, June 2011. 6 pages. [cited at p. 31]

[105] A. Ramirez-Treviño, E. Ruiz-Beltràn, I. Rivera-Rangel, and E. Lopez-Mellado. Online fault diagnosis of discrete event systems. A Petri net-based approach. *IEEE Trans. on Automation Science and Engineering*, 4(1):31–39, 2007. [cited at p. 36]

[106] W. Reisig and J. Vautherin. An algebraic approach to high level Petri nets. In *Proceedings of the Eighth European Workshop on Application and Theory of Petri Nets*, pages 51–72. Universidad de Zaragoza (Spain), 1987. [cited at p. 9]

[107] P.-A. Reynier and F. Servais. Minimal coverability set for Petri nets: Karp and miller algorithm with pruning. In *Proceedings of the 32nd international conference on Applications and theory of Petri Nets*, PETRI NETS'11, pages 69–88, Berlin, Heidelberg, 2011. Springer-Verlag. [cited at p. 22]

[108] A. Roman. Synchronizing finite automata with short reset words. *Applied Mathematics and Computation*, 209(1):125 – 136, 2009. Special Issue International Conference on Computational Methods in Sciences and Engineering 2005 (ICCMSE-2005). [cited at p. 29, 65]

[109] A. Roman. The np-completeness of the road coloring problem. *Inf. Process. Lett.*, 111:342–347, March 2011. [cited at p. 4]

[110] K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 15(4):285 – 297, 1988. [cited at p. 30]

[111] M. Sampath. *A Discrete Event Systems Approach to Failure Diagnosis*. PhD thesis, The University of Michigan, Ann Arbor, Michigan, 1995. [cited at p. 94]

[112] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete-event systems. *IEEE Trans. Automatic Control*, 43(7):908–929, July 1998. [cited at p. 5, 34]

[113] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Trans. Automatic Control*, 40 (9):1555–1575, 1995. [cited at p. 5, 34, 35, 37]

[114] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. Control Systems Technology*, 4(2):105–124, 1996. [cited at p. 34, 37]

[115] Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, New York, NY, USA, 1 edition, 2008. [cited at p. 64, 66]

[116] Y.-N. Shen, F. Lombardi, and A.T. Dahbura. Protocol conformance testing using multiple uio sequences. *Communications, IEEE Transactions on*, 40(8):1282 –1287, aug 1992. [cited at p. 30]

[117] D.P. Sidhu and T.K. Leung. Formal methods for protocol testing: A detailed study. *IEEE Transactions on Software Engineering*, 15:413–426, 1989. [cited at p. 30]

[118] M. N. Sokolovskii. Diagnostic experiments with automata. *Cybernetics and Systems Analysis*, 7:988–994, 1971. 10.1007/BF01068822. [cited at p. 30]

[119] V.S. Sreenivas and M.A. Jafari. Fault detection and monitoring using time Petri nets. *IEEE Trans. Systems, Man and Cybernetics*, 23(4):1155–1162, 1993. [cited at p. 36]

[120] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. [cited at p. 133, 134]

[121] A. N. Trahtman. The existence of synchronizing word and cerny conjecture for some finite automata. In *Second Haifa Workshop on Graph Theory, Combinatorics and Algorithms*, pages 17–20, Haifa, June 2002. [cited at p. 3]

[122] A. N. Trahtman. Some results of implemented algorithms of synchronization. In *10th Journees Mentoises d'inform.*, Liege, Belgium, September 8-11 2004. [cited at p. 29, 32]

[123] A. N. Trahtman. The road coloring problem. *Israel Journal of Mathematics*, 172:51–60, 2009. 10.1007/s11856-009-0062-5. [cited at p. 3]

[124] F. Tricas, F. Garcia-Valles, J.M. Colom, and J. Ezpeleta. A Petri net structure-based deadlock prevention solution for sequential resource allocation systems. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 271–277, 2005. [cited at p. 56]

[125] J. Černý. Poznámka k homogénnym experimentom s konečnými automatmi. (slovak) [a note on homogeneous experiments with finite automata]. *Mathematica Slovaca*, 3:208–216, 1964. [cited at p. 3]

[126] T. Ushio, L. Onishi, and K. Okuda. Fault detection based on Petri net models with faulty behaviors. In *Proc. SMC'98: IEEE Int. Conf. on Systems, Man, and Cybernetics (San Diego, CA, USA)*, pages 113–118, October 1998. [cited at p. 37]

[127] M. Van den Heuvel, R. Bril, and J. Lukkien. Transparent synchronization protocols for compositional real-time systems. *Industrial Informatics, IEEE Transactions on*, PP(99):1, 2011. [cited at p. 3]

[128] N. Viswanadham and T. L. Johnson. Fault detection and diagnosis of automated manufacturing systems. In *Proc. 27th IEEE Conf. on Decision and Control*, pages 2301–2306, Austin, Texas, December 1988. [cited at p. 4, 5]

[129] F.-Y. Wang, Y. Gao, and M.C. Zhou. A modified reachability tree approach to analysis of unbounded Petri nets. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 34(1):303 – 308, feb. 2004. [cited at p. 22]

[130] Y. Wen and M. Jeng. Diagnosability analysis based on T-invariants of Petri nets. In *Networking, Sensing and Control, 2005. Proceedings, 2005 IEEE.*, pages 371– 376, March 2005. [cited at p. 38]

[131] Y. Wen, C. Li, and M. Jeng. A polynomial algorithm for checking diagnosability of Petri nets. In *Proc. SMC'05: IEEE Int. Conf. on Systems, Man, and Cybernetics*, pages 2542–2547, October 2005. [cited at p. 38]

[132] Y. Wu and C.N. Hadjicostis. Algebraic approaches for fault identification in discrete-event systems. *IEEE Trans. Robotics and Automation*, 50(12):2048–2053, 2005. [cited at p. 36]

[133] X. Xie and M. D. Jeng. Ercn-merged nets and their analysis using siphons. *Robotics and Automation, IEEE Transactions on*, 15(4):692–703, 1999. [cited at p. 56]

[134] E.C. Yamalidou, E.D. Adamides, and D. Bovin. Optimal failure recovery in batch processing using Petri net models. *Proceedings of the 1992 American Control Conference*, 3:1906–1910, 1992. [cited at p. 31]

[135] E.C. Yamalidou and J.C. Kantor. Modeling and optimal control of discrete-event chemical processes using Petri nets. *Computers & Chemical Engineering*, 15(7):503 – 519, 1991. [cited at p. 31]

[136] S. H. Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in discrete-event systems: framework and model reduction. *IEEE Trans. Automatic Control*, 48(7):1199–1212, July 2003. [cited at p. 5]

[137] S. Hashtrudi Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in discrete-event systems: framework and model reduction. *IEEE Trans. Automatic Control*, 48(7):1199–1212, July 2003. [cited at p. 35]

[138] M.C. Zhou and F. DiCesare. *Petri net synthesis for discrete event control manufacturing systems*. Kluwer, 1993. [cited at p. 110]

[139] H. Zhu and X. He. A methodology of testing high-level Petri nets. *Information and Software Technology*, 44(8):473 – 489, 2002. [cited at p. 31]

List of Publications Related to the Thesis

Published papers

Journal papers

- M.P. Cabasino, A. Giua, M. Pocci, C. Seatzu, *Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems.* Control Engineering Practice, Vol. 19, Issue 9, pp. 989-1001, September 2011. (Relation to Chapter 5)
- M. Pocci, I. Demongodin, N. Giambiasi, A. Giua, *Testing experiments on synchronized Petri nets,* IEEE Transactions on Automation Science and Engineering, T-ASE. (Relation to Chapter 3)

Conference papers

- Marco Pocci, Isabel Demongodin, Norbert Giambiasi, Alessandro Giua, *Testing Discrete Event Systems: Synchronizing Sequences using Petri Nets,* 2010 European Modeling & Simulation Symposium, EMSS10. (Fes, Morocco), Oct 2010. (Relation to Chapter 3)
- Marco Pocci, Isabel Demongodin, Norbert Giambiasi, Alessandro Giua, *Synchronizing Sequences On Not Strongly Connected Petri Nets,* Symposium On Theory of Modeling and Simulation, DEVS/TMS'11 (Boston, MA, USA), April 2011. (Relation to Chapter 3)
- Marco Pocci, Isabel Demongodin, Norbert Giambiasi, Alessandro Giua, *Séquences de synchronisation sur les réseaux de Petri,* 4èmes Journées Doctorales MACS (Marseille, France), June 2011. (Relation to Chapter 3)
- M. Pocci, I. Demongodin, N. Giambiasi, A. Giua, *A new algorithm to compute synchronizing sequences for synchronized Petri nets,* the IEEE Tencon Conference for Region 10, TENCON (Xi'an, Shaanxi, China), October 22-25, 2013. (Relation to Chapter 3)

Submitted papers

Conference papers

- M. Pocci, I. Demongodin, N. Giambiasi, A. Giua, *Testing experiments on unbounded systems: synchronizing sequences using Petri nets,* 12th IFAC - IEEE International Workshop on Discrete Event Systems, WODES'14 (Cachan, France), May 14-16, 2014. (Relation to Chapter 4)

Appendices

Appendix A

Strongly connected component decomposition on automata

When operating on automata, it can be useful to work over its graphical representation, i.e. its directed graph.

A directed graph can be partitioned in its strongly connected components. A component is called strongly connected if there is a path from each of its vertex to every other of its vertices. The strongly connected components of a directed graph $\mathcal{G}$ are its maximal strongly connected subgraphs. If each strongly connected component is contracted to a single vertex, the resulting graph is a directed acyclic graph, the condensation of $\mathcal{G}$.

The strongly connected components of a directed graph can be computed efficiently by the way of three main approaches: the *Kosaraju's algorithm* [88], also known as the Kosaraju-Sharir algorithm, the *Tarjan's algorithm* [120] and the *path-based strong component algorithm* [35].

In practice the last two, the Tarjan's and the path-based algorithm, are preferred, since they terminate their computation by only one depth-first search rather than two.

Here, we provide the Tarjan's algorithm — since both previously presented algorithms are equivalently efficient — to decompose the graph of a given automaton in its maximally strongly connected components.

The algorithm is starts from a random node, the one having the lowest index, and conducts depth-first searches on every not explored nodes. The strongly connected components form sets of subtrees.

Each time a node is visited, it is placed on an initially empty stack S, that keeps track of the visiting order. When the search returns from a subtree, the nodes are taken from the stack and it is determined whether each node is the root of a strongly connected component. If a node is the root of a strongly connected component, then it and all of the nodes taken off before it form that strongly connected component.

Note that the property of a node of being a root node applies only to the algorithm itself, since outside of it a strongly connected component has no single "root" node.

The root node is simply the first node of the strongly connected component which is encountered twice during the depth-first traversal. When a node is identified as the root node (step 6.), once recursion on its successors has finished, all nodes on the stack from the root upwards form a complete strongly connected component. To find the root, each node is given a depth search index, called *v.index*. Such an index numbers the nodes in the order they are encountered. In addition, each node is assigned a value *v.lowlink* that is equal to the smallest index of some node reachable from v, and is always less than or equal to v.index if no other node is reachable from v. Therefore v is the root of a strongly connected component if and only if $v.lowlink == v.index$. The value *v.lowlink* is computed during the depth first search such that it is always known when needed.

Algorithm A.1. *[120] (Tarjan's algorithm for strongly connected component decomposition)*
Input: *a graph* $\mathcal{G} = (V, E)$.
Output: *the set of strongly connected components defined as partition of the set of vertices* V.

1. *index := 0*

2. *S := empty*

3. ***For*** *each vertex* $v \in V$, *do*

  3.1 ***if*** *v.index is not defined*

    3.1.1 *strongconnect(v)*

function *strongconnect(v)*

1. *v.index := index* *[Setting the depth index for v to the smallest unused index]*

2. *v.lowlink := index*

3. *index := index + 1*

4. *S.push(v)*

5. ***For*** *each connected vertex* $v, w \in V$, *do* *[Analyzing successors of v]*

  5.1. ***if*** *w.index is not defined* *[Successor w has not yet been visited; recurse on it]*

    5.1.1. *strongconnect(w)*

    5.1.2. *v.lowlink := min(v.lowlink, w.lowlink)*

  5.2. ***else if*** $w \in S$ *[Successor w is in stack S and hence in the current SCC]*

    5.2.1. *v.lowlink := min(v.lowlink, w.index)*

6. ***If*** *v.lowlink = v.index* *[v is a root node, pop the stack and generate an SCC]*

6.1. *start a new strongly connected component.*

6.2. ***While*** $w \neq v$

6.2.1. $w := S.pop();$

6.2.2. *add w to current strongly connected component;*

6.3. *output the current strongly connected component*

end function ■

S is the stack, which starts out empty (step **2.**) and stores the history of nodes explored but not yet committed to a strongly connected component (step **4.**).

The outermost loop (step **5.**) searches each node that has not yet been visited, ensuring that nodes which are not reachable from the first node are still eventually traversed. The function *strongconnect* computes a single depth-first search of the graph, determining every successors of node v, and returning all strongly connected components of that subgraph.

When each recursion is done, if its low-link is still set to its index, then it is the root node of a strongly connected component, formed by all of the nodes above it on the stack (step **6.**). The nodes are popped out of the stack only when an entire strongly connected component has been found (step **6.2.1.**). At this moment all of these nodes are presented as a strongly connected component.

Test and diagnosis of Discrete Event Systems using Petri Nets

State-identification experiments are designed to identify the final state of a *discrete event system* (DES) when its initial state is unknown. A classical solution to this problem, assuming the DES has no observable outputs, consists in determining a *synchronizing sequence*, i.e. a sequence of input events that drives the system to a known state. This problem was addressed and essentially solved in the 60' using finite state machines or automata. The main objective of this thesis is to use Petri nets for solving the state-identification problem more efficiently and for a wider class of systems, with possibly infinite states.
We start showing that the classical method based on automata for constructing a synchronizing sequence can be easily applied with minor changes to *synchronized Petri nets*, a class of non-autonomous nets where each transition is associated with an input event. The proposed approach is fairly general and it works for arbitrary bounded nets with a complexity that is polynomial with the size of the state space. However, as most of the problem solving methods for DES, it incurs in the well-known state-space explosion problem.
Looking for more efficient solutions, we begin by considering a subclass of Petri nets called *state machines*. We first consider strongly connected state machines and propose a framework for synchronizing sequence construction that exploits structural criteria, and thus does not require an exhaustive enumeration of the state space of the net. These results are further extended to larger classes of nets, namely non strongly connected state machines and nets containing state machine subnets. Finally we consider the class of unbounded nets that describe infinite state systems: even in this case we are able to compute a set of sequences to synchronize the marking of bounded places. A Matlab toolbox implementing all approaches previously described has been designed and applied to a series of benchmarks.
An additional problem addressed in this thesis and partially related to state-identification, is the *failure diagnosis* of DES, i.e., the process of identifying the occurrence of a fault based on observable symptoms. Our work is centered on diagnosis of DES using model-based methods where a common assumption requires that a fault model be available. The system is represented by a *labeled Petri net*, in which transitions can be either observable or silent; a fault is modeled by a silent transition. We consider an efficient approach, based on *basis markings*, to compute a diagnosis state which produces different degrees of alarm, such as "normal" or "faulty" or "uncertain". A Matlab toolbox for solving the diagnosis problem and the related problem of diagnosability has been designed, and tested on a parametric example.

Test et diagnostic des systèmes à événements discrets par les réseaux de Petri

L'exécution d'un test d'identification d'état d'un *système à événement discret* (SED) a pour but d'en identifier l'état final, lorsque son état initial est inconnu. Une solution classique à ce problème, en supposant que le SED n'ait pas de sorties observables, consiste à déterminer une *séquence de synchronisation*, c.à-d., une séquence d'événements d'entrée qui conduit le système sur un état connu. Ce problème a été abordé et complètement résolu dans les années 60' à l'aide de *machines à états finis* ou d'*automates*. L'objectif principal de cette thèse est d'utiliser les réseaux de Petri dans le but d'obtenir une résolution plus optimal du problème d'identification d'état et pour une plus large classe de systèmes, dont le nombre d'états pourrait être infini.
Dans une première approche, nous montrons que la méthode classique des automates pour la construction des séquences de synchronisation peut être aisément étendue — par des modifications mineures — aux *réseaux de Petri synchronisés*. Pour cette classe de réseaux non-autonomes, toute transition est associée à un événement d'entrée.
L'approche proposée est assez générale, dans la mesure où elle s'applique à des réseaux bornés arbitraires. Cependant, comme la plupart des méthodes s'appliquant aux SEDs, elle engendre le problème classique d'explosion combinatoire du nombre d'états. Dans le but d'obtenir des des meilleures solutions, nous considérons une classe spéciale de réseaux de Petri, connue sous le nom de *graphes d'état* (GdE). Pour ces réseaux, nous considérons d'abord les GdE fortement connexes et nous proposons deux approches pour la construction de séquences de synchronisation. Ces approches exploitent les propriétés structurelles du réseau et ne nécessitent pas ainsi une énumération exhaustive de l'espace d'état. Ces résultats s'étendent par la suite aux GdE non fortement connexes et puis à tout RdP synchronisés composés de GdE. Enfin, nous considérons la classe des réseaux de Petri non bornés et proposons de construire des séquences qui synchronisent le marquage des places non bornées. Une boîte à outils fournit toutes les approches précédemment décrites et est appliquée à des différents bancs d'essai.
Dans cette thèse, nous traitons aussi le diagnostic de fautes de SED, c.à-d., le processus d'identification des causes d'un échec basé sur l'observation de symptômes. Nous nous intéressons au diagnostic de SED à l'aide de méthodes analytiques basées sur modèle, sous l'hypothèse que le modèle de faute soit disponible. Le système est alors modélisé par un *réseau de Petri étiqueté*, pour lequel les transitions sont *observables* si elles sont associées à un événement de sortie, ou *silencieuses*; tout comportement fautif est modélisé par une transition silencieuse. Nous considérons une approche efficace basée sur la notion de *marquages de base* qui calcule un état de diagnostic, qui caractérise différents degrés d'alarme, tels que "normal", "fautif" ou "incertain". Une boîte à outils pour la résolution du problème de diagnostic et de celui de la diagnosticabilité est fournie et testée pour un exemple paramétrique.