Petri Nets as Discrete Event Models for Supervisory Control

by Alessandro Giua

A Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirement for the Degree of Doctor of Philosophy. Major Subject: Computer and Systems Engineering.

Rensselaer Polytechnic Institute (Troy, New York)

July 1992

Este, que ves, engaño colorido,
que del arte ostentando los primores,
con falsos silogismos de colores
es cauteloso engaño del sentido.

This coloured counterfeit that thou beholdest,
vainglorious with the excellencies of art,
is, in fallacious syllogisms of colour,
nought but a cunning dupery of sense.

Inundación Castálida, II, 1-4

Juana de Asbaje y Ramírez,
Sor Juana Inés de la Cruz (1651-1695)

Contents

Acknowledgement

Frank DiCesare has guided me in a spirit of friendship through my Master's and Ph.D. thesis, providing learning, encouragement, and support all at once.

Manuel Silva has given me the wonderful opportunity of visiting the University of Zaragoza and of working with his group. Part of this research has been inspired by him.

Alan A. Desrochers, David R. Musser, and Badrinath Roysam have provided additional guidance as members of my doctoral committee.

The Petri net group meetings at Rensselaer have been a fruitful learning experience. I acknowledge the many discussions with Padma Akella, Tiehua Cao, Xin Chen, Mu-Der Jeng, Jagdish Joshi, Hauke Jungnitz, Jongwook Kim, Inseon Koh, Doo Yong Lee, Jim Watson, Mengchu Zhou.

I am grateful to the Computer Integrated Manufacturing project of the Center for Manufacturing Productivity and Technology Transfer which has supported the research of this thesis during the last two years.

The years of my graduate studies have been a period of freedom and carelessness, marking the transition between youth and maturity. Never will love be more poignant or friends more dear; the memories will last forever. For this I have to thank Philippe Jeanne, Thanos Tsolkas, Ermanno Astorino, Antonio De Dominicis, Gloria-Deo Agbasi, Hauke Jungnitz, Giampiero Beroggi and Penny Spring, Agostino Abbate and Josefina Quiles, Didier Laporte, Elisabetta Bruzzone, Nora Si-Ahmed, Amitava Maulik, Persefone Vouyoukas, Doo Yong Lee, Josep Tornero, Markus Vincze, Michael Eppinger, Rolf Münger, Keith Hanna, Alejandro Beascoechea, Markus Hoerler, Carlos Garcia, Leonardo Lanari, Scott Bieber, Brady Richter, Cristina Gesto, Andreas Glatzl, Jose Neira, Maria Angeles Salas, Antonio Ramírez, Enrique Teruel, Joaquin Ezpeleta, Bruno Gaujal, and Rogerio Rodriguez.

Abstract

Discrete event systems represents a new field of control theory of increasing importance in the domains of manufacturing, communications, and robotics. *Supervisory Control* theory, based on formal languages, is a well established framework for the study of discrete event systems.

The thesis discusses the use of *Petri nets* in Supervisory Control. Place/Transition nets have been used by several authors to represent discrete event systems. In our approach, the *supervisor*, i.e., the control agent that restricts the behavior of a system within a legal behavior, is represented as a Place/Transition net as well. The advantage of such a supervisor, as opposed to a supervisor given as a feedback function, is that a closed loop model of the controlled system may be constructed and analyzed using the techniques pertaining to Petri net models.

We show that a Petri net supervisor may not exist if the system's behavior or the legal behavior are nonregular Petri net languages. By defining a new class of Petri net languages, called *deterministic P-closed*, it is possible to derive necessary and sufficient conditions for the existence of supervisors as nets.

The thesis presents an algorithm, based on the *concurrent composition* operator, for the design of Petri net supervisors and discusses how a composed net may be validated. In a first approach, based on incidence matrix analysis, important properties of the net, such as controllability or such as the absence of blocking states, may be studied by Integer Programming techniques. In a second approach, we consider a class of specifications, called *generalized mutual exclusion constraints*, and discuss several possible structures for the supervisor capable of enforcing them.

Chapter 1

INTRODUCTION

1.1 Introduction

The object of the study of traditional control theory have been systems of continuous and synchronous discrete variables, modeled by differential or difference equations. However, as the scope of control theory is being extended into the domains of manufacturing, robotics, computer and communication networks, and so on, there is an increasing need for different models, capable of describing systems that evolve in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events. Such systems, whose states have logical or symbolic, rather than numerical, values that change in response to events which may also be described in non-numerical terms, are called *discrete event systems* (DES) [Ramadge 89b] and the corresponding models are called *discrete event models* (DEM) [Inan 88].

These systems require control and coordination to ensure the orderly flow of events. As controlled (or potentially controllable) dynamic systems, DES qualify as a proper subject for control theory. Hence a twofold issue presents itself: we need a *powerful* class of DEM, capable of capturing the essential features of discrete, asynchronous and possibly nondeterministic systems, and a *unifying* theory for their control.

The goal of the present research is that of applying a *Petri net model* within the framework of *Supervisory Control*, a control theory for DES introduced recently by Ramadge and Wonham [Ramadge 83, Ramadge 87]. We study how Petri net models may be used to solve supervisory control problems, i.e, the design of a supervisor capable of restricting the behavior of a system within a legal behavior.

1.2 Background and Motivation

In this section we rather informally discuss the motivation of this research. We want to show why Petri nets are a good discrete event model and a powerful tool for Supervisory Control.

1.2.1 Discrete Event Systems and Models

A DES is a dynamic system with a discrete state space and piecewise constant state trajectories (see Figure 1.1); the time instant at which state transitions occur, as well as the actual transitions, will in general be unpredictable.

The state transitions of a DES are called *events* and may be labeled with the elements of some alphabet Σ. These labels usually indicate the physical phenomenon that caused the change in state. For example, in a manufacturing environment typical event labels are *"machine 1 starts working on part A"*, *"machine 1 breaks down"*, etc.

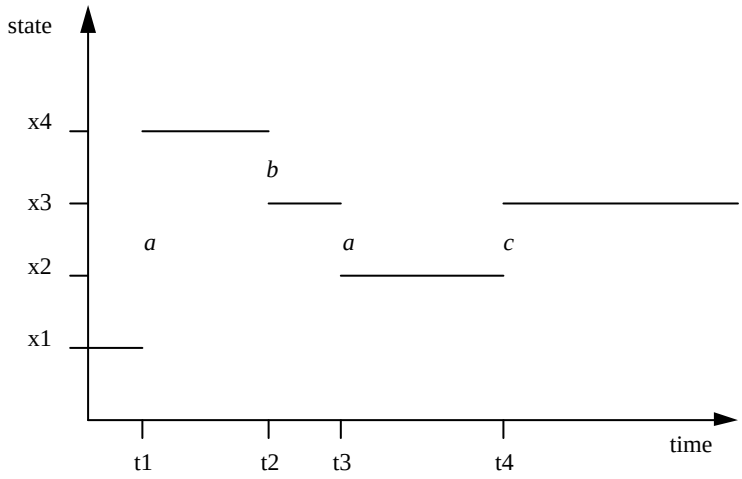


Figure 1.1: State trajectories for a Discrete Event System.

Model Classification

The many areas in which DES arise and the different aspects of behavior relevant in each area have led to the development of a variety of DEM. We have the following classification [Ramadge 86, Ramadge 89b].

1. *Logical DEM*, in which a common simplifying assumption is to ignore the times of occurrence of the events and consider only the order in which they occur. This simplification is justified when the model is to be used to study properties of the event dynamics that are independent of specific timing assumptions.

2. *Timed* or *performance DEM*, which are intended for the study of properties explicitly dependent on interevent timing. These models can be further classified as:

   a) *nonstochastic*: if the timing is known a priori;

   b) *stochastic*: if the timing is not known a priori due to random delays or random occurrences of events.

Logical Models

The behavior captured in a logical model is the sequences of discrete events or *traces* that a system generates [Inan 88]. Let Σ be the set of discrete events labels and let Σ^* be the set of all finite sequences of events in Σ (including the empty trace λ). Thus a possible evolution of the system may be represented, as in Figure 1.1, by

$$w = abac\ldots,$$

where $w \in \Sigma^*$ and $a, b, c \in \Sigma$.

There are two main assumptions here.

1. We are interested in the *order* in which the events occur but not in the *real time* at which they occur;

2. An event is *atomic*, i.e., it is considered to occur in a single step, even if in the real system it may be implemented as a sequence of instructions.

In a logical model, the behavior of a given system is thus given by a subset language $L \subset \Sigma^*$, consisting of all the traces (or *strings*) that the system can generate. In most systems of interest, L will be an infinite set so that it cannot be given simply by listing all the traces. We will study systems that allow a logical model.

1.2.2 Evaluation of Discrete Event Models

A great number of logical DEM have been proposed. In order to evaluate and compare their suitability for control applications, different properties have to be considered [Inan 88].

Descriptive power

The *descriptive power* of a DEM has different facets. Inan and Varaya [Inan 88] have used the terms *language complexity* and *algebraic complexity* to denote two aspects of the descriptive power. We will call these two aspects *language power* and *algebraic power*. Additionally, we will consider a third aspect that we call *representational power*.

A discrete event model must give a means to specify the set of admissible event trajectories. This may be done using different formalisms.

- A *state transition structure* is used by automata and net models such as finite state machines, Petri nets, Büchi automata [Ramadge 89a].
- A *set of equations* is used by boolean models [Aveyard 74], communicating sequential processes [Hoare 85], finitely recursive processes [Inan 88].
- *Logical calculus* is used in temporal logic models [Pnueli 79].

Two different models may represent the same behavior although they use different formalisms. It is often the case, however, that some behavior can be easily modeled with one particular formalism and not so easily modeled with a different one. We call this qualitative aspect **representational power**. In the following section, we will compare different models with Petri nets with the aim of showing that the representational power of Petri nets is fairly large. In fact, Petri nets are not subject to the representational shortcoming typical of other models.

We have seen that the logical behavior of a system is represented in the model by a language $L \subset \Sigma^*$. In general, different formalisms generate different classes of languages, i.e., each model has its own **language power**. As an example, the class of languages generated by finitely recursive processes is a superset of Petri net languages that in turn are a superset of regular languages (the class of languages generated by finite state machines). It is desirable that the language power of a model be as large as possible, so that the model may be used to represent a large variety of systems. However, formal language theory shows that the analysis complexity grows with the language power of a model. The extreme case is that of Turing machines that have the largest language complexity but whose languages have many undecidable properties.

Complex systems can be regarded as being built out of interacting subsystems. A modeling formalism will be useful if it contains operators that combine one or more models in ways that reflect the ways in which systems are connected. In our view of a DEM as a language generator, these operators, union, intersection, concurrent composition, etc., are defined on languages. The class of operators related to each model define its **algebraic power**.

Performance Evaluation

Once we are sure that a model has captured the essential features of a system, we want to use that model for determining whether the system has the desired properties or whether it is free of abnormal behavior.

This can be done in two steps. Firstly, "translate" the desirable properties of the system into properties of the model. Secondly, "derive effective algorithmic, analytical, or simulation methods to verify that the model possesses the desired properties" [Inan 88].

In this last step, the issue of *computational complexity* is a key concern. For example, the number of states in a transition structure for specifying the admissible event trajectories may depend exponentially on some system parameter. In such cases simple algorithms for verification or synthesis, e.g., an exhaustive search over the state space, rapidly become computationally intractable. Thus one tries to mitigate the complexity by the use of aggregation or modularity, or by exploiting hierarchical or other special structures.

Control Implementation

The modeling and analysis of systems is only the first step in the study of DES. The final goal is to modify (by control action) the set of admissible trajectories so that each event trajectory has the desired properties. A model should then find application in the framework of a control theory. It should also provide a guide to constructing a controller. At best, this is done by an automatic compilation of the model into control code; at worst, "ad hoc" solutions must be studied.

1.2.3 Petri Net Models

In this work we study a particular Petri net (PN) model: Place/Transition (P/T) nets. Petri nets satisfy all the requirements that a good DEM should have [Giua 92a].

1. **Descriptive power**
   Petri nets have great descriptive power. They have been designed specifically to model systems with interacting components and as such are able to capture many characteristics of an event driven system, namely concurrency, asynchronous operations, deadlocks, conflicts, etc. Furthermore, the PN formalism permits description of logical models (P/T nets, Colored PN), timed models (Timed PN) and performance models (Stochastic PN, Generalized Stochastic PN).

   When logical models are considered, classes of languages generated by Petri nets, called Petri nets languages, may be defined. These classes are supersets of regular languages and with the introduction of a great number of operators define an algebra. However, PN languages do not have all the nice closure properties of, say, regular languages.

   The basic P/T net cannot model a condition of the form: "Fire transition t if place p is empty". Hence a P/T Petri net cannot simulate a register machine which in turn is equivalent to a Turing machine [Ichikawa 88a]. It is possible to extend the formalism with the introduction of *inhibitory arcs*, i.e., arcs from places to transitions that prevent the firing of a transition if the input place is marked. However we note that the descriptive power of P/T nets is large enough for most common applications, and often even more restricted models are considered, such as conservative PN.

   Conservative PN are essentially equivalent to finite state machines, since the number of reachable markings (i.e., the number of 'states' of the model) is finite. However, they still make use of the full representational power of Petri nets. In fact, since the states of a PN are represented by the possible markings and not by the places, they allow a compact description, i.e., the structure of the net may be maintained small in size even if the number of the markings grows.

2. **Performance Evaluation**
   The desired properties of a system map fairly well into properties of the corresponding Petri

net model. Many algorithms, with a well developed mathematical and practical foundation, have been developed to study these properties.

The analysis techniques for Petri nets may be divided into the following groups (see [Silva 85]).

- *Analysis by enumeration.* It requires the construction of the *reachability tree* representing the set of reachable markings and transition firings. If this set is not finite, a finite *coverability tree* may be constructed.
- *Analysis by transformation.* A net N_1 is transformed, according to particular rules, into a net N_2 while maintaining the properties of interest. The analysis of the net N_2 is assumed to be simpler than the analysis of the net N_1. Examples of this analysis technique are *reduction methods*, that permit the simplification of the structure of a net.
- *Structural analysis*. It permits the demonstration of several properties almost independently of the initial marking. Structural analysis may be based on the study of the state equation of the net or on the study of the net graph.
- *Simulation analysis*. It is useful for timed nets and to study the behavior of nets that interact with an external environment.

Petri nets also represent a hierarchical modeling tool and allow reduction of the computational complexity of analysis by exploiting *modular synthesis*. With modular synthesis, complex systems may be constructed by aggregation of simpler modules while preserving the properties of interest.

3. **Control Implementation**
   Petri nets languages offer a simple means of applying control-theoretical ideas and in this thesis we will show that they are consistent with Supervisory Control theory.

   Petri net based controllers may be implemented in both hardware and software. Programmable Logic Controllers [Silva 89b, Silva 83] and Petri-net like languages [Murata 86] have been used in different applications.

   In [Crockett 87, Kasturia 88] a Petri net interpreter is implemented using a mixture of application dependent and independent code. However, it is necessary to point out that there exists no general technique for compiling a Petri net description into a control system.

It may be worth comparing the *representational power* of PN with that of other models for concurrent systems [Best 91]. It is possible to partition the current approaches into: $a)$ models in which the basic notion is that of *state*, such as state machines; $b)$ models in which the basic notion is that of *action*, such as interleaving models. Petri nets show more flexibility in this respect, since states and actions are treated on equal footing and both step and interleaving semantics can be defined on Petri nets [Best 91].

In the next example, we will see that Petri nets give a very compact description of systems which, due to concurrent behavior, have a large state space.

Example 1.1. Suppose we have a cyclic process where five jobs may be in four different stages. In Figure 1.2 we have a Petri net where each token represents a job and each place represents a different stage. Although the number of reachable markings is 56, the PN model is very simple compared to a state machine model where we need to explicitly represent all states.

Models based on actions do not suffer from this state space explosion, since they do not require the explicit enumeration of all the states. As an example, an interleaving model of the system considered in this example may be described as follows. Let L_1 be the behavior of the cyclic process with a single job, i.e., a single token. Then possible evolutions of the system are:

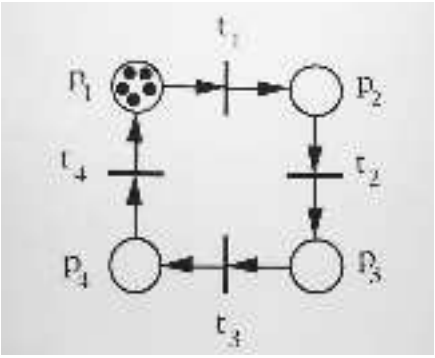


Figure 1.2: Petri net in Example 1.1.

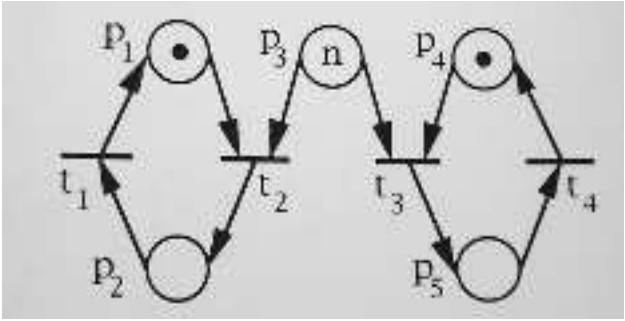


Figure 1.3: Petri net in Example 1.2.

$\lambda, t_1, t_1t_2, t_1t_2t_3, \ldots$, i.e., $L_1 = \overline{(t_1\ t_2\ t_3\ t_4)^*}$; here the bar stands for the prefix operator and $*$ is the Kleene star operator. Then the behavior of the process when five jobs are been processed concurrently is given by

$$L_5 = L_1 \parallel L_1 \parallel L_1 \parallel L_1 \parallel L_1,$$

where $\parallel$ is the concurrent composition operator.

The shortcoming of models based on actions lies in the cumbersome way in which specifications involving counters and semaphores are modeled.

Example 1.2. Consider the Petri net in Figure 1.3. Here we have two sequences (t_2t_1) and (t_3t_4) that can be run up to a total of n times, n being the number of tokens in p_3. Using an interleaving model to represent this process is awkward: we have to explicitly specify all concurrent firing sequences. In fact, for $n = 1$ the behavior is

$$L_1 = \overline{t_2t_1 + t_3t_4}.$$

For $n = 2$ we have

$$L_2 = \overline{(t_2t_1)^2 + (t_2t_1) \parallel (t_3t_4) + (t_3t_4)^2}.$$

For a generic n

$$L_n = \overline{(t_2t_1)^n + (t_2t_1)^{n-1} \parallel (t_3t_4) + \ldots (t_2t_1) \parallel (t_3t_4)^{n-1} + (t_3t_4)^n}.$$

1.2.4 Supervisory Control

Ramadge and Wonham provide a unifying framework for the control of discrete event systems [Ramadge 89b], and so far their *Supervisory Control Theory* is the most general and comprehensive theory presented for logical DES. An introduction to the theory is presented in Appendix B. Here we simply discuss those aspects of Supervisory Control that have motivated this research.

We may distinguish three different areas of interest in the supervisory control approach.

1. The **formal language level** is concerned with theoretical issues. Qualitative properties of DES, such as *stability* [Brave 90], *controllability* [Ramadge 87], *observability* [Cieslak 91, Özveren 90], *nonblockingness* [Ramadge 87], etc., are defined from a very general perspective as properties of languages. This "abstract" perspective has made possible the creation of a model-independent theory.

   At this level, new language operators, that will play a central role in solving the control problem, are also defined. Examples of these operators are the *supremal controllable sublanguage* (SCS) [Wonham 87], *infimal controllable superlanguage* (ICS) [Lafortune 90a], etc. The closure properties of different classes of languages under these operators are also discussed. The class of regular languages is closed under the SCS and ICS operators, as shown by Ramadge and Wonham [Wonham 87], and Lafortune and Chen [Lafortune 90a]. No results have been published on Petri net languages.

   Finally, language theory provides a rigorous formalism for proving important theorems on the existence of supervisors. The existence of finite state supervisors have been discussed by Ramadge and Wonhan [Wonham 87] and Ushio [Ushio 90]. Some results on Petri net supervisors with inhibitory arcs have been presented by Sreenivas and Krogh [Sreenivas 92].

2. The **system level** deals with the concept of DES, seen as a language generator over an alphabet of events Σ. One the most important features is the partition of Σ into two disjoints sets: the set of *controllable* events Σ_c, and the set of *uncontrollable* events Σ_u. The uncontrollable events are those which cannot be disabled by a control action.

   The basic control problem is the following. We are given a DES $\mathbf{G}$ generating the language $L(\mathbf{G})$ and a specification, i.e., a legal language L. We want to restrict the behavior of the system, by disabling only controllable transitions, within the limits of the legal language.

   The control structure that restricts the behavior of the system is called a *supervisor*. The supervisor receives as input the sequence of events generated by $\mathbf{G}$ and determines a control input that specifies which events are to be disabled in $\mathbf{G}$. If we consider the system **G** as the *plant* (or the object to be controlled) and the supervisor as the controlling agent, it is often possible to realize a supervisor simply as another DES **S**. **S** and **G** are supposed to run in parallel and the closed loop structure is again a DES, denoted by **S/G**, the *action* of **S** on **G**. It has been noted [Cieslak 91] that this approach allows us to evaluate and compare the effect of different control policies on the behavior of the uncontrolled system, since open loop plant and controller are treated separately; previous approaches require a separate model for each control policy and only permit models of closed loop systems.

   Design algorithms have been devised to compute the control structure of a supervisor for a given control problem, given the knowledge of the system and of the specification. These algorithms are based on language operators, thus they are model independent.

3. At the **model level** a particular model is chosen to describe the physical device to be controlled. In classical control theory there exist different models for, say, linear systems, such as transfer function, state variables, matrix-fraction description, polynomial matrix description, etc., all capable of describing systems whose behavior is defined by a differential linear

equation. In the case of a discrete event system, the behavior is defined as a formal language; hence the models used are language generators.

In Section 1.2.2, we have already discussed the good properties that a discrete event model should feature.

1.3 Objectives of the Thesis

Supervisory Control is so far the most complete theory for the control of DES. It has a sound mathematical foundation, based on formal languages; it provides exact algorithms to design a supervisor for large classes of control problems; it captures the physical constraints the controller must satisfy, since it assume that only a subset of the events is controllable; it is sufficiently general to be applied to different models.

Another issue of paramount importance is the complexity involved in solving a control problem. Here the choice of one model rather than another one is a key factor that may drastically affect the class of problems that can be solved.

Firstly, since the language power of each model is limited, there may exist systems whose behavior may not be expressed with a given model. It is often assumed that physical systems have a finite state space. However there are cases where it may be necessary to use a model such as Petri nets, with a possibly infinite number of states.

Secondly, the representational power of a model may not be specifically tailored to represent a given system's behavior. The exponential growth of the state space of a model with the number of interconnected modules is a well known blight in the study of concurrent systems which makes the use of simple models, such as finite state machines, impracticable. We have seen, in the previous discussion, that Petri nets reduce this complexity by means of its representation power. Thus Petri nets may potentially be viewed as an effective means to realize supervisory controllers, allowed by language theory but in practice until now unattainable because of the state space explosion problem.

Thirdly, each model has its own tool-set of design and analysis techniques, which may be used more or less efficiently for the solution of a control problem. Petri nets provide a compact representation of systems, but a brute force analysis of the model may often require the exhaustive search of the reachability tree, thus making its computational complexity exponential as well. There are ways to mitigate the complexity involved in the analysis of the model by using techniques that are primarily based on the structure of a net rather than on its behavior. Examples of these techniques are *incidence matrix analysis* and *modular synthesis*. Although these techniques have been extensively investigated in the Petri net literature, so far they have been used in the context of Supervisory Control only by Holloway and Krogh [Holloway 90, Krogh 91, Holloway 92a].

We are finally ready to state the goal of this research: *"Investigate the use of Petri nets models within the framework of Supervisory Control."*

The fulfillment of this goal requires several steps.

1. At the formal language level, we need to study Petri net languages in the context of Supervisory Control. This will allow us to characterize the class of control problems that may be solved by Petri net models, i.e., the class of problems for which a Petri net supervisor exists. In particular, we need to consider the closure properties of Petri net languages under the operators used in Supervisory Control theory.

2. At the discrete event system level, we need to study the counterpart on a Petri net structure of the linguistic operators required in the design of a supervisor. Wonham has proposed an algorithm which uses three different operators: shuffle, selfloop and intersection. We propose an alternative design that requires only one operator: concurrent composition. The

counterpart of this operator on the net structure can be easily defined and preserves the modularity of the overall system.

It is often the case that once the coarse structure of a supervisor has been obtained, by concurrent composition of the system and specification modules, we need to refine the net, to avoid reaching a set of undesirable markings. We need to derive refinement procedures which preserve the modular structure of the net in this step as well.

3. At the model level, we need to make full use of the Petri net analysis techniques to efficiently validate a Petri net supervisor, i.e., to prove that the supervisor has the desired properties.

   We will explore two different techniques. The first one is based on incidence matrix analysis, in which the reachability set of a marked net is represented by the integer solutions of a set of linear inequalities. Classic incidence matrix analysis is based on the solution of the state equation of a net. This approach, however, has been shown useless [Colom 89] in determining some properties of net systems such as the existence of home markings, i.e., markings that may be reached by any marking in the reachability set of a Petri net. Determining the existence of home markings is a problem essentially equivalent to that of determining if a net is nonblocking, which is an important property of discrete event systems. Thus, we need to modify this approach in order to be able to use this analysis technique in the context of Supervisory Control.

   A second area of our research is the use of modular synthesis techniques for nets with uncontrollable transitions, i.e., transitions that may not be prevented from firing by a control agent. We will explore a class of specification, called mutual exclusion constraints, that have also been considered in [Krogh 91]. Our aim is that of efficiently deriving a Petri net structure for a supervisor for this class of problems.

1.4 Organization of the Thesis

This chapter has introduced the topic of the thesis and stated the objectives of research.

Chapter 2 discusses in depth the literature relevant to the present research.

Chapter 3 introduces the basic notation on Petri nets, formal languages and Supervisory Control.

Chapter 4 discusses the use of Petri nets in the framework of Supervisory Control. Firstly, it is shown that the trimming of an unbounded Petri net is not always possible; this leads to the definition of a class of Petri net languages, that may be generated by nonblocking generators. Secondly, necessary and sufficient conditions for the existence of a Petri net supervisor under the hypothesis that the system's behavior and the legal behavior are both Petri net languages are derived. Finally, by means of an example, it is shown that Petri net languages are not closed under the supremal controllable sublanguage operator.

Chapter 5 shows how Petri nets may be used to design a supervisor. The design requires two steps. In the first step, a coarse structure for a supervisor is synthesized by means of concurrent composition of different modules. In the second step, the structure is refined to avoid reaching forbidden markings. We show that for conservative Petri nets there exists a standard refinement procedure that only requires introducing new arcs and possibly duplicating some transitions, i.e., no new places are introduced. In both steps, the use of Petri nets allows us to keep the structure of the model 'small'.

Chapter 6 discusses how incidence matrix analysis for Petri net models may be used to validate supervisors. A new class of P/T nets, called Elementary Composed State Machine nets, is defined. The reachability problem for this class can be solved by incidence matrix analysis. In fact it is possible to derive a set of linear inequalities that exactly defines the set of reachable markings.

Thus, important properties of discrete event systems, such as the absence of blocking states or controllability, may be analyzed by Integer Programming techniques.

Chapter 7 defines a class of specifications, called generalized mutual exclusion constraints, for discrete event systems modeled using P/T nets. These specifications may be easily enforced on a net system where all transitions are controllable, by a set of places called monitors. However, when some of the transitions of the net are uncontrollable this technique is not always applicable. For some classes of nets, we prove that mutual exclusion constraints may always be enforced by monitors, even in the presence of uncontrollable transitions. For one of these classes, marked graphs with control safe places, we compare a monitor-based solution of mutual exclusion problems with several supervisory based solutions.

Chapter 8 concludes the thesis with a discussion of the original contributions and listing possible further directions of research.

Appendix A reviews some basic concepts of formal languages and Petri net languages.

Appendix B is a short introduction to Supervisory Control and supervisory design techniques.

Appendix C is a glossary of the notation used in the thesis.

Chapter 2

LITERATURE REVIEW

The literature of interest to this research covers different topics. Firstly, we review the work done in Supervisory Control. This provides the theoretical basis of our research. Secondly, it is interesting to compare PN with other logical models that have been used in Supervisory Control. Thirdly, since our approach is based on formal languages, we review the work on Petri net languages. Finally, we consider the efficient validation of PN models, using techniques based on incidence matrix analysis, reduction rules, and synthesis operators.

2.1 Supervisory Control Theory

The supervisory theory by Ramadge and Wonham is so far the most comprehensive theory for the control of discrete event systems. It is based on the concept of a *supervisor* [Ramadge 83, Ramadge 87], i.e., an agent that is capable of disabling the controllable transitions of a DES in response to the traces of events generated. The Supervisory Control Problem (SCP) consists in designing a supervisor which restricts the traces generated by the system within a legal behavior. If the legal behavior is a *controllable language* [Wonham 87] a supervisor exists. If the legal behavior is not controllable, we have to further restrict the system's behavior to the *supremal controllable sublanguage* for which a supervisor exists. The authors prove in these seminal papers that the supremal controllable sublanguage always exists (but may be the empty language), and in the regular case may be determined with a finite procedure.

In [Lafortune 90a] a new control problem is studied: the *Supervisory Control Problem with Blocking* (SCPB). Here it is assumed that in some cases the solution to the SCP (supremal controllable sublanguage) may be too conservative. A dual concept is defined — the *infimal controllable superlanguage* — and is used to determine a supervisor that may also permit blocking in order to achieve a larger behavior. In [Lafortune 90a] Lafortune and Chen introduce two performance measures (in terms of *satisficing* and *blocking*), and techniques to improve each of these two conflicting measures. An extension of this work is [Lafortune 91] where the *Supervisory Control Problem with Tolerance* (SCPT) is defined. Given a desired and tolerated behavior, the problem is that of designing a controller such that the controlled system never goes beyond the tolerated behavior and achieves as much as possible of the desired behavior. Under very general hypotheses on desired and tolerated behavior, Lafortune and Lin show that a solution to SCPT exists and is unique, but may be blocking. A non blocking solution exists but is not necessarily unique.

In [Cieslak 91] Cieslak *et al.*, discuss and solve the *Supervisory Control and Observation Problem* (SCOP) and the *Decentralized Supervisory Control Problem* (DSCP). In SCOP the assumption is that a mask is present between the controlled system and supervisor, so that the supervisor cannot observe all the transitions, or cannot distinguish between some of them. In DSCP it is assumed that the control action is enforced by local supervisors that control only subsystems. In

[Lin 88, Lin 90] Lin and Wonham discuss the Decentralized SCOP (DSCOP) where both partial observations and decentralized control are incorporated into the control structure. However, the only mask operator considered in this paper is the language projection operator.

In [Brave 90] Brave and Heymann define *stabilization* as the ability of a discrete event process to reach a set of target states from an arbitrary initial state and then remain there indefinitely. A slightly different problem that the authors examine is recovery under control failure. In both cases they present design algorithms for controllers that improve the stabilization of processes.

In [Ushio 90] Ushio discusses the conditions under which a *finite state supervisor* (FSS) may be constructed to solve a SCP. From [Ramadge 87] it was known that a FSS exists when both system's behavior and specification language are regular. Here the author derives necessary and sufficient conditions for the general case.

The case of the infinite state supervisor is discussed by Sreenivas and Krogh [Sreenivas 92]. They use Petri nets with inhibitory arcs (PNIA), which are known to have a modeling power equivalent to Turing machines, to describe infinite state systems. Thus, they prove that a PNIA supervisor exists if the system's and specification behaviors are Turing computable languages. However, important properties, such as determining if the behavior of a PNIA is controllable, are undecidable.

In [Ramadge 86, Wonham 88b] a *modular* approach to the design of supervisors is considered. The specification language is composed of different specifications, each enforced by a single supervisor. A global control law can be enforced by the conjunction of all the supervisors. Unfortunately in the general case the resulting system may be blocking. When the languages controlled by each supervisor are non-conflicting, however, the non-blocking condition is assured as well.

In [Ramadge 89b, Tadmor 89, Tsitsiklis 87] different problems of computation and the related issue of computational complexity are considered. A review of the theory is presented in [Ramadge 88, Ramadge 89b, Wonham 88a].

2.2 Logical Models for Discrete Event Systems

Numerous approaches to the modeling of discrete event systems have appeared in the literature.

2.2.1 Transition Models

Aveyard [Aveyard 74] presents a Boolean matrix equation model for a class of DES in which the state change associated with each event occurrence is deterministic, and in which all units or entities are permanent. The model, one of the first to be presented, permits checking of simple properties such as whether the system is deterministic and if it will hang or cycle.

Ramadge [Ramadge 89a] introduces Büchi automata in the modeling of DES. The model is used to extend the concept of controllable language to non-finite strings and to derive conditions for the existence of a supervisor to implement a prescribed closed-loop behavior. Several interesting control synthesis problems have a computationally tractable solution when Büchi automata models are used.

2.2.2 Temporal Logic

Hailpern and Owicki [Hailpern 83] use *temporal logic* (a formalism introduced by Pnueli [Pnueli 79] for formalizing the semantics of concurrent programs) to model computer communication protocols. The model is used for the modular verification of safety and liveness for parallel processes.

The temporal logic formalism is applied by Ostroff and Wonham [Ostroff 90b, Ostroff 90a] to the control of real-time discrete event systems. Temporal logic is a useful tool in the formal-

ization of the control problem. However the validation of the model is based on theorem proving techniques and is difficult to perform.

2.2.3 Communicating Processes

Milne and Milner [Milne 79] define concurrent processes by means of a net algebra and define a minimal set of operations for composing processes. The final result of this research is presented in [Milner 80], where the Calculus of Communicating Systems (CCS) is introduced. CCS is one of the standard models for concurrent systems on which recent and on-going research is focused. In [De Cindio 83] De Cindio *et al.* compare CCS with Petri nets. They prove that CCS defines a class of concurrent systems composed of interacting sequential automata. In fact the CCS models are isomorphic to a subclass of Petri nets, Superposed Automata Nets defined in [De Cindio 82].

Inan and Varaiya [Inan 88] present a class of discrete event models called *finitely recursive processes* whose formal structure is based on Hoare's communicating sequential processes [Hoare 85]. The descriptive power of the model and its use in performance evaluation is also investigated. The authors prove that their formalism can easily model any Petri net. However the proof is limited to ordinary Petri nets: when the multiplicity of the arcs is greater than one, the complexity of the finitely recursive process grows and nondeterminism needs to be introduced.

2.2.4 Controlled Petri Nets

Several authors have used a particular Petri net model, called *controlled Petri nets*, within the framework of Supervisory Control. Controlled PN have been introduced by Ichikawa and Hiraishi [Ichikawa 88a] to study the input-output behavior of discrete event models. This model is a standard PN with the addition of *external input places*, whose marking is given by an external function. It is furthermore assumed that only the marking of a set of so called *external output places* may be observed. In the paper, Ichikawa and Hiraishi restrict the firing policy on the net to be *decision-free*, i.e., all enabled transitions should fire simultaneously. Under these hypotheses the modeling power of the net is increased by the capability of detecting whether a place is empty; they can prove that this model is equivalent to Turing machines.

Krogh [Krogh 87] applies controlled PN to Supervisory Control. He assumes that each transition is associated to at most one external input place and studies how the behavior of the net can be restricted within a certain behavior by a particular marking function for the external input places, i.e., a particular control law. This control law is implementing a state feedback rather than a trace feedback, i.e., it is a function of the actual marking of the system rather than a function of the string of events generated by the system. The author defines the control law that generates the supremal controllable sublanguage as the *maximally permissible feedback*. He also notices that the maximally permissible feedback may not be unique because of the particular firing policy that permits the simultaneous firing of enabled transitions.

Ushio and Matsumoto [Ushio 88] derive necessary and sufficient conditions for the uniqueness of the maximally permissible feedback on controlled PN. In a subsequent paper [Ushio 89], Ushio also derives necessary and sufficient conditions for the existence of a control law that satisfies a given specification.

Holloway and Krogh [Holloway 90, Krogh 91] show that, for restricted classes of nets and restricted specifications, the control problem may be solved with great computational efficiency by analysis of the net structure. The class of controlled Petri nets considered is cyclic marked graphs and the admissible specifications consist of forbidden markings that simultaneously assign tokens to one or more set of places. Although these restrictions are certainly heavy, the authors show an interesting manufacturing example that requires the coordination of automated guidance vehicles. Liveness properties under control are discussed in [Holloway 92a].

The main characteristic of the controlled PN approach to supervisory control is the fact that no transition structure for a controller is given. The control law is a function of the actual marking of the net, but need to be computed at each step, whereas if a transition structure is given for a supervisor, a closed-loop model can be constructed and analyzed to ensure that it has the desired properties.

2.3 Petri Net Languages

Baker [Baker 72] is one of the first to introduce the basic idea of associating a language with a Petri net and using this language to describe the behavior of the net. In his Master's thesis [Baker 73], Petri nets are considered in the context of formal language theory, and their prefix language is studied.

In Hack [Hack 75a, Hack 75b], two basic language classes are distinguished: the prefix language and the marked language. Different languages which result from different types of transition labeling are also considered. For each of these classes, simple closure properties, characterizations and decidability problems (e.g., for membership, emptiness, and finiteness) are obtained.

A survey and tutorial on Petri net languages is presented in [Peterson 81] Chapter 6. Here the definition and classification of these languages is given; the closure properties of Petri net languages under several form of composition (union, intersection, concatenation, concurrency, and substitution) and under some operations (reversal, complement, and indefinite concatenation) are examined; the relationship between Petri net languages and other classes of formal languages are investigated.

Another comprehensive tutorial on Petri net languages is a paper by Jantzen [Jantzen 87]. Closure properties and relationship among the different classes of Petri net languages are reported with additional results, not presented by Peterson.

Parigot and Peltz [Parigot 86] have characterized PN languages in terms of logical formalism. The language power of Petri nets is shown to be greater than that of finite automata as a result of the additional ability of Petri nets to test if a string of parentheses is well formed. The logical formalism may be used to prove that a given language is a Petri net language and to construct a Petri net having a given *finite* behavior. Peltz [Peltz 86] has also extended the study to the *infinite sequential* behavior of Petri nets.

2.4 Incidence Matrix Analysis of Petri Nets

Incidence matrix and related analysis techniques are used by several authors to validate properties of P/T nets.

Colom [Colom 89] develops a methodology for the verification of assertions on P/T nets in terms of markings and firing count vectors. This approach is extremely general, i.e., can be applied to any P/T net but, unfortunately, can only guarantee necessary or sufficient conditions. There exists assertions, such as determining if a marking is a home state, for which neither a necessary nor a sufficient condition can be given. Within this framework, Silva and Colom [Silva 89a] give algorithms to efficiently compute the structural synchronic invariants of P/T nets.

Berthomieu [Berthomieu 87] uses the set of linear equations given by the P-semiflows to represent the space of reachable markings. In this case, only properties that depend on the markings can be proved. However, the author also shows that it is possible to prove some properties that depend on the firing count vector by adding to the net new places whose marking indicates the number of times a given transition has fired.

Johnen [Johnen 87] uses an hybrid approach, based partly on incidence matrix analysis and partly on the analysis of the state space, to verify that a given marking is a home state.

Ichikawa and Hiraishi study under which conditions a firing count vector $\vec{\sigma}$, satisfying the state equation of a Petri net, yields a firing sequence. In [Ichikawa 88b], as reported in [Murata 89], they prove that in acyclic nets, $\vec{\sigma}$ always yields a firing sequence. In [Ichikawa 88a, Murata 89] other classes of nets, such as trap-circuit nets, trap-containing-circuit nets, etc., are considered. For these classes, there exist necessary and sufficient conditions, based on the analysis of the firing subnet, to determine if $\vec{\sigma}$ gives a firing sequence. Murata also discusses reachability in marked graphs in [Murata 77].

Avrunin *et al.* [Avrunin 91] use a formalism similar to Petri net incidence matrix analysis for verifying properties of concurrent systems described as finite state automata.

2.5 Synthesis and Reduction of Petri Nets Models

In the *synthesis* of Petri net models there are two main approaches: *bottom-up* and *top-down*. Following the first approach a complete net is constructed combining subnets under given operators; in the second approach, transition and places in a net can be replaced by a more detailed subnet. In both cases the synthesis rules are such as to preserve some properties of the original modules.

A complementary approach is that of defining *reduction rules* that preserve the properties of interest, while simplifying the structure of the net to make analysis easier.

Berthelot [Berthelot 86, Berthelot 87] has presented several reduction rules. In [Berthelot 86] he also discusses the composition of nets by common transitions as a complementary technique to the reduction methods. The classes of reduction rules include: *place transformations*, that reduce the net structure by eliminating redundant places but do not modify the state space; *transition transformations*, which by fusing transitions reduce both structure and state space of the net. The properties preserved by these transformations are: covering with P-semiflows (i.e., conservativeness), proper termination, home states and liveness. The author also proves that there are classes of nets to which these transformations can be repeatedly applied to reduce the net to a single transition. Two such classes are: live and bounded marked graphs (a structural class); live, bounded, and persistent nets (a behavioral class).

Best and Fernández [Best 86] define the notion of *S-net*, a net in which each transition has at most one input place and one output place. An *S-decomposition* is a partition of a net into S-net components.

Hack [Hack 72, Hack 74] studies the composition of state machines, defining a *state machine decomposable net* as a net constructed by composition of strongly connected state machines along common transitions.

The class of nets obtained by composition of state machine modules is named *Superposed Automata Nets* by De Cindio, *et al.* [De Cindio 82].

Narahari and Viswanadham [Narahari 85] define a *union operator* [1] for pure Petri nets, i.e., PN with no selflooping transitions. Two theorems are presented, stating that the P-semiflows and/or the T-semiflows of the union of two Petri nets can be immediately expressed — in specific cases — in terms of the invariants of the two subnets. The invariants are used to analyze important qualitative aspects of the system such as absence/existence of deadlocks. Finally, the authors illustrate how this technique may be applied to the modeling of flexible manufacturing systems.

Beck and Krogh [Krogh 86] present a methodology for constructing a class of P/T nets for modeling discrete processes. The synthesis of Petri nets is realized by joining subnets along common *simple elementary paths*. This composition maintains the properties of *safeness* and *liveness*; the P-semiflows of the resulting net can also be easily found in term of the P-semiflows

[1] This operator is different from the union operator as defined in [Peterson 81], which given two Petri nets generating, respectively, languages L_1 and L_2 constructs a Petri net generating the language $L = L_1 \cup L_2$.

of the subnets. The authors introduce a modified Petri net model [Beck 86] more suitable for the modeling of manufacturing systems by means of the synthesis approach they propose.

Datta and Gosh [Datta 84, Datta 86] present a technique for the modular synthesis of PN. The nets being considered are *regular nets*, i.e., nets where forks and joins are symmetrical, and where the subnets between a fork and a join are marked graphs. The modules are connected through a set of input/output places, that act as mailboxes. Finally, it is possible to give rules to connect the modules in a way that preserves liveness and boundedness.

Zhou [Zhou 90] has studied a formal methodology for the synthesis of Petri nets in manufacturing. The approach followed is called *hybrid*, in the sense that it is partially top-down and partially bottom-up. The author has also discussed how liveness properties for systems with shared resources depend on the value of the initial marking of the net.

Koh [Koh 92] has studied the compositions of live and bounded circuits along paths. The composition allows the fusion of overlapping paths, and has been extended to colored Petri nets. The author has studied under which conditions the composed system is live and bounded.

Chapter 3

BASIC NOTATION

3.1 Petri Nets

3.1.1 Place/Transition Nets

A *Place/Transition net* (P/T net) is a structure $N = (P, T, Pre, Post)$ where:

- P is a set of *places* represented by circles, $|P| = m$;
- T is a set of *transitions* represented by bars, $|T| = n$;
- $Pre : P \times T \to I\!N$ is the *pre-incidence function* that specifies the arcs directed from places to transitions;
- $Post : P \times T \to I\!N$ is the *post-incidence function* that specifies the arcs directed from transitions to places.

Here $I\!N = \{0, 1, 2, \ldots\}$. It is assumed that $P \cap T = \emptyset$ and $P \cup T \neq \emptyset$.

A net is *pure* if it has no selfloops, i.e., if $[Pre(p,t) = 0 \vee Post(p,t) = 0]$ for every place p and for every transition t. If a net is pure the incidence functions can be represented by a single matrix, the *incidence matrix* of the net, defined as $C(p,t) = Post(p,t) - Pre(p,t)$.

The *preset* and *postset* of a transition t are respectively

$$^\bullet t = \{p \in P \mid Pre(p,t) > 0\},$$

$$t^\bullet = \{p \in P \mid Post(p,t) > 0\}.$$

The *preset* and *postset* of a place p are respectively

$$^\bullet p = \{t \in T \mid Post(p,t) > 0\},$$

$$p^\bullet = \{t \in T \mid Pre(p,t) > 0\}.$$

A *trap* is a set of places $\mathcal{T} \subseteq P$ such that

$$\bigcup_{p \in \mathcal{T}} p^\bullet \subseteq \bigcup_{p \in \mathcal{T}} {}^\bullet p.$$

A trap is *minimal* if it is not the superset of another trap. A trap is *basic* if it is not the union of other traps.

A *siphon* is a set of places $\mathcal{S} \subseteq P$ such that

$$\bigcup_{p \in \mathcal{S}} {}^\bullet p \subseteq \bigcup_{p \in \mathcal{S}} p^\bullet.$$

A *P-semiflow* (or *nonnegative P-invariant*) is a vector $Y : P \to \mathbb{N}$, such that $Y \geq \vec{0}$ and $Y^T \cdot C = \vec{0}$. The *support* of Y is $Q_Y = \{p \in P \mid Y(p) > 0\}$.

The *reversal* of $N = (P, T, Pre, Post)$ is the net $N^R = (P, T, Post, Pre)$, i.e., a new net where the direction of all arcs of N is reversed.

A *state machine* is a P/T net such that each transition has exactly one input arc and one output arc. A *marked graph* is a P/T net such that each place has exactly one input arc and one output arc.

A P/T net is *connected* if in the underlying graph[1] there exists a path (not necessarily directed) from any vertex to all others; *strongly connected* if there exists a directed path from any vertex to all others; *acyclic* if no directed path forms a cycle.

3.1.2 Marked Nets

A *marking* is a vector $M : P \to \mathbb{N}$ that assigns to each place of a P/T net a non-negative integer number of tokens, represented by black dots. $M(p)$ denotes the number of tokens assigned by marking M to place p. The set of all markings defined on a net $N = (P, T, Pre, Post)$ is $\mathbb{N}^{|P|}$.

A *net system* or *marked net* $\langle N, M_0 \rangle$ is a net N with an initial marking M_0.

A transition $t \in T$ is *enabled* at a marking M if $M \geq Pre(\cdot, t)$. If t is enabled at M, then t may fire yielding a new marking M' with $M' = M + C(\cdot, t)$. We will write $M\ [t\rangle\ M'$ to denote that t may fire at M yielding M'.

A *firing sequence* from M_0 is a (possibly empty) sequence of transitions $\sigma = t_1 \dots t_k$ such that $M_0\ [t_1\rangle\ M_1\ [t_2\rangle\ M_2 \cdots\ [t_k\rangle\ M_k$. We will also write $M_0\ [\sigma\rangle\ M_k$ to denote that we may fire σ at M_0 yielding M_k.

A marking M is *reachable* in $\langle N, M_0 \rangle$ if there exists a firing sequence σ such that $M_0\ [\sigma\rangle\ M$.

Given a marked net $\langle N, M_0 \rangle$, the set of firing sequences (also called *language* of the net) is denoted $L(N, M_0)$ and the set of reachable markings (also called *reachability set* of the net) is denoted $R(N, M_0)$.

Let marking M be reachable from marking M_0 by firing a sequence of transitions σ. Then the following *state equation* is satisfied: $M = M_0 + C \cdot \vec{\sigma}$, where $\vec{\sigma} : T \to \mathbb{N}$ is a vector of non-negative integers, called the *firing count vector*. $\vec{\sigma}(t)$ represents the number of times transition t appears in σ. The set of markings M such that there exists a vector $\vec{\sigma}$ satisfying the previous state equation is called the *potentially reachable set* and is denoted $PR(N, M_0)$. Note that $PR(N, M_0) \supseteq R(N, M_0)$.

The *firing subnet* given by a firing count vector $\vec{\sigma}$ consists of all the transitions $t \ni \vec{\sigma}(t) \geq 0$, and of their input and output places.

Let B be a basis of P-semiflows of the net N. A marking $M \in PR(N, M_0)$ satisfies the following system of equations: $B^T \cdot M = B^T \cdot M_0$. The set of markings satisfying the previous system of equations is denoted $PR^B(N, M_0)$ [Colom 89]. Note that $PR^B(N, M_0) \supseteq PR(N, M_0)$.

A marked net $\langle N, M_0 \rangle$ is:

- *Bounded*, if there exists a nonnegative integer k such that $M(p) \leq k$ for every place p and for every marking $M \in R(N, M_0)$;
- *Safe* (or *1-bounded*), if $M(p) \leq 1$ for every place p and for every marking $M \in R(N, M_0)$;
- *Conservative*, if there exists a vector of positive integers Y such that $Y^T \cdot M = Y^T \cdot M_0$ for every marking $M \in R(N, M_0)$;
- *Live*, if from every marking $M \in R(N, M_0)$ there exists a firing sequence containing all transitions;

[1] The graph underlying a Petri net has as vertices places and transitions, and as directed edges the arcs specified by Pre and $Post$.

- *Reversible*, if the initial marking M_0 is reachable from every reachable marking $M \in R(N, M_0)$.

A place p of a marked net $\langle N, M_0 \rangle$ is *implicit* [Silva 85] if $L(N, M_0) = L(N', M_0')$, where $\langle N', M_0' \rangle$ is the marked net obtained from $\langle N, M_0 \rangle$ by removing place p and all its input/output arcs. A place p of a net N is *structurally implicit* if there exists an initial marking M_0 such that p is implicit in $\langle N, M_0 \rangle$.

3.2 Formal Languages

Let Σ be an alphabet, i.e., a set of symbols. A language L over Σ is a set of strings composed with symbols in Σ, i.e., $L \subseteq \Sigma^*$.

Let $\Sigma_1, \ldots, \Sigma_n$ be alphabets and let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$. Given a string $w \in \Sigma^*$, the *projection* (or *restriction*) of w on Σ_i is the string $w \uparrow_i$ obtained from w by deleting all the symbols not belonging to Σ_i. Formally the projection operator is a mapping from Σ^* to Σ_i^*.

Let $\Sigma_1, \ldots, \Sigma_n$ be alphabets; let $L_1, \ldots, L_n$ be languages defined on them; let $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_n$. The *concurrent composition* (or *synchronization*) of $L_1, \ldots, L_n$, denoted $L = L_1 \parallel \ldots \parallel L_n$, is the language over Σ defined as

$$L = \{w \in \Sigma^* \mid w \uparrow_i = w_i \in L_i, \;\; i = 1, \ldots, n\}.$$

As a particular case, when the alphabets $\Sigma_1, \ldots, \Sigma_n$ are disjoint, i.e., $\Sigma_i \cap \Sigma_j = \emptyset$ $(i \neq j)$, L is called a *shuffle language* and is denoted by $L = L_1 \parallel_d \ldots \parallel_d L_n$.

Let L be a language on Σ, and let Σ_e be a disjoint alphabet. The *selfloop* of L with respect to Σ_e is the language

$$L' = L \parallel_d \Sigma_e^*.$$

Let $\Sigma_1, \ldots, \Sigma_n$ be alphabets; let $L_1, \ldots, L_n$ be languages defined on them; let $\Sigma = \Sigma_1 \cap \ldots \cap \Sigma_n$. The *intersection* of $L_1, \ldots, L_n$, denoted $L = L_1 \cap \ldots \cap L_n$, is the language over Σ defined as

$$L = \{w \in \Sigma^* \mid w \in L_i, \;\; i = 1, \ldots, n\}.$$

3.3 Petri Net Languages

This section presents the notation used for Petri net languages. A more detailed review of Petri net languages is presented in Appendix A.

We will represent a discrete event system (DES) as a *labeled Petri net* (or *Petri net generator*) [Jantzen 87, Peterson 81], i.e., as a 4-tuple $\mathbf{G} = (N, \ell, M_0, F)$ where

- $N = (P, T, Pre, Post)$ is a Petri net structure;
- $\ell : T \rightarrow \Sigma$ is a labeling function that assigns to each transition a label from the alphabet of events Σ and will be extended to a mapping $T^* \rightarrow \Sigma^*$ as shown in Appendix A.1;
- M_0 is an initial marking;
- F is a finite set of final markings.

The two languages associated with $\mathbf{G}$ are the *L-language* (also called *marked behavior* in the context of Supervisory Control) and the *P-language* (also called *closed behavior*) defined as follows [Peterson 81].

Given a DES $\mathbf{G} = (N, \ell, M_0, F)$, the *L-type language* of $\mathbf{G}$ is

$$L_m(\mathbf{G}) = \{\ell(\sigma) \in \Sigma^* \mid \sigma \in T^*, M_0\ [\sigma\rangle\ M \in F\},$$

and the *P-type language* of $\mathbf{G}$ is

$$L(\mathbf{G}) = \{\ell(\sigma) \in \Sigma^* \mid \sigma \in T^*, \exists M' \ni M_0\ [\sigma\rangle\ M'\}.$$

Note that in this definition of labeled net we are assuming that ℓ is a λ-free labeling function, according to the terminology of [Peterson 81], i.e., no transition is labeled with the empty string λ and two (or more) transitions may have the same label. The classes of L-type and P-type languages generated by labeled nets is denoted $\mathcal{L}$ and $\mathcal{P}$ respectively.

A *deterministic* PN generator [Jantzen 87] is such that the string of events generated from the initial marking uniquely determines the sequence of transitions fired. Formally, a DES $\mathbf{G} = (N, \ell, M_0, F)$ is deterministic if $\forall t, t' \in T$, with $t \neq t'$ and $\forall M \in R(N, M_0), \quad M\ [t\rangle \wedge M\ [t'\rangle \Longrightarrow \ell(t) \neq \ell(t')$. We will always assume that the generators considered here are deterministic. The classes of L-type and P-type PN languages generated by deterministic PN generators are denoted, respectively, $\mathcal{L}_d$ and $\mathcal{P}_d$. Note that $\mathcal{P}_d \subset \mathcal{P}$ and $\mathcal{L}_d \subset \mathcal{L}$ (strict inclusion) [Jantzen 87].

3.4 Supervisory Control

This section presents the language notation used for Supervisory Control. A more detailed review of Supervisory Control is presented in Appendix B.

Let L be a language on alphabet Σ. Its *prefix closure* is the set of all prefixes of strings in L: $\overline{L} = \{\sigma \in \Sigma^* \mid \exists \tau \in \Sigma^* \ni \sigma\tau \in L\}$. A language L is said to be *closed* if $L = \overline{L}$.

Two languages L_1 and L_2 are said to be *non-conflicting* [Wonham 88b] if $\overline{L_1 \cap L_2} = \overline{L_1} \cap \overline{L_2}$.

In the following, let $\mathbf{G} = (N, \ell, M_0, F)$ be a DES with alphabet of events Σ.

$\mathbf{G}$ is *nonblocking* if any string that belongs to its closed behavior may be completed to a string that belongs to the marked behavior. A deterministic DES is non-blocking if and only if $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$.

A language $K \subset \Sigma^*$ is said to be $L_m(\mathbf{G})$-*closed* if $K \cap L_m(\mathbf{G}) = \overline{K} \cap L_m(\mathbf{G})$. In the case that $K \subseteq L_m(\mathbf{G})$, this definition is reduced to the usual definition of $L_m(\mathbf{G})$-closed: $K = \overline{K} \cap L_m(\mathbf{G})$ [Ramadge 89b].

The alphabet of events Σ is partitioned into two disjoint subsets: Σ_c, the set of *controllable events*, and Σ_u, the set of *uncontrollable events*. The controllable events may be disabled by a controlling agent in order to restrict the behavior of the system within a legal behavior, while uncontrollable events may never be disabled.

A language $K \subset \Sigma^*$ is said to be *controllable* with respect to $L(\mathbf{G})$ if $\overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}$ [Ramadge 87]. The set of all languages controllable wrt $L(\mathbf{G})$ is denoted $\mathcal{C}(\mathbf{G})$.

If a language $L \subset \Sigma^*$ is not controllable with respect to $L(\mathbf{G})$ we may compute its *supremal controllable sublanguage* [Wonham 87] defined as: $L^{\uparrow} = \sup\{K \subseteq L \mid K \in \mathcal{C}(\mathbf{G})\}$.

Chapter 4

ON THE EXISTENCE OF PETRI NET SUPERVISORS

4.1 Introduction

This chapter discusses some theoretical issues related to the use of Petri nets in Supervisory Control Theory. We are interested in obtaining necessary and sufficient conditions for the existence of a Petri net supervisor under the hypothesis that the system's behavior and the legal behavior are both Petri net languages.

In the case of systems and specifications modeled by finite state machines (i.e., by generators of regular languages), Ramadge and Wonham [Wonham 87] have proved that the supervisor may also be modeled as a finite state machine. The PN models generally considered in Supervisory Control are conservative PN, a subclass of P/T nets whose set of reachable markings is finite. Thus the class of languages generated by these models is equivalent to the class of regular languages, and all the results for regular languages may apply to them as well.

Here we discuss the use of more general models with a possibly infinite set of reachable markings. We consider three problems [Giua 92c].

- The first problem regards the trimming of a blocking system, i.e., the modification of its structure in order that no blocking state may be reached while preserving its marked behavior. We prove that the trimming of a PN is not always possible and we define a new class of PN languages that may be generated by nonblocking generators.
- The second problem regards the existence of a PN supervisor, i.e., a supervisor whose control action is implicit in its net structure. The advantage of such a supervisor, as opposed to a supervisor given as a feedback function, is that a closed loop model of the controlled system may be constructed and analyzed using the same PN analysis techniques that are used to analyze the system. By suitable modification of the results given by Ramadge and Wonham [Ramadge 87, Ramadge 89b] we are able to derive necessary and sufficient conditions for the existence of supervisors as net systems.
- The final problem regards the closure of PN languages under extraction of the supremal controllable sublanguage. In this case, by means of an example, we show that PN languages are not closed under this operator.

4.2 Petri Nets and Blocking

A first issue when using Petri nets as discrete events models for supervisory control regards the trimming of blocking net systems. The problem is the following: given a PN generator $\mathbf{G}$ with

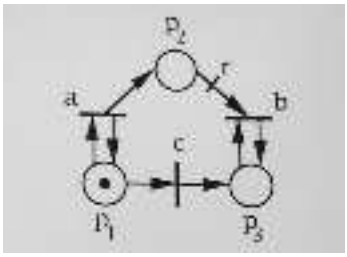


Figure 4.1: Blocking system in Example 4.1.

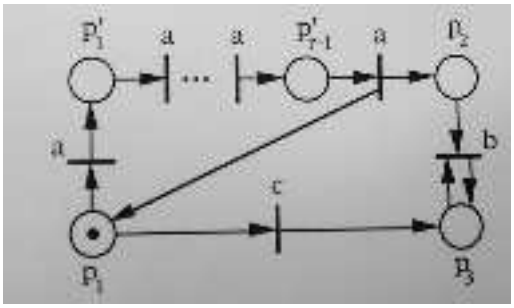


Figure 4.2: Trimmed system in Example 4.1.

marked behavior $L_m(\mathbf{G})$ and closed behavior $L(\mathbf{G}) \supset \overline{L_m(\mathbf{G})}$ we want to modify the structure of the net so that no blocking markings may be reached, i.e., so that $L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$.

On a simple model such as a state machine this may be done, trivially, by removing all states that are reachable but not coreachable (i.e., no final state may be reached from them) and all their input and output transitions.

On Petri net models the trimming may be more complex. In Chapter 5 is discussed the case of conservative nets. If the Petri net is conservative the trimming may be done without major changes of the net structure, in the sense that we have to add new arcs and possibly duplicate transitions without introducing new places.

As the following example shows, unbounded Petri net models may require more extensive changes of the net structure.

Example 4.1. Let $\mathbf{G}$ be the PN generator in Figure 4.1, with $M_0 = (100)^T$ and set of final markings $F = \{(001)^T\}$. The behaviors of this system are: $L_m(\mathbf{G}) = \{a^m c b^n \mid m = rn, n \geq 0\}$ and $L(\mathbf{G}) = \{\overline{a^m c b^n} \mid m \geq rn, n \geq 0\}$. The system is clearly blocking, since the string $w = a^{r+1}c \in L(\mathbf{G})$, for example, cannot be completed to a string in $L_m(\mathbf{G})$ (assuming $r > 1$). To avoid reaching a blocking state we require that p_2 contain a multiple of r tokens when the transition labeled c is allowed to fire. A possible solution is shown in Figure 4.2, where we have introduced $r - 1$ new places and new transitions labeled a.

There are some cases, furthermore, in which the trimming of a net system is not possible. The reason for this is given by the following theorem.

Theorem 4.1. *There exist L-type Petri net languages whose prefix closure is not a P-type Petri net language, i.e.,* $\exists L \in \mathcal{L} \ni \overline{L} \notin \mathcal{P}$.

The proof of this theorem will be given by means of the next example and the following proposition.

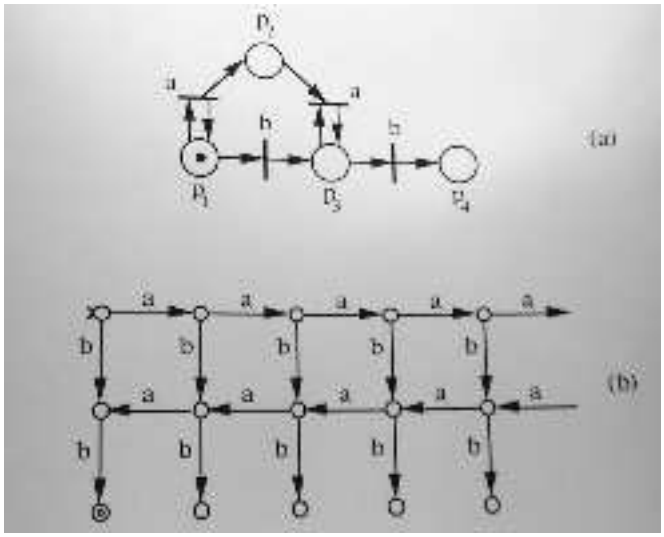


Figure 4.3: Blocking system $\mathbf{G}$ in Example 4.2.

Example 4.2. Let $\mathbf{G}$ be the PN generator in Figure 4.3.(a), with $M_0 = (1000)^T$ and set of final markings $F = \{(0001)^T\}$. The behaviors of this system are: $L_m(\mathbf{G}) = \{a^m b a^m b \mid m \geq 0\}$ and $L(\mathbf{G}) = \{\overline{a^m b a^n b} \mid m \geq n \geq 0\}$. From the reachability tree of the system, shown in Figure 4.3.(b), it is clear that the system is blocking. To avoid reaching a blocking state we require that p_2 be empty before firing the transition inputting into p_4. However, since p_2 is unbounded this may not be done with a simple P/T structure. It is possible to introduce an inhibitor arc from p_2 to the transition inputting into p_4. Inhibitor arcs increase the modeling power, and the analysis complexity, of Petri nets to that of a Turing machine, thus we cannot properly consider these models as P/T nets. Petri nets with inhibitory arcs have been studied in the context of Supervisory Control by Sreenivas and Krogh [Sreenivas 92].

We may formally prove that the prefix closure of the marked language of the system $\mathbf{G}$ discussed in Example 4.2 is not a P-type Petri net language. The proof is based on the pumping lemma for P-type PN languages, given in [Jantzen 87].

Lemma 4.1 (Pumping lemma). *Let $L \in \mathcal{P}$. Then there exist numbers k, l such that any string $w \in L$, with $\mid w \mid \geq k$, has a decomposition $w = xyz$ with $1 \leq \mid y \mid \leq l$ such that $xy^i z \in L, \forall i \geq 1$.*

Proposition 4.1. *$L = \{\overline{a^m b a^m b} \mid m \geq 0\}$ is not a P-type Petri net language.*

Proof. Given k and l according to the pumping lemma, let $w = a^k b a^k b$. Consider a partition $w = xyz$, with $1 \leq \mid y \mid \leq l$. Clearly $b \notin y$, since in this case $xy^i z \notin L$ for $i \neq 1$. However if $w = \underbrace{a^n}_{x}\underbrace{a^p}_{y}\underbrace{a^q b a^k b}_{z}$, $(n + p + q = k)$, or if $w = \underbrace{a^k b a^n}_{x}\underbrace{a^p}_{y}\underbrace{a^q b}_{z}$, $(n + p + q = k)$, then $xy^i z \notin L$ for $i \neq 1$. Hence L is not a P-type Petri net language. ◇

We conclude this section with the definition of a new class of L-type Petri net languages whose prefix closure can be generated by a deterministic nonblocking Petri net generator. This class will play an important role in characterizing the existence of Petri net supervisors, as we will discuss in the next section.

Definition 4.1. *A language $L \in \mathcal{L}$ (not necessarily deterministic) is said to be* deterministic P-closed *(DP-closed for short) if and only if its prefix closure is a deterministic P-type Petri net language, i.e., $\overline{L} \in \mathcal{P}_d$. The class of DP-closed Petri net languages is denoted $\mathcal{L}_{DP}$.*

As a side note, we point out that the class $\mathcal{L}_{DP}$ that we have defined does not coincide with any of the classes of PN languages generally considered in the literature [Jantzen 87]. The proof follows from the fact that $\mathcal{L}_{DP}$ is not closed under intersection, as we show in Example 4.5, while all previously defined classes of PN languages are closed under intersection.

4.3 Supervisor

A *supervisor* [Ramadge 87] is an agent that disables the controllable transitions of a system in order to restrict its behavior within a legal behavior. Here we recall the fundamental definitions, pointing to Appendix B.2 for a more detailed discussion.

Let us define a *control input* as a subset $\gamma \subseteq \Sigma$ satisfying $\Sigma_u \subseteq \gamma$ (i.e., all the uncontrollable events are present in the control input). If $a \in \gamma$, the event a is enabled by γ (permitted to occur), otherwise a is disabled by γ (prohibited from occurring); the uncontrollable events are always enabled. Let $\Gamma \subseteq 2^{\Sigma}$ denote the set of all the possible control inputs. Formally, a supervisor is a map $f : L(\mathbf{G}) \rightarrow \Gamma$ specifying, for each possible string of events $w \in L(\mathbf{G})$ generated by the system $\mathbf{G}$, the control input $\gamma = f(w)$ to be applied at that point. It is often the case that a supervisor is given as another discrete event system $\mathbf{S}$ that runs in parallel with $\mathbf{G}$. The control input at a given moment is represented by the events that are enabled in $\mathbf{S}$, thus the function f is implicit in the structure of $\mathbf{S}$.

The objective is to design a supervisor that selects control inputs in such a way that the controlled system's behavior is restricted within a legal specification language. We consider the case in which both the system $\mathbf{G}$ and the supervisor $\mathbf{S}$ are given as discrete event systems. The closed loop system under control is denoted $\mathbf{S}/\mathbf{G}$. Its closed behavior is $L(\mathbf{S}/\mathbf{G}) = L(\mathbf{S}) \cap L(\mathbf{G})$ and its controlled behavior is $L_m(\mathbf{S}/\mathbf{G}) = L(\mathbf{S}/\mathbf{G}) \cap L_m(\mathbf{G}) = L(\mathbf{S}) \cap L_m(\mathbf{G})$. Note that we are assuming that the supervisor does not mark strings, i.e., the marked strings of the system under control are all and only the marked strings of the uncontrolled system that survive under control.

The following two theorems, also reported in Appendix B.3, are due to Ramadge and Wonham [Ramadge 87, Ramadge 89b] and give necessary and sufficient conditions for the existence of a supervisor.

Theorem 4.2 ([Ramadge 87], P. 5.1). *For nonempty $L \subseteq L(\mathbf{G})$ there exists a supervisor $\mathbf{S}$ such that $L(\mathbf{S}/\mathbf{G}) = L$ if and only if L is prefix closed and controllable.*

Theorem 4.3 ([Ramadge 87], T. 6.1). *For nonempty $L \subseteq L_m(\mathbf{G})$ there exists a nonblocking supervisor $\mathbf{S}$ such that $L_m(\mathbf{S}/\mathbf{G}) = L$ if and only if L is $L_m(\mathbf{G})$-closed and controllable.*

If the system and the supervisor are Petri net generators, it is possible to construct a PN model of the closed loop system under control $\mathbf{S}/\mathbf{G}$ using the net counterpart of the intersection operator on languages [Peterson 81]. If $\mathbf{S}$ and $\mathbf{G}$ are deterministic generators (as we always assume) $\mathbf{S}/\mathbf{G}$ is deterministic as well. Note also that while the closed behavior of $\mathbf{S}/\mathbf{G}$ is $L(\mathbf{S}/\mathbf{G})$, the marked behavior of $\mathbf{S}/\mathbf{G}$ is not defined since the controlled behavior $L_m(\mathbf{S}/\mathbf{G})$ is not necessarily a deterministic L-type PN language.

4.4 Existence of Petri Net Supervisors

In this section, we present some theorems which give necessary and sufficient conditions for the existence of PN supervisors. In fact, although Theorem 4.2 and Theorem 4.3 specify under which conditions there exist supervisors for a given control problem, it may well be the case that these supervisors cannot be represented as Petri net generators, even if the system's behavior and the legal behavior are Petri net languages.

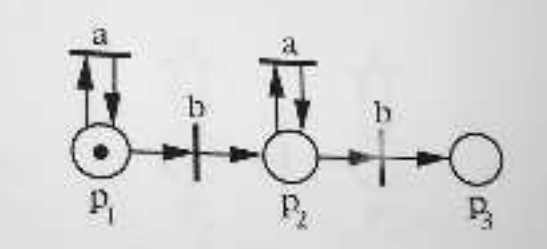


Figure 4.4: System $\mathbf{G}$ in Example 4.3 and Example 4.5.

The first two theorems regard the case in which we want to restrict the closed behavior of $\mathbf{G}$ within the limits of a legal behavior L.

Theorem 4.4. *Let $\mathbf{G}$ be a nonblocking PN and let $L \subseteq L(\mathbf{G})$ be a nonempty language. There exists a PN supervisor $\mathbf{S}$ such that $L(\mathbf{S}/\mathbf{G}) = L$ if and only if L is a controllable deterministic P-type PN language, i.e., $L \in \mathcal{P}_d \cap \mathcal{C}(\mathbf{G})$.*

Proof. (If) Since $L \in \mathcal{P}_d$ then there exists a PN generator $\mathbf{S} \ni L(\mathbf{S}) = L$. $\mathbf{S}$ is the desired supervisor. In fact, since $L \in \mathcal{C}(\mathbf{G})$ then $\mathbf{S}$, running in parallel with $\mathbf{G}$, will never block an uncontrollable transition of $\mathbf{G}$ and $L(\mathbf{S}/\mathbf{G}) = L(\mathbf{S}) \cap L(\mathbf{G}) = L$. (Only if) Since $L(\mathbf{S}/\mathbf{G}) = L$ then L is a P-type language for the PN representing the closed loop system and $L \in \mathcal{P}_d$. Also L is controllable according to Theorem 4.2. $\diamond$

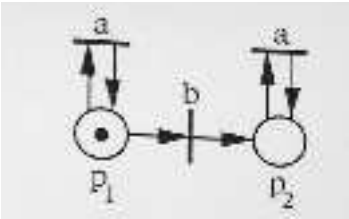
It may be interesting to consider the case in which the legal behavior L is not a subset of $L(\mathbf{G})$. In this case we are interested in restricting the system's behavior to $L \cap L(\mathbf{G})$, and we simply need to check whether $L \cap L(\mathbf{G})$ satisfies the conditions of the previous theorem. It may well be the case that $L \notin \mathcal{P}_d \cap \mathcal{C}(\mathbf{G})$ but $L \cap L(\mathbf{G}) \in \mathcal{P}_d \cap \mathcal{C}(\mathbf{G})$, i.e., a PN supervisor for this problem may exist even if L is not a PN language or if L is not closed and controllable. However, if $L \in \mathcal{P}_d$ we have the following theorem.

Theorem 4.5. *Let $\mathbf{G}$ be a nonblocking PN and let $L \subseteq \Sigma^*$, with $L(\mathbf{G}) \cap L \neq \emptyset$, and $L \in \mathcal{P}_d$. There exists a PN supervisor $\mathbf{S}$ such that $L(\mathbf{S}/\mathbf{G}) = L \cap L(\mathbf{G})$ if and only if $L \in \mathcal{C}(\mathbf{G})$.*

Proof. (If) The sets $\mathcal{P}_d$ and $\mathcal{C}(\mathbf{G})$ are closed under intersection, as proved, respectively, in [Jantzen 87, Ramadge 87]. Hence $L \cap L(\mathbf{G}) \in \mathcal{P}_d \cap \mathcal{C}(\mathbf{G})$ and by Theorem 4.4 there exists a PN supervisor $S \ni L(\mathbf{S}/\mathbf{G}) = L \cap L(\mathbf{G})$. (Only if) Since $L(\mathbf{S}/\mathbf{G}) = L \cap L(\mathbf{G})$ then $L \cap L(\mathbf{G}) \in \mathcal{P}_d$. Also $L \cap L(\mathbf{G})$ is controllable by Theorem 4.2, i.e., $\overline{[L \cap L(\mathbf{G})]}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{L \cap L(\mathbf{G})} \subseteq \overline{L} \Longrightarrow$ $\overline{[L \cap L(\mathbf{G})]}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{L}$ (since languages in $\mathcal{P}_d$ are prefix closed) $\Longrightarrow L\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{L}, \Longrightarrow$ $\overline{L}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{L}$ (since $L \in \mathcal{P}_d$), hence $L \in \mathcal{C}(\mathbf{G})$. $\diamond$

Let us consider now the case in which we want to restrict the marked behavior of $\mathbf{G}$ within the limits of a legal behavior L. In this case, unfortunately, the necessary requirements that L be controllable and $L_m(\mathbf{G})-$closed (by Theorem 4.3) are not sufficient to insure the existence of a nonblocking PN supervisor even if $L \in \mathcal{L}_d$. The following example discusses this case.

Example 4.3. Let $\mathbf{G}$ be the PN generator in Figure 4.4, with $\Sigma_u = \emptyset$, $M_0 = (100)^T$, and set of final markings $F = \{(001)^T\}$. The marked behavior of this system is: $L_m(\mathbf{G}) = \{a^*ba^*b\}$. Assume now that we want to restrict the marked behavior of $\mathbf{G}$ to $L = \{a^mba^mb \mid m \geq 0\}$ that is clearly controllable and $L_m(\mathbf{G})-$closed. However, since $L \notin \mathcal{L}_{DP}$ (Proposition 4.1) there exists no PN whose P-type language is $\overline{L}$. L is the L-type language of the system $\mathbf{E}$ in Figure 4.3.(a) with set of final marking $F = \{(0001)\}$, that however does not qualify as a supervisor for this problem, since $L_m(\mathbf{E}/\mathbf{G}) = L(\mathbf{E}) \cap L_m(\mathbf{G}) \supset L$.



captionSystem $\mathbf{G}$ in Example 5.2.

The example shows the need to introduce the further requirement that the legal behavior L be DP-closed for insuring the existence of a PN supervisor.

Theorem 4.6. *Let $\mathbf{G}$ be a nonblocking PN and let $L \subseteq L_m(\mathbf{G})$ be a nonempty language. There exists a nonblocking PN supervisor $\mathbf{S}$ such that $L_m(\mathbf{S}/\mathbf{G}) = L$ if and only if $L \in \mathcal{L}_{DP} \cap \mathcal{C}(\mathbf{G})$ and L is $L_m(\mathbf{G})-$closed.*

Proof. (If) Since $L \in \mathcal{L}_{DP}$ then there exists a PN $S \ni L(\mathbf{S}) = \overline{L} \subseteq L(\mathbf{G})$. $\mathbf{S}$ is the desired supervisor. In fact: since $L \in \mathcal{C}(\mathbf{G})$, then clearly $\overline{L} \in \mathcal{C}(\mathbf{G})$; since L is $L_m(\mathbf{G})-$closed, then $L_m(\mathbf{S}/\mathbf{G}) = L(\mathbf{S}) \cap L_m(\mathbf{G}) = \overline{L} \cap L_m(\mathbf{G}) = L$; $\mathbf{S}$ is nonblocking, since $L(\mathbf{S}/\mathbf{G}) = L(\mathbf{S}) \cap L(\mathbf{G}) = \overline{L} \cap L(\mathbf{G}) = \overline{L} = \overline{L(\mathbf{S}/\mathbf{G})}$. (Only if) Since $\mathbf{S}$ is a nonblocking supervisor, then $L \in \mathcal{C}(\mathbf{G})$ and L is $L_m(\mathbf{G})-$closed, by Theorem 4.3. We need to prove that $L = L_m(\mathbf{S}/\mathbf{G}) = L(\mathbf{S}) \cap L_m(\mathbf{G}) \in \mathcal{L}_{DP}$. First note that $L(\mathbf{S}) \in \mathcal{P}_d$, hence $L(\mathbf{S}) \in \mathcal{L} \supset \mathcal{P} \supset \mathcal{P}_d$ [Jantzen 87]. (Note that $L(\mathbf{S})$ although a deterministic P-type language may be a nondeterministic L-type language.) Since $\mathcal{L}$ is closed under intersection [Jantzen 87], then $L \in \mathcal{L}$. Also $\overline{L} = L(\mathbf{S}/\mathbf{G})$ by the non blocking hypothesis, hence $\overline{L} \in \mathcal{P}_d$ (since it is the deterministic P-type language of closed loop PN generator). It follows that $L \in \mathcal{L}_{DP}$. $\diamond$

Theorem 4.6 does not require L to be deterministic. In fact we want to restrict the marked behavior of $\mathbf{G}$ to L by means of a *non marking* deterministic supervisor, that effectively solely constrains the closed behavior of $\mathbf{G}$. Hence it is sufficient that $\overline{L}$ be a deterministic P-type language in order to construct a deterministic supervisor. The following example will clarify this point.

Example 4.4. Let $\mathbf{G}$ be the PN generator in Figure 4.5, with $\Sigma_u = \{b\}$, $M_0 = (10)^T$, and set of final markings $F = \{(01)^T\}$. The marked behavior of this system is: $L_m(\mathbf{G}) = \{a^*ba^*\}$. Assume now that we want to restrict the marked behavior of $\mathbf{G}$ to $L = \{a^mba^n \mid m \geq n \geq 0\}$. L is an L-type PN language and it can be proved that it is not deterministic, i.e., it cannot be accepted by a deterministic generator with a finite set of final states F. However $L \in \mathcal{L}_{DP}$, since $\overline{L}$ is the deterministic P-type language of the marked net $\mathbf{S}$ in Figure 4.6. Since L is controllable and $L_m(\mathbf{G})-$closed, $\mathbf{S}$ is a proper supervisor and is such that $L_m(\mathbf{S}/\mathbf{G}) = L$.

We would like to extend Theorem 4.6 to the case in which the legal behavior L is not a subset of $L_m(\mathbf{G})$. Unfortunately in this case the hypotheses of Theorem 4.6 are not sufficient to insure the existence of a PN supervisor. In fact, it may be possible that, although $L \in \mathcal{L}_{DP}$, $L \cap L_m(\mathbf{G}) \notin \mathcal{L}_{DP}$.

Proposition 4.2. *The class of DP-closed PN languages $\mathcal{L}_{DP}$ is not closed under intersection.*

The proof of the proposition follows from the following example.

Example 4.5. Let $\mathbf{G}$ be the PN generator in Figure 4.4, $M_0 = (100)^T$, and set of final markings $F = \{(001)^T\}$. The marked behavior of this system is: $L_m(\mathbf{G}) = \{a^*ba^*b\} \in \mathcal{L}_{DP}$. Consider the language $L = \{a^mb(bc)^*(a(bc)^*)^mb \mid m \geq 0\}$. Clearly $L \in \mathcal{L}_{DP}$ since it is the marked behavior of the nonblocking generator $\mathbf{E}$ in Figure 4.7, with the set of final markings $F = \{(0001)^T\}$. Unfortunately $L \cap L_m(\mathbf{G}) = \{a^mba^mb \mid m \geq 0\} \notin \mathcal{L}_{DP}$, as we have shown in Proposition 4.1.

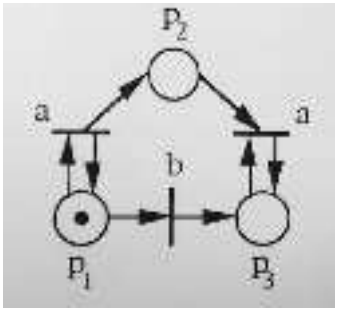


Figure 4.5: Supervisor **S** in Example 5.2.

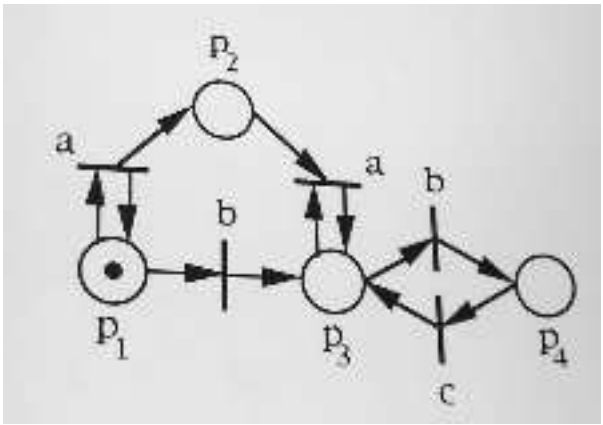


Figure 4.6: Generator **E** in Example 4.5 and Example 4.6.

The problem in the previous example is that L and $L_m(\mathbf{G})$ are *conflicting*, i.e., $\overline{L} \cap \overline{L_m(\mathbf{G})} \supset \overline{L \cap L_m(\mathbf{G})}$. Hence, while $\overline{L} \cap \overline{L_m(\mathbf{G})} \in \mathcal{P}_d$ (since $\mathcal{P}_d$ is closed under intersection) it may be the case that $\overline{L \cap L_m(\mathbf{G})} \notin \mathcal{P}_d$. If however the two languages are *not conflicting*, we have that $\overline{L \cap L_m(\mathbf{G})} = \overline{L} \cap \overline{L_m(\mathbf{G})} \in \mathcal{P}_d$.

Theorem 4.7. *Let $\mathbf{G}$ be a nonblocking PN and let $L \subseteq \Sigma^*$, with $L_m(\mathbf{G}) \cap L \neq \emptyset$. There exists a nonblocking PN supervisor $\mathbf{S}$ such that $L_m(\mathbf{S}/\mathbf{G}) = L \cap L_m(\mathbf{G})$ if $L \in \mathcal{L}_{DP} \cap \mathcal{C}(\mathbf{G})$, L is $L_m(\mathbf{G})-$closed, L and $L_m(\mathbf{G})$ are not conflicting.*

Proof. The class $\mathcal{L}$ and the set $\mathcal{C}(\mathbf{G})$ are closed under intersection [Jantzen 87, Ramadge 87]. Also, by the non-conflicting hypothesis, $\overline{L \cap L_m(\mathbf{G})} \in \mathcal{P}_d$. Hence $L \cap L_m(\mathbf{G}) \in \mathcal{L}_{DP} \cap \mathcal{C}(\mathbf{G})$. Finally since L is also $L_m(\mathbf{G})-$closed we can write: $\overline{L \cap L_m(\mathbf{G})} \cap L_m(\mathbf{G}) = \overline{L} \cap \overline{L_m(\mathbf{G})} \cap L_m(\mathbf{G}) = \overline{L} \cap L_m(\mathbf{G}) = L \cap L_m(\mathbf{G})$, i.e., $L \cap L_m(\mathbf{G})$ is $L_m(\mathbf{G})-$closed. By Theorem 4.6 there exists a nonblocking supervisor $S \ni L_m(\mathbf{S}/\mathbf{G}) = L \cap L_m(\mathbf{G})$. $\diamond$

The last theorem gives a sufficient, but not necessary, condition for the existence of a PN supervisor. In fact while: $L \in \mathcal{C}(\mathbf{G}) \Longrightarrow L \cap L_m(\mathbf{G}) \in \mathcal{C}(\mathbf{G})$ and L is $L_m(\mathbf{G})-$closed $\Longrightarrow$ $L \cap L_m(\mathbf{G})$ is $L_m(\mathbf{G})-$closed, the converse is not true. A necessary and sufficient condition for the existence of a PN supervisor in the case the legal behavior L is not a subset of $L_m(\mathbf{G})$ may be given, as a trivial corollary of Theorem 4.6, requiring that $L \cap L_m(\mathbf{G}) \in \mathcal{L}_{DP} \cap \mathcal{C}(\mathbf{G})$ and that $L \cap L_m(\mathbf{G})$ be $L_m(\mathbf{G})-$closed.

4.5 Petri Net Languages and Supremal Controllable Sublanguage

Our final result regards the closure of PN languages under the *supremal controllable sublanguage operator* $^\uparrow$ [Wonham 87]. This result has been known, at least partially, by the control community but has never been published before to the best of our knowledge.

Proposition 4.3. *The classes $\mathcal{P}_d$ and $\mathcal{L}_{DP}$ of PN languages are not closed under the $^\uparrow$ operator.*

The proof of this proposition will be given by means of the following example.

Example 4.6. Let $\mathbf{G}$ be the PN generator in Figure 4.8, with $\Sigma_u = \{a\}$, $M_0 = (1000)^T$, and set of final markings $F = \{(0001)^T\}$. Consider now the generator $\mathbf{E}$ in Figure 4.7 with the set of final markings $F = \{(0001)^T\}$. The two languages $L(\mathbf{E}) \in \mathcal{P}_d$ and $L_m(\mathbf{E}) \in \mathcal{L}_{DP}$ are not controllable. To show this we have drawn, in Figure 4.9, the reachability tree of the two systems; since $\mathbf{E}$ refines $\mathbf{G}$, we have represented the arcs that belong to the reachability tree of both marked nets with continuous lines, while the arcs that only belong to the reachability tree of $\mathbf{G}$ have been represented by dotted lines. $L(\mathbf{E})$ and $L_m(\mathbf{E})$ are not controllable because of the presence of the dotted arcs associated to the uncontrollable transition a. If we apply the $^\uparrow$ operator, we obtain the two supremal controllable sublanguages: $L(\mathbf{E})^\uparrow = \{\overline{a^m b a^m (bc)^* b} \mid m \geq 0\}$ and $L_m(\mathbf{E})^\uparrow = \{a^m b a^m (bc)^* b \mid m \geq 0\}$, that are the closed and marked behavior of the generator in Figure 4.10. $L(\mathbf{E})^\uparrow \notin \mathcal{P}_d$, as can be proved using the pumping lemma in the same way we have done in Proposition 4.1 (the string $w = a^k b a^k b$ may be used to prove that no pumping is possible). $L_m(\mathbf{E})^\uparrow \notin \mathcal{L}_{DP}$, since $\overline{L_m(\mathbf{E})^\uparrow} = L(\mathbf{E})^\uparrow \notin \mathcal{P}_d$.

4.6 Conclusions

The results of this chapter show that although unbounded PN do not have all desirable properties from the point of view of supervisory control, there are cases in which PN supervisors may be constructed.

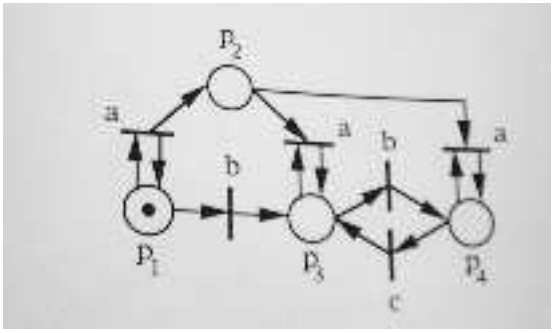


Figure 4.7: Generator $\mathbf{G}$ in Example 4.6.

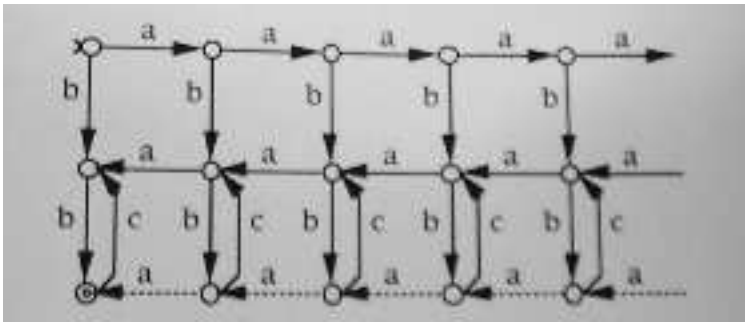


Figure 4.8: Reachability tree of $\mathbf{G}$ and $\mathbf{E}$ in Example 4.6.

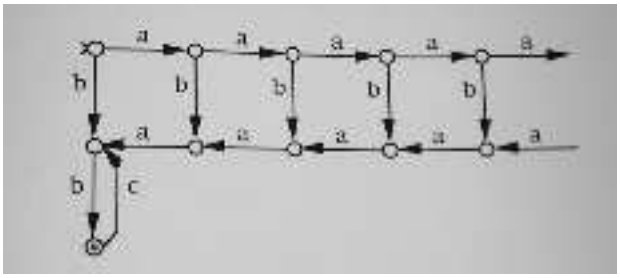


Figure 4.9: Generator of $L(\mathbf{E})^{\uparrow}$ and $L_m(\mathbf{E})^{\uparrow}$ in Example 4.6.

The main difficulties when using Petri nets as discrete event models for supervisory control stems out from the fact that not all L-type PN languages are DP-closed and from the fact that the class $\mathcal{L}_{DP}$ of DP-closed languages is not closed under intersection. Another undesirable property is given by the fact that the classes $\mathcal{P}_d$ and $\mathcal{L}_{DP}$ are not closed under the $^{\uparrow}$ operator.

In the remaining part of the thesis we will consider simpler Petri net models, whose reachability set is finite.

Chapter 5

SUPERVISORY DESIGN USING PETRI NETS

5.1 Introduction

In this chapter we discuss how Petri net models may be used to design supervisors within the framework of Supervisory Control. The purpose is that of applying a model that has a great representational power in the theoretical approach of Ramadge and Wonham. The discussion is focused at the model level: we discuss the supervisory design algorithm and its counterpart on the structure of a Petri net [DiCesare 91, Giua 91]. The design requires two steps. In the first step, a coarse structure for a supervisor is synthesized by means of concurrent composition of different modules. In the second step, the structure is refined to avoid reaching undesirable markings.

The chapter is structured as follows. In Section 5.2, we review the monolithic supervisory design as formulated by Wonham and present a design based on the concurrent composition operator. In Section 5.3, we show that this design is well suited for Petri nets models. In particular, we discuss how to refine a coarse structure for a supervisor, by introducing new arcs (and possibly duplicating transitions) to avoid reaching undesirable markings. This procedure may always be applied when the net is conservative. In Section 5.4, we discuss the advantages of PN over state machines, as far as the transition structure of the model is concerned.

5.2 Monolithic Supervisor Design

A supervisory design algorithm has been presented by Wonham [Wonham 88a]. The algorithm is based on language operators and it may be implemented using different models. The only requirement is that the model specify the closed and marked behavior of a system. We also require that the linguistic operators introduced in Section 3.2 be extended to the model. For example, given two DES $\mathbf{G}_1, \mathbf{G}_2$, with closed behaviors $L(\mathbf{G}_1), L(\mathbf{G}_2)$ and marked behaviors $L_m(\mathbf{G}_1)$, $L_m(\mathbf{G}_2)$ we denote: $\mathbf{G} = \mathbf{G}_1 \parallel \mathbf{G}_2$ the DES whose behaviors are: $L(\mathbf{G}) = L(\mathbf{G}_1) \parallel L(\mathbf{G}_2)$ and $L_m(\mathbf{G}) = L_m(\mathbf{G}_1) \parallel L_m(\mathbf{G}_2)$. In a similar fashion, we can extend other linguistic operators to operators on the systems.

Assume we are given m nonblocking DES $\mathbf{G}_1, \ldots, \mathbf{G}_m$ working concurrently. The alphabet of these systems are $\Sigma_1, \ldots, \Sigma_m$ and they are disjoint, i.e., $\Sigma_i \cap \Sigma_j = \emptyset$ for every $i \neq j$. We want to enforce some specifications on the joint behavior of these systems, represented by n different nonblocking generators $\mathbf{H}_1, \ldots, \mathbf{H}_n$. Each specification generator $\mathbf{H}_j$ is defined on an alphabet $\Sigma_{H_j} \subseteq \Sigma_1 \cup \ldots \cup \Sigma_m$ and effectively constrains only the events that belong to its alphabet.

A *monolithic supervisor* may be constructed using the following algorithm [Wonham 88a].

Algorithm 5.1. Wonham Supervisory Design

1. *Construct by shuffling* $\mathbf{G} = \mathbf{G}_1 \parallel_d \ldots \parallel_d \mathbf{G}_m$. $\mathbf{G}$ *models the joint concurrent behavior of the overall system under the assumption that the actions of the single subsystems are asynchronous and independent. The alphabet of* $\mathbf{G}$ *is* $\Sigma = \Sigma_1 \cup \ldots \cup \Sigma_m$.

2. *Augment the specifications, selflooping each* $\mathbf{H}_j$ *with the event labels present in* $\mathbf{G}$ *but unconstrained by* $\mathbf{H}_j$*, i.e., the labels in* $\Sigma_{e_j} = \Sigma \backslash \Sigma_{H_j}$. *Call these new generators* $\mathbf{H}'_1, \ldots, \mathbf{H}'_n$.

3. *Construct by intersection* $\mathbf{H} = \mathbf{H}'_1 \cap \ldots \cap \mathbf{H}'_n$. $\mathbf{H}$ *represents the global specification we want to enforce on the behavior of* $\mathbf{G}$.

4. *Construct by intersection* $\mathbf{E} = \mathbf{H} \cap \mathbf{G}$. $\mathbf{E}$ *represents the largest behavior of the overall system that satisfies all the constraints imposed by the specifications.*

5. *The generator* $\mathbf{E}$ *obtained through the above construction, in the general case does not represent a proper supervisor, since the following two properties are not automatically ensured:*

   - nonblockingness*, i.e., the generator* $\mathbf{E}$ *may contain blocking states from which a final state cannot be reached;*
   - controllability*, i.e., the language* $L(\mathbf{E})$ *may not be controllable with respect to* $L(\mathbf{G})$. *This means that when* $\mathbf{G}$ *and* $\mathbf{E}$ *run in parallel it may be possible to reach a state from which an uncontrollable event is enabled in* $\mathbf{G}$ *but is not enabled in* $\mathbf{E}$.

   We have to further minimally restrict the behavior of $\mathbf{E}$*, to obtain a nonblocking and controllable generator* $\mathbf{S}$. *This is the counterpart, on the DES structure, of the supremal controllable sublanguage operator.*

The system $\mathbf{S}$ obtained through this procedure gives us the transition structure of a supervisor. We will give an example of this construction using state machine models.

Example 5.1. Consider the systems $\mathbf{G}_1$ and $\mathbf{G}_2$ in Figure 5.1 and the specification $\mathbf{H}$. We may think of $\mathbf{G}_1$ as a robot that picks up one part (event a) and loads the part on a machine (event b). $\mathbf{G}_2$ is a machine that may start working (event c) and may output the produced part on conveyor A or B (respectively events d and e) before returning to the idle state. The specification we consider, represented by the generator $\mathbf{H}$, specifies that the machine may start working only after it has been loaded (events b and c must occur alternatively), and that the robot may load a new part on the machine only after the previously loaded part has been outputted on conveyor A (events d and b must occur alternatively). We will assume that the only uncontrollable event for this problem is event b.

The first step of design requires the construction of $\mathbf{G} = \mathbf{G}_1 \parallel_d \mathbf{G}_2$, shown in Figure 5.2.

In the second step we construct the augmented specification $\mathbf{H}'$, shown in Figure 5.3, by selflooping $\mathbf{H}$ with the unconstrained events, i.e., with a, e. Since $\mathbf{H}$ is the only specification generator, step 3 is not required in this example.

In the fourth step we construct the system $\mathbf{E} = \mathbf{G} \cap \mathbf{H}'$, shown in Figure 5.4. The system $\mathbf{E}$ is not a supervisor, because it contains blocking states (from which the final marking may not be reached), and also non controllable states. As an example, after $\mathbf{G}$ has executed the string of events $w = aba$, events b and c may occur in $\mathbf{G}$ while the control pattern computed by $\mathbf{E}$ contains only the event c. Since b is an uncontrollable event, the language generated by $\mathbf{E}$ is not controllable.

If we remove the blocking and noncontrollable states, we obtain the nonblocking and controllable generator $\mathbf{S}$ in Figure 5.5, which is a proper supervisor for this problem.

The supervisor $\mathbf{S}$ constructed using Algorithm 5.1 is called *monolithic*, because it represents both a *supervisor* and a *closed loop model* of the system under control. In fact $L(\mathbf{S}) = L(\mathbf{S}/\mathbf{G})$. It is often the case that a simpler supervisor exists. For the control problem in Example 5.1, a

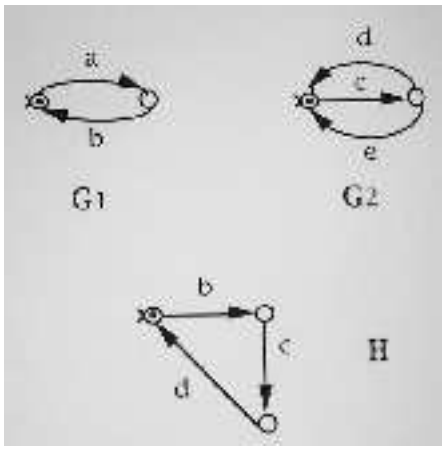


Figure 5.1: Systems and specification for the control problem in Example 5.1.

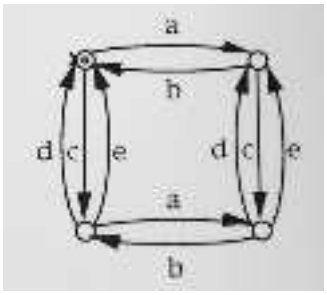


Figure 5.2: Shuffle system $\mathbf{G} = \mathbf{G}_1 \parallel_d \mathbf{G}_2$ in Example 5.1.

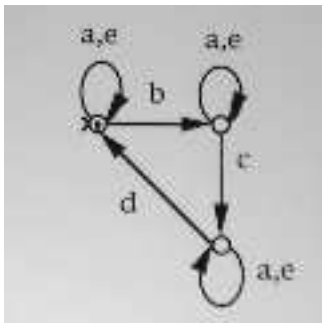


Figure 5.3: Specification generator $\mathbf{H}'$ in Example 5.1.

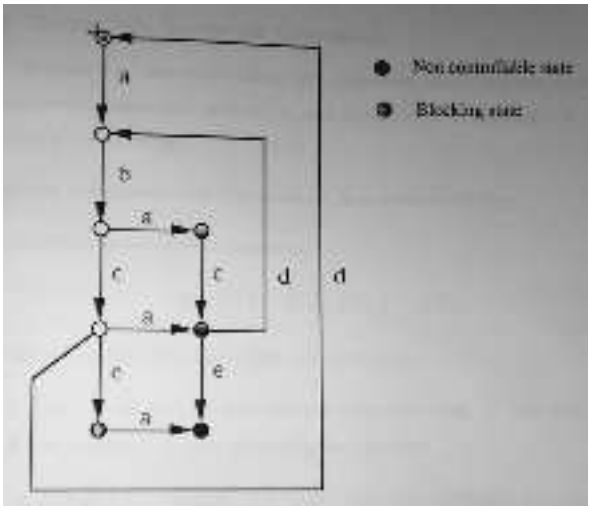


Figure 5.4: System $\mathbf{E} = \mathbf{G} \cap \mathbf{H}'$ for the control problem in Example 5.1.

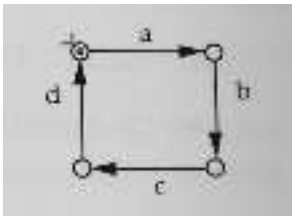


Figure 5.5: Monolithic supervisor **S** for control problem in Example 5.1.

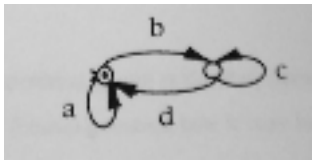


Figure 5.6: A reduced structure supervisor **S**' for the control problem in Example 5.1.

reduced structure supervisor $\mathbf{S}'$ is shown in Figure 5.6. We may easily check that $L(\mathbf{S}/\mathbf{G}) = L(\mathbf{S}'/\mathbf{G}) \subset L(\mathbf{S}')$.

Note also that since we have associated a final state with the specification $\mathbf{H}$, the supervisor we have constructed is a *marking supervisor* according to the terminology in Appendix B.2.

5.2.1 Design Using Concurrent Composition

We propose an alternative design for a supervisor, where only one operator, the *concurrent composition* operator, is used to construct the generator $\mathbf{E}$. The algorithm is the following.

Algorithm 5.2. Concurrent Composition Supervisory Design

1. *Construct by concurrent composition*

$$\mathbf{E} = \mathbf{G}_1 \parallel \ldots \parallel \mathbf{G}_m \parallel \mathbf{H}_1 \parallel \ldots \parallel \mathbf{H}_n;$$

2. *Refine* $\mathbf{E}$ *so that it is nonblocking and controllable.*

It is easy to see that the two approaches are equivalent. In fact with Algorithm 5.1 we construct a system generating the language

$$E_1 = [L(\mathbf{G}_1) \parallel_d \ldots \parallel_d L(\mathbf{G}_m)] \cap [L(\mathbf{H}_1) \parallel_d \Sigma_{e_1}^*] \cap \ldots \cap [L(\mathbf{H}_n) \parallel_d \Sigma_{e_n}^*],$$

and with Algorithm 5.2 we construct a system generating the language

$$E_2 = L(\mathbf{G}_1) \parallel \ldots \parallel L(\mathbf{G}_m) \parallel L(\mathbf{H}_1) \parallel \ldots \parallel L(\mathbf{H}_n).$$

Since $(\forall w \in \Sigma^*)\ \ w \in [L(\mathbf{H}_j) \parallel_d \Sigma_{ej}^*] \Longleftrightarrow w \uparrow_{H_j} \in L(\mathbf{H}_j)$ (here $w \uparrow_{H_j}$ is the projection of string w on alphabet Σ_{H_j}), we can see that:

$$\begin{aligned} E_1 &= \{w \mid w \in [L(\mathbf{G}_1) \parallel_d \ldots \parallel_d L(\mathbf{G}_m)],\ w \in [L(\mathbf{H}_j) \parallel_d \Sigma_{ej}^*]\ (j = 1, \ldots, n)\} \\ &= \{w \mid w \uparrow_i \in L(\mathbf{G}_i)\ (i = 1, \ldots, m),\ w \uparrow_{H_j} \in L(\mathbf{H}_j)\ (j = 1, \ldots, n)\} \\ &= E_2. \end{aligned}$$

The concurrent composition design is simpler, since it requires only one operator. We will show in the following section how it may be easily implemented using Petri net models.

5.3 Monolithic Design Using Petri Nets

In this section, we will use Petri net models in the design of a supervisor, following the Algorithm 5.2. The coarse structure for a supervisor, i.e., the system $\mathbf{E}$, may be efficiently constructed and the properties of nonblockingness and controllability may be expressed as properties of the net system $\mathbf{E}$. Secondly, we will discuss how $\mathbf{E}$ may be refined to obtain a supervisor $\mathbf{S}$ without major changes in its structure.

The net counterpart of the concurrent composition operator, described in Appendix A.5, requires merging transitions with the same label. This construction is based on the structure of the net and does not require the explicit enumeration of its state space.

Example 5.2. For the control problem discussed in Example 5.1 we have represented in Figure 5.7 the systems and specification as Petri nets. The system $\mathbf{E} = \mathbf{G}_1 \parallel \mathbf{G}_2 \parallel \mathbf{H}$ obtained by concurrent composition is shown in Figure 5.8.

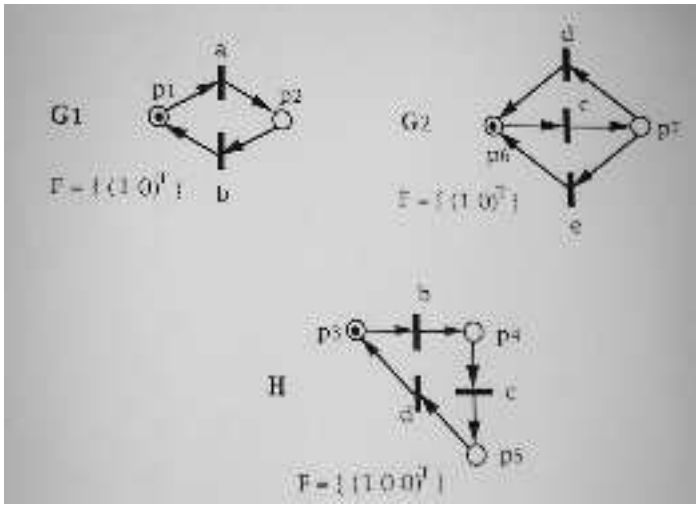


Figure 5.7: Systems and specifications as Petri nets for the control problem in Example 5.2.

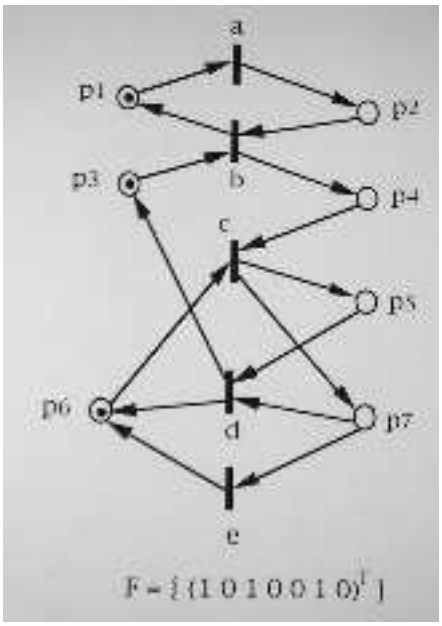


Figure 5.8: System $\mathbf{E} = \mathbf{G}_1 \parallel \mathbf{G}_2 \parallel \mathbf{H}$ as a Petri net for the control problem in Example 5.2.

It is important to note how the properties of interest, namely nonblockingness and controllability, may be expressed in terms of net properties. We will use the notation for concurrent systems presented in Appendix A.5. Let $\mathbf{G} = \mathbf{G}_1 \parallel \ldots \parallel \mathbf{G}_m$ and $\mathbf{H} = \mathbf{H}_1 \parallel \ldots \parallel \mathbf{H}_n$, be the system and specification components of $\mathbf{E}$. Assume that $\mathbf{E} = (N, \ell, M_0, F)$, $\mathbf{G} = (N_1, \ell_1, M_{0,1}, F_1)$, and $\mathbf{H} = (N_2, \ell_2, M_{0,2}, F_2)$.

The system $\mathbf{E}$ is nonblocking if and only if a final marking may be reached from any reachable marking, i.e., if and only if

$$(\forall M \in R(N, M_0))\ (\exists M_f \in F)\ [M_f \in R(N, M)].$$

Conversely, any reachable marking M' such that $(\nexists M_f \in F)\ [M_f \in R(N, M')]$ is a blocking marking, and we should prevent the system from reaching it.

Let $T_u \subseteq T$ be the set of uncontrollable transitions of $\mathbf{E}$, i.e., the transitions labeled by uncontrollable events. Given a marking $M \in R(N, M_0)$, let $M \uparrow_1$ ($M \uparrow_2$) be the restriction of M to the places of $\mathbf{G}$ ($\mathbf{H}$). Then $\mathbf{E}$ will be controllable if and only if it is not possible to reach a marking M such that an uncontrollable transition t is enabled by $M \uparrow_1$ in $\mathbf{G}$, but it is not enabled by $M \uparrow_2$ in $\mathbf{H}$. In other words, $\mathbf{E}$ is controllable if and only if

$$(\forall t \in T_u)\ (\nexists M \in R(N, M_0))\ [M \uparrow_2 \ngeq Pre_2(\cdot, t) \wedge M \uparrow_1 \geq Pre_1(\cdot, t)].$$

Once the coarse structure of a candidate supervisor is constructed by means of concurrent composition, we need to refine it to obtain a nonblocking and controllable generator. We have to remove a set of undesirable states and all the transitions leading to them. We assume in this section that the knowledge of which states are undesirable, and which are the transitions leading to them, is given.

The next example shows the problems involved in the refinement of a net.

Example 5.3. Let us consider the system $\mathbf{E}$ constructed in Example 5.2. Here the controllable event set is $\Sigma_c = \{a, c, d, e\}$ and the uncontrollable event set is $\Sigma_u = \{b\}$. $\mathbf{E}$ is blocking and uncontrollable. The undesirable markings, that should be removed, are those shown in the reachability tree in Figure 5.9, equivalent to the state machine model in Figure 5.4. Refining the PN is complex. First, we should certainly remove the transition labeled by e since its firing always leads to an undesirable state and it is controllable. After removal of this transition, the transition labeled by a will be enabled by the following reachable markings: $M' = (1010010)^T, M'' = (1001010)^T, M''' = (1000101)^T$. We want to block the transition labeled a when the markings M'' and M''' are reached. Since

$$M'(p_3) = 1 > M''(p_3) = M'''(p_3) = 0,$$

we can add an arc from p_3 to a and from a to p_3 as in Figure 5.10.

The following algorithm can be given for the refinement of a net.

Algorithm 5.3. *Let t be a transition to be controlled, i.e., a transition leading from an admissible marking to an undesirable marking. Let a be its label.*

1. *Determine the set of admissible reachable markings that enable t, and partition this set into the disjoint subsets $\overline{M}_e$ (the markings from which t should be allowed to fire), and $\overline{M}_d$ (the markings from which t should not be allowed to fire, to avoid reaching an undesirable marking). If $\overline{M}_e = \emptyset$ remove t and stop, else continue.*

2. *Determine a construct in the form:*

$$\begin{aligned} \mathcal{U}(M) = \ & [(M(p_1^1) \geq n_1^1) \wedge \ldots \wedge (M(p_{k1}^1) \geq n_{k1}^1)] \vee \\ & \ldots \\ & \vee [(M(p_1^l) \geq n_1^l) \wedge \ldots \wedge (M(p_{kl}^l) \geq n_{kl}^l)], \end{aligned}$$

such that $\mathcal{U}(M) =$ TRUE if $M \in \overline{M}_e$, and $\mathcal{U}(M) =$ FALSE if $M \in \overline{M}_d$.

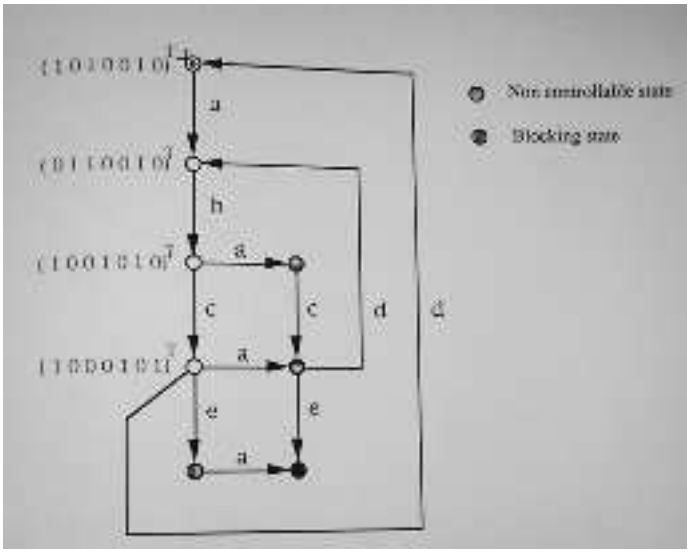


Figure 5.9: Reachability tree of the system **E** in Example 5.2 and Example 5.3.

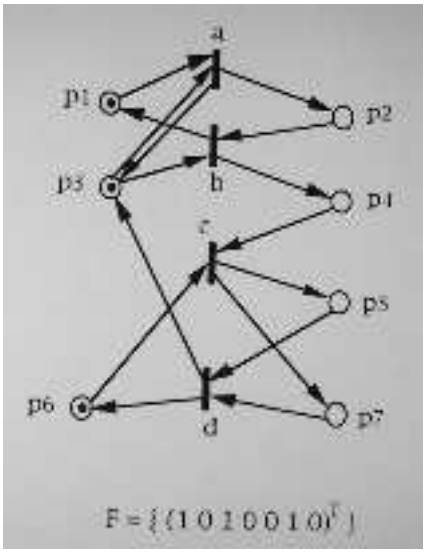


Figure 5.10: Supervisor **S** as a Petri net for the control problem in Example 5.2 and Example 5.3.

3. *Replace transition t with l transitions $t^1, \ldots, t^l$ labeled a. The input (output) arcs of transition $t^j,\ j = 1, \ldots, l$, will be those of transition t plus n_i^j arcs inputting from (outputting to) place $p_i^j,\ i = 1, \ldots, k_j$.*

It is clear that following this construction there is an enabled transition labeled a for any marking in $\overline{M}_e$, while none of these transitions are enabled by a marking in $\overline{M}_d$. We also note that in general several constructs of this form may be determined. The one which requires the minimal number of transitions, i.e., the one with smaller l, is preferable.

The following theorem gives a sufficient condition for the applicability of the algorithm.

Theorem 5.1. *The construct of Algorithm 5.3 can always be determined if the net is conservative.*

Proof: A net is conservative if there exists an integer vector $Y > \vec{0}$ such that for any two markings M and M' reachable from the initial marking $Y^T M = Y^T M'$. Hence if $M \neq M'$ there exists a place p such that $M(p) > M'(p)$. Also the set of reachable markings is finite.

On a conservative net, consider $M_i \in \overline{M}_e$, $M_j \in \overline{M}_d$. We have that $\overline{M}_e$ and $\overline{M}_d$ are finite sets and also there exists a place p_{ij} such that $M_i(p_{ij}) = n_{ij} > M_j(p_{ij})$. Hence

$$\mathcal{U}(M) = \bigvee_{i \in \overline{M}_e} \left[\bigwedge_{j \in \overline{M}_d} (M(p_{ij}) \geq n_{ij}) \right]$$

is a construct for Algorithm 5.3. ◇

According to this Theorem we may always find a refinement construct for conservative nets. Unfortunately, the construct may contain up to $|\overline{M}_e|$ *OR* clauses, i.e., up to $|\overline{M}_e|$ transitions may be substituted for a single transition to control. Note, however, that it is often possible to determine a simpler construct as in Example 5.3, where the construct for the transition labeled a was $\mathcal{U}(M) = [M(p_3) \geq 1]$.

5.4 Petri Nets and State Machines

The design using concurrent composition appears to be highly efficient when using Petri net structures. Here we would like to discuss the main advantages of using Petri nets rather than state machines in the supervisory design.

1. PN have a higher language complexity than state machines. Hence they have the possibility of describing a larger class of systems.

   The concurrent composition design algorithm may be used to construct the system $\mathbf{E}$ even if we consider Petri net models with infinite state space, since it is based on the finite structure of the net. However, as suggested by Theorem 5.1, since systems with infinite state space are not conservative it may be impossible to refine the net in step 2 of the algorithm. This was also expected from the results of Chapter 4 since the nonblocking generator of the supremal controllable sublanguage of $\mathbf{E}$ may not have a Petri net representation.

2. Suppose we restrict ourselves to consider only conservative Petri net models. Conservative PN are essentially equivalent to state machines, since the number of reachable markings (i.e., the number of "states" of the model) is finite. However, since the states of a PN are represented by the possible markings and not by the places, they allow a compact description, i.e., the structure of the net may be maintained small in size even if the number of the markings grows.

   To express the difference between the two approaches in quantitative terms, assume that we start with bottom level modules that are state machines. Furthermore assume that in each

module each symbol labels at most one transition. Consider k modules $\mathbf{G}_1, \ldots, \mathbf{G}_k$, and let m_i and n_i be the number of states (i.e., places) and transitions of each of them. The system $\mathbf{G} = \mathbf{G}_1 \parallel \ldots \parallel \mathbf{G}_k$ can be constructed as a state machine or as an ordinary Petri net.

- When **G** is constructed as a state machine the number m of states and the number n of transitions will be:

$$m \leq \prod_{i=1}^{k} m_i, \quad n \leq \sum_{i=1}^{k} \left(n_i \prod_{j \neq i} m_j \right)$$

  The upper bound is reached when the alphabets of the modules are disjoint; in this case the coupling of the modules is so weak that the overall state set will be identical to the cartesian product of the states of the modules. The bound gives however a clear idea of the exponential growth of the model even in the general case when the alphabets of the modules are not disjoint [Tadmor 89].

- When **G** is constructed as an ordinary Petri net, following the algorithm in Appendix A.5, we have a number of places equal to: $m = \sum_{i=1}^{k} m_i$ and a number of transitions: $n = \|\Sigma\| \leq \sum_{i=1}^{k} n_i$ where Σ is the alphabet associated to **G**.
  However, it is necessary to point out that in the phase of the refinement of the coarse structure of the supervisor the number of transitions introduced in Algorithm 5.3 may in the worst case approach the size of $\overline{M}_e$ as suggested by Theorem 5.1.

3. Petri nets allow modular synthesis, i.e., the net can be considered as composed of interrelated subnets, in the same way as a complex systems can be regarded as composed of interacting subsystems. In Figure 5.8, e.g., we may still recognize the individual modules that compose the system **E**, while in the state machine representation in Figure 5.4 this is not possible. This permits us to express controllability as a property of the generator **E**.

5.5 Conclusions

We can summarize the results of this chapter as follows:

- For regular languages, Ramadge and Wonham have proved that it is always possible to determine the supremal controllable sublanguage of a given configuration of systems and specifications with a finite procedure (See Appendix B, Theorem B.3).

- For DES modeled with conservative Petri nets (that generate regular languages) we can always refine the net in order that the supremal controllable behavior is achieved. In the refinement, following Algorithm 5.3, the modular structure of the net is preserved (but in some cases it may be necessary to introduce a large number of transitions).

Hence conservative Petri nets may be used within the framework of supervisory control, providing a more compact representation in terms of transition structure when compared with state machine models.

There is, however, another problem to be addressed. In this chapter we have assumed that the set of undesirable markings is given. In practice, a brute force approach to determine this set requires the construction of the reachability tree. PN may offer means to solve this problem in more efficient ways.

- Use incidence matrix analysis to determine whether undesirable markings are reachable. Although in the general case this analysis gives only a necessary condition for reachability, there are classes of nets such that a necessary and sufficient condition may be derived [Giua 90b, Murata 89].

- Derive synthesis procedures that insure the desired properties for the composed system. In this case we have efficient “closed form solutions” at the cost of limiting our design to special classes of nets [Datta 84, Datta 86, Krogh 86, Krogh 91].

These approaches will be investigated in the next chapters.

Chapter 6

INCIDENCE MATRIX ANALYSIS

6.1 Introduction

This chapter discusses how incidence matrix analysis for Petri net models may be used to validate supervisors for the control of discrete event systems. We will consider a class of P/T nets called *Elementary Composed State Machines* (ECSM). The most interesting property of this class of nets lies in the fact that the set of reachable markings is an (integer) convex set and that the set of linear inequalities $A \cdot M \geq A \cdot M_0$ which defines it may be computed from the analysis of the simple state machine modules that compose the net.

In classic incidence matrix analysis, the set of reachable markings of a system $\langle N, M_0 \rangle$ is approximated by the solutions of the *state equation*, i.e., by the set

$$PR(N, M_0) = \{M \in I\!N^{|P|} \mid (\exists \vec{\sigma} \in I\!N^{|T|})[M = M_0 + C \cdot \vec{\sigma}]\},$$

where C is the incidence matrix of the net. In another approach, a *basis of P-semiflows* B is used to approximate the reachability set, defining the set

$$PR^B(N, M_0) = \{M \in I\!N^{|P|} \mid B^T \cdot M = B^T \cdot M_0\}.$$

The first approximation is generally better, in the sense that

$$R(N, M_0) \subseteq PR(N, M_0) \subseteq PR^B(N, M_0)$$

However, the computation of $PR^B(N, M_0)$ does not involve the firing count vector $\vec{\sigma}$, and this feature has additional advantages that we will discuss in the following.

We propose an approach similar to the computation of $PR^B(N, M_0)$ where we use, in addition to the equations derived from the P-semiflows, inequalities derived from the basic traps of the net and we define

$$PR^A(N, M_0) = \{M \in I\!N^{|P|} \mid A \cdot M \geq A \cdot M_0\}.$$

The matrix A is computed from the analysis of the structure of net N. Furthermore, we show that for ECSM, the class of nets that we consider here, $PR^A(N, M_0) = R(N, M_0)$, i.e., the set of integer solutions of the inequality $A \cdot M \geq A \cdot M_0$ is exactly the set of reachable markings [Giua 92d].

There are significant differences between our approach and the analysis based on the state equation.

- The use of the incidence matrix does not permit verifying propositions such as

$$(\forall M \in PR(N, M_0))[M_f \in PR(N, M)]$$

with a linear algebraic formalism; this proposition, as we will see, expresses the nonblocking property of a system. This is because the set $PR(N, M_0)$ is defined in terms of linear equations containing the variables M and $\vec{\sigma}$. We define the set of reachable markings as the solution of a set of inequalities which do not contain the firing count vector $\vec{\sigma}$, and we will be able to write simple integer programming problems to study nonblocking properties of systems.

- For the class of ECSM nets, the state equation gives only necessary but not sufficient conditions for reachability, since it may contain spurious solutions (solutions which do not correspond to reachable markings), i.e., in general it holds $PR(N, M_0) \supset R(N, M_0)$. However, for the same class of nets there exists a matrix A such that $PR^A(N, M_0) = R(N, M_0)$, i.e., the reachability set of an ECSM may be exactly described by a set of linear inequalities, without spurious solutions.

- Note that since the domain $PR^A(N, M_0)$ represents the integer solutions of a set of linear inequalities, it is a convex set of integers. Thus, there exists a matrix A such that $PR^A(N, M_0) = R(N, M_0)$ if and only if the reachability set of system $\langle N, M_0 \rangle$ is a convex set. If a net is such that the state equation does not contain spurious solutions, this does not imply that its reachability set is a convex set. As an example, it has been proved that the state equation of acyclic nets does not contain spurious solutions [Murata 89]. The net we will discuss in Example 6.2 is acyclic but does not have a convex reachability set. This means that there are classes of nets such that $PR(N, M_0) = R(N, M_0)$ and such that there does not exist a matrix A such that $PR^A(N, M_0) = R(N, M_0)$.

In our approach determining if a marking is reachable requires the use of Integer Programming. Integer Programming problems may not be solved, in general, in polynomial time. However, it is possible to relax the constraint that the solution of $A \cdot M \geq A \cdot M_0$ be integer to obtain a sufficient condition for the validation of the net. Thus, we may use Linear Programming techniques to prove that a given undesirable marking is not reachable.

The model considered in this chapter is based on state machine P/T nets with multiple tokens. State machines may model choice, since a place may have more that one outgoing arc, but strongly limit the possibility of modeling concurrency, since the only concurrent behavior is given by the presence of multiple tokens in the net. To describe concurrent systems, the model is extended by composing the state machine modules through concurrent composition, an operator that requires the merging of common transitions.

In this approach it is necessary to restrict the type of compositions considered, in order to guarantee some important properties [Giua 90b]. The final model, called Elementary Composed State Machines (ECSM nets), can model both choice and concurrent behavior, and at the same time has the property that the space of reachable markings is a linear integer convex set, i.e., it is given by the integer solutions of a set of linear inequalities.

Section 2 deals with state machines and reachability using incidence matrix analysis and shows how it is possible to derive the set of linear inequalities that defines the space of reachable markings. In Section 3 is defined the class of Elementary Composed State Machine nets and the results derived in Section 2 are extended to this class. Section 4 finally shows how this approach may be applied to the validation of supervisors for the control of discrete event systems.

6.2 Incidence Matrix Analysis for State Machines

In this section we discuss two ways of determining if a marking of a state machine is reachable from the initial marking. In the first approach we consider the state equation of the net. In the

second we show how the reachability set may be described by a set of equations that do not involve the firing count vector.

State machines represent a very simple PN model that has been extensively investigated. For instance, Murata [Murata 89] reports several results on the liveness and reachability characterization of this class of nets. However, the focus is generally on live state machines (i.e., strongly connected state machines). In the framework of Supervisory Control, the requirement that the model be live is not important, while it is required that the system be *nonblocking*, i.e., that from any reachable marking it may be possible to reach a final marking. This motivates the attention given in this chapter to state machines that are not necessarily live.

6.2.1 State Equation

The use of the state equation for characterizing the reachability set of a PN is based on the following observation.

Note 6.1. *Let $\langle N, M_0\rangle$ be a marked net and C its incidence matrix.*

$$M \in R(N, M_0) \Longrightarrow (\exists \vec{\sigma} \in \mathbb{N}^m)\ [M = M_0 + C \cdot \vec{\sigma}].$$

The existence of a vector $\vec{\sigma}$ satisfying the previous state equation gives a necessary but not sufficient condition for the reachability of a given marking. In the general case, a solution M and $\vec{\sigma}$ may exist for the previous equation, but $\vec{\sigma}$ yields no firable sequence of transitions and M cannot be reached.

For state machines, however, the following theorem holds.

Theorem 6.1. *Let $\langle N, M_0\rangle$ be a marked state machine.*

$$M \in R(N, M_0) \Longleftrightarrow (\exists \vec{\sigma} \in \mathbb{N}^m)\ [M = M_0 + C \cdot \vec{\sigma}].$$

Proof: Only $\Longleftarrow$ needs to be proved. At least one of the transitions given by $\vec{\sigma}$ may fire if $M_0 \neq M$. In fact, consider a place $p_i \ni M_0(p_i) > M(p_i) \geq 0$. (The existence of such a place is ensured because state machines are conservative.) Since

$$M(p_i) - M_0(p_i) = \sum_{j=1}^{m} c_{ij}\vec{\sigma}_j < 0,$$

there exists a transition t_j such that $\sigma(t_j) > 0$ and $c_{ij} = -1$, i.e., t_j outputs from p_i. Since $M_0(p_i) > 0$, t_j may fire reaching a marking M' such that: $M = M' + C \cdot \vec{\sigma}'$, where $\vec{\sigma}' < \vec{\sigma}$. Repeating the previous reasoning it is possible to conclude that eventually M will be reached. $\diamond$

In [Murata 89] is presented the same result in the case of *strongly connected* state machines.

Ichikawa and Hiraishi [Ichikawa 88b] have shown that for acyclic nets, i.e., nets whose underlying graph has no directed circuits, a vector $\vec{\sigma}$ solution of the state equation yields a firable sequence of transitions. Theorem 6.1 does not imply this stronger property.

Example 6.1. Consider the net in Figure 6.1, where $M_0 = (2\ 0\ 0\ 0\ 0)^T$ and $M = (1\ 0\ 1\ 0\ 0)^T$. The incidence matrix of the net is

$$C = \begin{bmatrix} -1 & 0 & -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$

The state equation is satisfied by: $\vec{\sigma} = (1\ 1\ 0\ 0\ 1\ 1)^T$. While $\vec{\sigma}$ does not yield a firable sequence of transitions, M can always be reached: just consider the vector $\vec{\sigma}' = (1\ 1\ 0\ 0\ 0\ 0)^T (< \vec{\sigma})$ that yields the firable sequence: $\sigma' = t_1 t_2$.

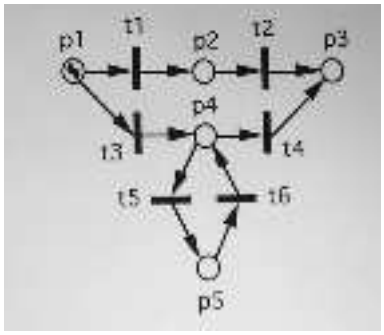


Figure 6.1: State machine in Example 6.1.

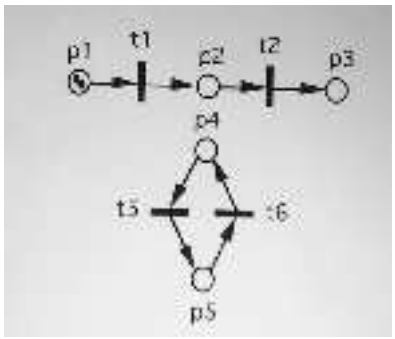


Figure 6.2: Firing subnet in Example 6.1.

The problem, as the previous example shows, is that the firing count vector could be counting, in addition to the sequence of transitions that needs to be fired to go from M_0 to M, a cycle that cannot be fired because it is not marked by a token. Hence the following two theorems hold.

Theorem 6.2. *Let $\langle N, M_0\rangle$ be a marked state machine and M be a marking of N. A vector $(\vec{\sigma} \in I\!N^m)\ [M = M_0 + C \cdot \vec{\sigma}]$ yields a firable sequence of transitions from M_0 if $(\nexists \vec{\sigma}' \in I\!N^m)\ [M = M_0 + C \cdot \vec{\sigma}',\ \vec{\sigma}' < \vec{\sigma}]$.*

Proof: If $\vec{\sigma}$ is the minimal vector satisfying the state equation for M and M_0, then the firing subnet given by $\vec{\sigma}$ is acyclic and all transitions may fire, as proved in [Ichikawa 88b]. ◇

Note that Theorem 6.2 gives a sufficient but not necessary condition. In Example 6.1, consider the vector $\vec{\sigma} = (0\ 0\ 1\ 1\ 1\ 1)^T$ solution of the state equation. $\vec{\sigma}$ is not minimal but yields a firable sequence $\sigma = t_3t_5t_6t_4$.

Theorem 6.3. *Let $\langle N, M_0\rangle$ be a marked state machine and M be a marking of N. A vector $(\vec{\sigma} \in I\!N^m)\ [M = M_0 + C \cdot \vec{\sigma}]$ yields a firable sequence of transitions from M_0 if and only if in the firing subnet given by $\vec{\sigma}$ each connected component is marked by M_0.*

Proof: The presence of a token in each connected component ensures that the cycles eventually present in $\vec{\sigma}$ can be executed. ◇

The firing subnet for Example 6.1 is in Figure 6.2; here there are two connected components $\{p_1, p_2, p_3\}$ and $\{p_4, p_5\}$, the second of which is not marked; hence its transitions cannot fire.

6.2.2 Defining the Set of Reachable Markings

Theorem 6.4. *Let $\langle N, M_0\rangle$ be a marked state machine. The set of reachable markings $R(N, M_0)$ is an integer linear convex set, i.e., can be defined as the integer solutions of a set of linear*

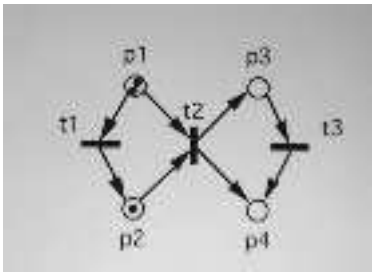


Figure 6.3: Net in Example 6.2, with a non convex reachability set.

inequalities.

Proof: Follows from Algorithm 6.1 presented below. ◇

This property does not hold in general for ordinary P/T nets.

Example 6.2. On the net in Figure 6.3, an ordinary conservative PN, $M = (0\ 1\ 0\ 2)$ can be reached from $M_0 = (2\ 1\ 0\ 0)$ but $M' = \frac{M_0+M}{2} = (1\ 1\ 0\ 1)$, which is a linear convex combination of M_0 and M, cannot be reached.

To determine the set of linear inequalities that the set of reachable markings of a state machine must satisfy, the following algorithm may be used. Note first that a connected (not necessarily strongly connected) state machine has a single P-semiflow whose support contains all the places.

Algorithm 6.1. *Let $\langle N, M_0\rangle$ be a marked state machine, and $\mathcal{T}^i = \{p^i_1, \ldots, p^i_k\}$ a trap of N. Define $M(\mathcal{T}^i) = M(p^i_1) + \ldots + M(p^i_k)$.*

1. *Consider all* basic *traps*[1] *of the net along with the support of the P-semiflow $\mathcal{T}^0$.*
2. *For each trap $\mathcal{T}^i$ write the inequality: $M(\mathcal{T}^i) \geq n_i$, where n_i is the number of tokens assigned to $\mathcal{T}^i$ by the initial marking. For $\mathcal{T}^0$ write the equation: $M(\mathcal{T}^0) = n_0$, where n_0 is the total number of tokens in the net.*
3. *If for a trap $\mathcal{T}^i$ $(i \neq 0)$ the number of tokens is $n_i = 0$, the inequality for $\mathcal{T}^i$ can be removed.*
4. *If $\mathcal{T}^i \subset \mathcal{T}^j$ $(j \neq 0)$ and $n_i = n_j$, the inequality for $\mathcal{T}^j$ can be removed, since it is implied by the inequality for $\mathcal{T}^i$.*
5. *The remaining set of inequalities plus the inequalities: $M \geq \vec{0}$, gives the set of markings reachable from the initial marking. These inequalities will be indicated as $\mathcal{A}(N)$.*

According to this Algorithm, the reachability set of a state machine $\langle N, M_0\rangle$ can be represent by the set $PR^A(N, M_0) = \{M \in I\!N^{|P|} \mid A \cdot M \geq A \cdot M_0\}$, where each row of the matrix A is the characteristic vector of the P-semiflow of the net or of one of the basic traps of the net.

Here is an example of application for the algorithm.

Example 6.3. Consider the net in Figure 6.4. Here the basic traps are:

$$\begin{aligned}
\mathcal{T}^0 &= \{p_1, p_2, p_3, p_4, p_5, p_6\},\\
\mathcal{T}^1 &= \{p_2\},\\
\mathcal{T}^2 &= \{p_5\},\\
\mathcal{T}^3 &= \{p_6\},\\
\mathcal{T}^4 &= \{p_1, p_2, p_5\},\\
\mathcal{T}^5 &= \{p_3, p_5, p_6\},\\
\mathcal{T}^6 &= \{p_3, p_4, p_5, p_6\}.
\end{aligned}$$

[1] A trap is basic if it is not the union of other traps.

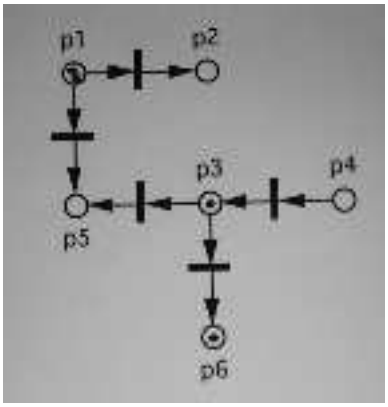


Figure 6.4: State machine in Example 6.3.

Note, e.g., that the trap $\{p_2, p_5\}$ is not considered, since it is given by the union of $\mathcal{T}^1$ and $\mathcal{T}^2$. The corresponding set of linear inequalities is:

$$
\begin{array}{ll}
(1) & M(p_1) + M(p_2) + M(p_3) + M(p_4) + M(p_5) + M(p_6) = 4, \\
(2) & M(p_2) \geq 0, \\
(3) & M(p_5) \geq 0, \\
(4) & M(p_6) \geq 1, \\
(5) & M(p_1) + M(p_2) + M(p_5) \geq 2, \\
(6) & M(p_3) + M(p_5) + M(p_6) \geq 2, \\
(7) & M(p_3) + M(p_4) + M(p_5) + M(p_6) \geq 2.
\end{array}
$$

Inequality (2) and (3) will be removed in step 3 of the algorithm, inequality (7) will be removed in step 4 as it is implied by (6). Finally the set of markings reachable from the initial marking is given by:

$$
\begin{array}{l}
M(p_1) + M(p_2) + M(p_3) + M(p_4) + M(p_5) + M(p_6) = 4, \\
M(p_6) \geq 1, \\
M(p_1) + M(p_2) + M(p_5) \geq 2, \\
M(p_3) + M(p_5) + M(p_6) \geq 2, \\
M \geq \vec{0}.
\end{array}
$$

Note 6.2. *Given a trap $\mathcal{T}$, the complement $P \setminus \mathcal{T}$ is a siphon, i.e., there is a siphon $\mathcal{S}^i = P \setminus \mathcal{T}^i$ corresponding to each trap $\mathcal{T}^i$, $(i \neq 0)$ considered in the Algorithm 6.1. Hence the inequality for $\mathcal{T}^i$ may be rewritten as $M(\mathcal{S}^i) \leq n_0 - n_i$.*

In fact siphons and traps are dual concepts and a dual algorithm, in terms of siphons, may be given for determining the linear inequalities satisfied by the reachable markings.

Note 6.3. *Let $\langle N, M_0 \rangle$ be a marked strongly connected state machine. Then the set of equations $\mathcal{A}(N)$ trivially reduces to:*

$$
\begin{array}{c}
M(\mathcal{T}^0) = n_0, \\
M \geq \vec{0}.
\end{array}
$$

Finally, it is necessary to discuss the correctness of the algorithm, i.e., the fact that the set of markings that satisfy the linear inequalities required by Algorithm 6.1 is exactly the reachability set. The idea is to show that a description of a state machine in term of traps captures the behavior of the net. This will be proven through the following propositions.

Proposition 6.1. *Let $\mathcal{T}$ be a trap of a state machine. The number of tokens in $\mathcal{T}$ is nondecreasing.*

Proof: On state machines, the firing of any transition preserves the total number of tokens. Also a transition has a single input and output arc. Hence no transition may output from a place in $\mathcal{T}$ to a place not in $\mathcal{T}$. ◇

Proposition 6.2. *Let $\mathcal{T}$ be a* minimal *trap*[2] *of a state machine. Then a token in $\mathcal{T}$ can move to any place in $\mathcal{T}$.*

Proof: For state machines, it is sufficient to prove that all places in $\mathcal{T}$ are strongly connected, i.e., there is a path from any place to all others. This is trivially true if $\mathcal{T}$ consist of a single place. Assume $\mathcal{T}$ consists of k places. Consider a place $p_1 \in \mathcal{T}$. There must be a transition from p_1 to some place $p_2 \in \mathcal{T}$, otherwise $\{p_1\}$ is a trap and $\mathcal{T}$ is not minimal. Consider $\{p_1, p_2\}$. With the same reasoning it is possible to infer that there must be a transition leading from $\{p_1, p_2\}$ to a place $p_3 \in \mathcal{T}$. This reasoning can be carried out until $\{p_1, p_2, ...p_k\} = \mathcal{T}$ is reached. Hence p_1 is connected to all places in $\mathcal{T}$. Since this reasoning can be applied to any place $p_i \in \mathcal{T}$, the proof is complete. ◇

Proposition 6.3. *Let $\mathcal{T}$ be a trap, and let $\mathcal{T}' = \{p \in \mathcal{T} \mid p \in \mathcal{T}'', \mathcal{T}'' \subset \mathcal{T}, \mathcal{T}''$ is a trap $\}$. Then a token in $\mathcal{T} \setminus \mathcal{T}'$ can move to any place in $\mathcal{T}$.*

Proof: Similar to the proof of Proposition 6.2. ◇

The soundness of the algorithm is then proved by the following reasoning. The marking of the traps is strictly nondecreasing (Proposition 6.1). The tokens initially present in a minimal trap may freely move to any place of the trap (Proposition 6.2). The tokens initially present in a trap that is not minimal are free to move to any place of the trap provided they also satisfy the constraints enforced by the traps contained in the non minimal trap (Proposition 6.3). These are the only constraints that the set of reachable markings must satisfy and these constraints are captured by Algorithm 6.1.

We point out that while in general the number of basic traps of Petri net may be exponential in the number of places, for state machines the number of basic traps is at most equal to the number of places. This may be proved by the following inductive reasoning. Consider a state machine with n places and no transitions; clearly there are n basic traps each one containing a single place. Now assume we have a state machine with its set of basic traps, and assume we add a new transition from place p to place p'. Then all the basic traps that do not contain p or that contain p' are still basic traps. All the basic traps that contain p and do not contain p' will not be traps in the new net and we need to add to each of them the minimal trap containing p' (there is only one such minimal trap) to obtain new basic traps. After the new traps are computed, it may well be the case that two of them are identical, that is the number of basic traps may only decrease when we add a transition.

6.3 Composition of State Machines Modules

The section discusses how the properties studied in Section 2 for state machines are preserved by concurrent composition. We will consider a restricted class of compositions.

Let us first introduce the following definitions.

Definition 6.1. *A* simple path *of a net N is a sequence $\theta = t_0 p_1 t_1 \dots p_r t_r$ of transitions and places such that*

$$(\forall i = 1, \dots, r)\ [{}^\bullet p_i = \{t_{i-1}\}, p_i^\bullet = \{t_i\}, t_{i-1}^\bullet = \{p_i\}, {}^\bullet t_i = \{p_i\}].$$

We assume that $(\forall i, j = 0, \dots, r \wedge i \neq j)[t_i \neq t_j]$. Note also that a single transition may be considered as a simple path with no places.

[2] A trap is minimal if it is not the superset of another trap.

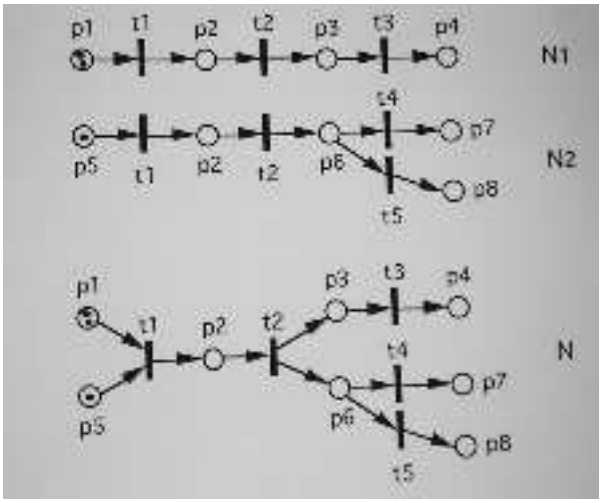


Figure 6.5: Two state machines composed along a simple path in Example 6.4.

Definition 6.2. *Given a net N, and k simple paths $\theta_i = t_{i,0} \ldots, t_{i,r_i}$ $(i = 1, \ldots, k)$, the k paths are looped in N if*

$$(\exists p \in P)\ (\forall i = 1, \ldots, k)\ [{}^\bullet t_{i,0} = t_{i,r_i}^\bullet = \{p\}, Pre(p, t_{i,0}) = Post(p, t_{i,r_i}) = 1].$$

In simple words, the k paths are looped if there exists a place p such that the initial (and final) transitions of all paths only input from (and only output to) place p with a single arc.

In this chapter, we will consider compositions of subnets that share: 1) a simple path; 2) a set of simple paths provided all paths are looped in one of the nets. We assume that the marking of the places along the composed paths are the same on all subnets.

For this class of compositions we will slightly change the definition of composition given in Appendix A.5. In fact here we are looking at composition based on the structure of the net rather that on the language generated by the net.

Example 6.4. The two nets in Figure 6.5 are composed along the simple path $\theta_1 = t_1 p_2 t_2$. The composed net is shown in Figure 1.b.

The two nets in Figure 6.6 are composed along the simple paths $\theta_1 = t_1 p_2 t_2$ and $\theta_2 = t_3 p_3 t_4$. Paths θ_1 and θ_2 are looped in N_2.

A net N obtained by composition of subnets N_1 and N_2 is denoted $N = N_1 \parallel N_2$.

We also recall the notation used for composed systems. Let N be a composed system $N = N_1 \parallel \ldots \parallel N_n$, and let M be a marking, $\vec{\sigma}$ be a firing count vector, and σ a firing sequence defined on it. The *projection of M on N_i*, denoted $M \uparrow_i$, is the vector obtained from M by removing all the components associated to places not present in N_i. The *projection of $\vec{\sigma}$ on N_i*, denoted $\vec{\sigma} \uparrow_i$, is the vector obtained by $\vec{\sigma}$ removing all the components associated to transitions not present in N_i. The *projection of σ on N_i*, denoted $\sigma \uparrow_i$, is the firing sequence obtained by σ removing all the transitions not present in N_i.

6.3.1 State Equation

Let $N = N_1 \parallel \ldots \parallel N_n$ be a composed net, and let $\vec{\sigma}$ be the firing count vector solution of $M = M_0 + C \cdot \vec{\sigma}$, where C is the incidence matrix of N. Assume that on each module N_i $(\forall i = 1, \ldots, n)$, the firing count vector $\vec{\sigma} \uparrow_i$ yields a firable sequence σ_i, i.e., $M_0 \uparrow_i \ [\sigma_i\rangle\ M \uparrow_i$

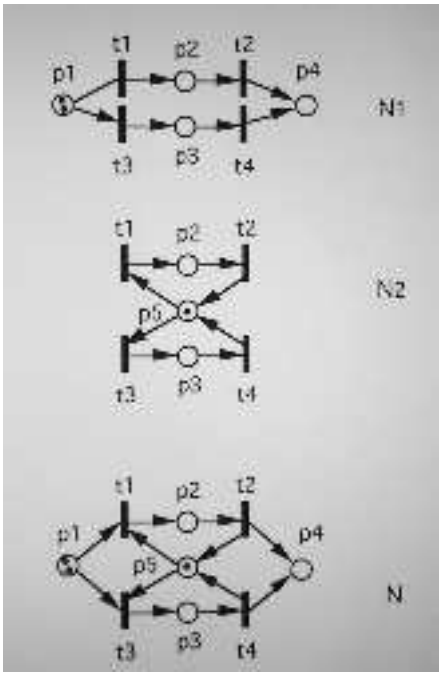


Figure 6.6: Two state machines composed along two simple paths in Example 6.4.

, $(i = 1, \ldots, n)$. This does not imply, however, that on the composed net $\vec{\sigma}$ yields a firable sequence σ such that $M_0 \, [\sigma\rangle \, M$.

There exist particular compositions, however, such that the reachability of a marking of the overall net may be determined simply by the analysis of the modules that compose it. These compositions, called elementary, are used to define the following class of P/T nets.

Definition 6.3. Elementary Composed State Machine *(ECSM) nets are the minimal class of P/T nets that is a superset of the class of state machines and is closed under the following compositions:*

1. *Composition of two nets sharing a single transition (or a simple path);*
2. *Composition of two nets through a set T_s of k transitions (or a set Θ of k simple paths) when the transitions (or simple paths) are looped in one of the nets.*

Although the two compositions we used to define ECSM may appear exceedingly simple, they permit the modular synthesis of realistic systems. As an example, the first kind of compositions may be used to construct the model of several systems connected through buffers or channels. The second kind of composition may be used to represent shared resources.

Theorem 6.5. *Let $\langle N, M_0 \rangle$ be a marked net where $N = N_1 \parallel N_2$ and assume that N_1 and N_2 have been composed by means of one of the compositions used to define the class of ECSM. A firing count vector for N yields a firable sequence if and only if on each module N_i the projection of the firing count vector yields a firable sequence $\vec{\sigma}_i$.*

Proof: The "only if" part is trivial. Let us prove the "if" part for single transitions (i.e., not simple path) composition. In the case of simple path compositions the proof is substantially the same (see also [Giua 90b]).

- *Composition of two nets sharing one transition t.*
  Consider a solution $\vec{\sigma}$ of the state equation $M = M_0 + C \cdot \vec{\sigma}$. By hypothesis on N_1 and

N_2, $\vec{\sigma} \uparrow_i$ yields a firable sequence σ_i such that $M_0 \uparrow_i \ [\sigma_i\rangle \ M \uparrow_i, \quad (i = 1, 2)$. Let the component of $\vec{\sigma}$ associated to the transition t have the value k, i.e., there are k occurrences of t in each σ_i. Then σ_i may be written as

$$\sigma_i = x_i^0 \ t \ x_i^1 t \ \ldots x_i^{k-1} \ t \ x_i^k, \quad (i = 1, 2).$$

Now let

$$\sigma = x_1^0 \ x_2^0 \ t \ x_1^1 \ x_2^1 \ t \ \ldots \ x_2^{k-1} \ t \ x_1^k \ x_2^k.$$

Clearly σ is a firable sequence given by the firing count vector $\vec{\sigma}$.

- *Composition of two net sharing a set of transitions T_s looped in one of the nets.*
  Consider a solution $\vec{\sigma}$ of the state equation $M = M_0 + C \cdot \vec{\sigma}$. By hypothesis on N_1 and N_2, $\vec{\sigma} \uparrow_i$ yields a firable sequence σ_i such that $M_0 \uparrow_i \ [\sigma_i\rangle \ M \uparrow_i, \quad (i = 1, 2)$. Let t_i^j be the j-th occurrence in σ_i of a transition in T_s. Then σ_i may be written as

  $$\sigma_i = x_i^0 \ t_i^1 \ x_i^1 t_i^2 \ \ldots x_i^{k-1} \ t_i^k \ x_i^k, \quad (i = 1, 2).$$

  Let N_2 be the looped net. Then any time a transition $t \in T_s$ fires any other transition in T_s may fire. Hence the following is a firable sequence for N_2 as well

  $$\sigma_2' = x_2^0 \ t_1^1 \ x_2^1 t_1^2 \ \ldots x_2^{k-1} \ t_1^k \ x_2^k.$$

  Now let

  $$\sigma = x_1^0 \ x_2^0 \ t_1^1 \ x_1^1 \ x_2^1 \ t_1^2 \ \ldots \ x_2^{k-1} \ t_1^k \ x_1^k \ x_2^k.$$

  Clearly σ is a firable sequence given by the firing count vector $\vec{\sigma}$. ◇

Corollary 6.1. *Let $\langle N, M_0\rangle$ be a marked ECSM net where $N = N_1 \parallel \cdots \parallel N_n$ and $N_i, \ (i = 1, \ldots n)$, is a state machine. A firing count vector for N yields a firable sequence if and only if on each module N_i the projection of the firing count vector yields a firable sequence.*

Proof: By recursive application of Theorem 6.5. ◇

Theorem 6.2 and Theorem 6.3 may be restated in this form for ECSM nets.

Theorem 6.6. *Let $\langle N, M_0\rangle$ be a marked ECSM net and M be a marking of N. $N = N_1 \parallel \ldots \parallel N_n$ and $N_i, \ (i = 1, \ldots, n)$, is a state machine.*

A vector $(\vec{\sigma} \in I\!N^m) \ [M = M_0 + C \cdot \vec{\sigma}]$ yields a firable sequence of transitions if

$$(\forall i = 1, \ldots, n)(\nexists \vec{\sigma}_i')\ [M \uparrow_i = M_0 \uparrow_i + C_i \cdot \vec{\sigma}_i', \ \vec{\sigma}_i' < \vec{\sigma} \uparrow_i].$$

Proof: Under these hypotheses, on each module $\vec{\sigma} \uparrow_i$ yields a firable sequence (by Theorem 6.2). Hence by Corollary 6.1 $\vec{\sigma}$ yields a firable sequence for N. ◇

Theorem 6.7. *Let $\langle N, M_0\rangle$ be a marked ECSM net and M be a marking of N. $N = N_1 \parallel \ldots \parallel N_n$ and $N_i, \ (i = 1, \ldots, n)$, is a state machine.*

A vector $(\vec{\sigma} \in I\!N^m) \ [M = M_0 + C \cdot \vec{\sigma}]$ yields a firable sequence of transitions if and only if in the firing subnet given by $\vec{\sigma} \uparrow_i \ (i = 1, \ldots, n)$ each connected component is marked by M_0.

Proof: On module N_i, the firing count vector $\vec{\sigma} \uparrow_i$ yields a firable sequence if and only if in the firing subnet all connected components are marked (by Theorem 6.3). Hence by Corollary 6.1 $\vec{\sigma}$ yields a firable sequence for N if and only if in the firing subnet for each module N_i all connected components are marked. ◇

6.3.2 Defining the Set of Reachable Markings

This subsection will show how it is possible to derive the set of linear inequalities that defines the set of reachable markings in ECSM nets. Firstly, the case of two state machines, composed through a single transition or a set of looped transitions, is discussed. Then these results will be extended to the composition of ECSM through simple paths.

Definition 6.4. *Let $\langle N, M_0\rangle$ be a marked state machine where M_0 assigns all the tokens to a place p_0, and let M_p be a marking that assigns a single token to place p and no token elsewhere. Given a transition t, it is possible to define the following two sets of places on the net:*

$$P_t = \{p \in P \mid (\exists\sigma)\ [\sigma(t) > 0,\ M_{p_0}\ [\sigma\rangle\ M_p]\},$$

$$P_{\overline{t}} = \{p \in P \mid (\exists\sigma)\ [\sigma(t) = 0,\ M_{p_0}\ [\sigma\rangle\ M_p]\}.$$

In simple words, the set P_t contains the places that may be marked by firing t at least once by an initial marking that assigns a token to the initial place p_0 and no tokens to the other places. The set $P_{\overline{t}}$ contains the places that may be marked without firing t by an initial marking that assigns a token to the initial place p_0 and no tokens to the other places.

Similarly, given a set of transitions T_s, it is possible to define the following two sets of places on the net:

$$P_{T_s} = \{p \in P \mid (\exists\sigma)\ (\exists t \in T_s)\ [\sigma(t) > 0,\ M_{p_0}\ [\sigma\rangle\ M_p]\},$$

$$P_{\overline{T}_s} = \{p \in P \mid (\exists\sigma)\ (\forall t \in T_s)\ [\sigma(t) = 0,\ M_{p_0}\ [\sigma\rangle\ M_p]\}.$$

Proposition 6.4. (Firing Bounds) *Let $\langle N, M_0\rangle$ be a marked state machine where M_0 assigns all the tokens to a place p_0. Now given a marking $M \in R(N, M_0)$, the number of times t has fired to reach M may vary and can assume any integer value in the range $[\rho_{\min}(M,t), \rho_{\max}(M,t)]$, where*

$$\rho_{\min}(M,t) = \sum_{p \in P_t \setminus P_{\overline{t}}} M(p);$$

$$\rho_{\max}(M,t) = \begin{cases} \sum_{p\in P_t} M(p) & \text{if there is no cycle containing } t, \\ 0 & \text{if there is a cycle containing } t \\ & \text{and } \sum_{p\in P_t} M(p) = 0, \\ \infty & \text{if there is a cycle containing } t \\ & \text{and } \sum_{p\in P_t} M(p) > 0. \end{cases}$$

$\rho_{\min}(M,t)$ and $\rho_{\max}(M,t)$ are called the firing bounds *of t for marking M.*

Proof: For each token in $P_t \setminus P_{\overline{t}}$ transition t must have fired at least once, while for each token in P_t transition t may have fired once if there is no cycle containing t or an arbitrary large number of times if there is a cycle containing t. ◇

Note, also, that

$$\rho_{\min}(M,t) \leq \sum_{p\in P} M_0(p),$$

and that when there is a cycle containing t a good approximate bound — that will be used in the following — for $\rho_{\max}(M,t)$ is

$$\rho'_{\max}(M,t) = h \sum_{p\in P_t} M(p) \leq \rho_{\max}(M,t),$$

where h is any integer large enough.

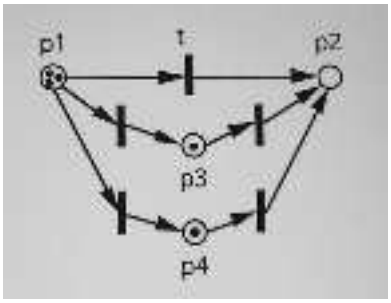


Figure 6.7: State machine in Example 6.4.

Proposition 6.4 is restricted to the case of an initial marking that assigns all tokens to a single place. This requirement is fair, in the sense that in the application cases of interest here, such as manufacturing systems, etc., the presence of multiple tokens in a state machine module indicates a multiplicity of identical resources, such as buffer spaces or identical machines. Hence the multiple tokens initially should be assigned to the same place. The following example will clarify the necessity for this restriction.

Example 6.5. In the net N_1 in Figure 6.7, $\rho_{\max}$ cannot be expressed as a simple linear function of a marking M but may be defined as

$$\rho_{\max}(M,t) = \min\{ \quad M(p_2),\ M(p_2)+M(p_3)-1, \\ M(p_2)+M(p_4)-1,\ M(p_2)+M(p_3)+M(p_4)-2\}.$$

The problem here is that the sets P_t and $P_{\bar{t}}$ are different for the different places marked by the initial marking. However, whenever these sets are identical for all the places marked by the initial marking, even if they are more than one, the results of Proposition 6.4 are valid. We will assume in the following, unless explicitly stated otherwise, that this initial marking condition is verified on all state machine modules considered.

When two nets N_i $(i = 1, 2)$ are composed through a single transition t, the marking of the overall net will be a subset of the cartesian product of the markings of the two modules. Given a reachable marking M_i on the net N_i, the marking $M = [M_1^T M_2^T]^T$ will be reachable on the composed net if and only if there is a sequence σ_i reaching M_i on the net N_i $(i = 1, 2)$, and $\sigma_1(t) = \sigma_2(t)$. If N_1 and N_2 are state machines, this requires that

$$[\rho^1_{\min}(M_1,t), \rho^1_{\max}(M_1,t)] \bigcap [\rho^2_{\min}(M_2,t), \rho^2_{\max}(M_2,t)] \neq \emptyset.$$

where the exponent i in $\rho^i_{\min}(M_i,t)$ and $\rho^i_{\max}(M_i,t)$ denotes that the firing bound is computed on the net N_i. Clearly the two intervals will not be disjoint if and only if: $\rho^1_{\min}(M_1,t) \leq \rho^2_{\max}(M_2,t)$, and $\rho^2_{\min}(M_2,t) \leq \rho^1_{\max}(M_1,t)$.

Theorem 6.8. *When two state machines N_1 and N_2 are composed through a single transition t the set of markings reachable from the initial marking M_0 for the composed net $N = N_1 \parallel N_2$ is given by the following set of linear inequalities $\mathcal{A}(N)$:*

$$\mathcal{A}(N_1),$$

$$\mathcal{A}(N_2),$$

$$\sum_{p \in P^1_t \setminus P^1_{\bar{t}}} M(p) \leq h_2 \sum_{p \in P^2_t} M(p),$$

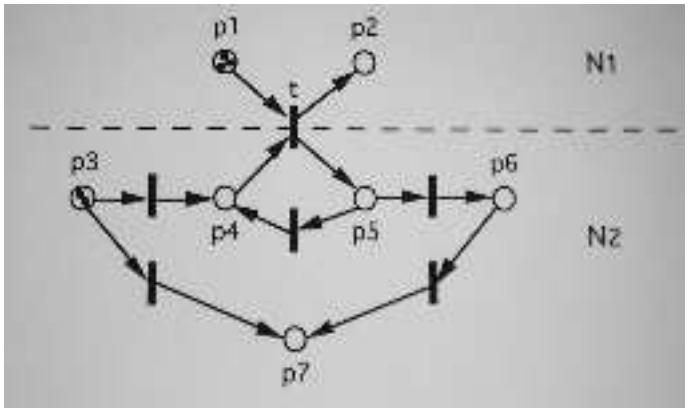


Figure 6.8: ECSM net in Example 6.6.

$$\sum_{p \in P_t^2 \setminus P_{\bar{t}}^2} M(p) \leq h_1 \sum_{p \in P_t^1} M(p),$$

where:

- $\mathcal{A}(N_i)$ $(i = 1, 2)$*, is the set of inequalities for the net* N_i *(as derived with Algorithm 6.1);*
- *the set of places* P_t^i *and* $P_{\bar{t}}^i$ *belongs to* N_i $(i = 1, 2)$*, and are determined as in Definition 6.4;*
- $h_1 = 1$ $(h_2 = 1)$*, if there is no cycle containing* t *in* N_1 (N_2)*, else* h_1 (h_2) *is equal to the number of tokens contained in net* N_2 (N_1)*. As noted before, we are using an approximated linear bound for* $\rho^i_{\max}(M_i, t)$*.*

Proof: Follows from Proposition 6.4 and the fact that if the last two inequalities are satisfied there exists a firable sequence σ_i for both modules that reaches $M \uparrow_i$ firing t an equal number of times. Hence the vector

$$(\vec{\sigma} \in I\!N^m)\ [\vec{\sigma} \uparrow_1 = \vec{\sigma}_1,\ \vec{\sigma} \uparrow_2 = \vec{\sigma}_2]$$

is a firing count vector for N and by Theorem 6.5 there exists a firable sequence for the overall net. ◇

Example 6.6. Consider the composed system of Figure 6.8. The set of places of interest are:

$$P_t^1 = \{p_2\},$$

$$P_{\bar{t}}^1 = \{p_1\},$$

$$P_t^2 = \{p_4, p_5, p_6, p_7\},$$

$$P_{\bar{t}}^2 = \{p_3, p_4, p_7\}.$$

Since there is no cycle containing t in N_1, then $h_1 = 1$, while, since there is a cycle containing t in N_2, $h_2 = 3$. The linear inequalities that define the space of reachable markings on N_1 is:

$$M(p_1) + M(p_2) = 3,$$

$$M(p_1), M(p_2) \geq 0,$$

and on N_2:

$$M(p_3) + M(p_4) + M(p_5) + M(p_6) + M(p_7) = 2,$$

$$M(p_3), M(p_4), M(p_5), M(p_6), M(p_7) \geq 0.$$

Hence the set of reachable markings on the composed system is defined by:

$$M(p_1) + M(p_2) = 3,$$

$$M(p_3) + M(p_4) + M(p_5) + M(p_6) + M(p_7) = 2,$$
$$M(p_2) \leq 3(M(p_4) + M(p_5) + M(p_6) + M(p_7)),$$
$$M(p_5) + M(p_6) \leq M(p_2),$$
$$M \geq \vec{0}.$$

Note 6.4. *The inequalities derived in Theorem 6.8 may not always be necessary. Suppose that on one of the modules, say N_1, one of the following conditions holds:*

1. *$P_t^1 \setminus P_{\bar{t}}^1 = \emptyset$. Hence*
$$0 = \sum_{p \in P_t^1 \setminus P_{\bar{t}}^1} M(p) \leq h_2 \sum_{p \in P_t^2} M(p)$$
is always verified.

2. *$P_t^1 = P_1$. Here P_1 is the set of places of N_1. In this case there is a cycle containing t and p_0 (the place marked by M_0), and t may fire infinitely often in N_1 for each reachable marking. Hence*
$$\sum_{p \in P_t^2 \setminus P_{\bar{t}}^2} M(p) \leq h_1 \sum_{p \in P_t^1} M(p)$$
is always verified.

In the following example we will show an example of a system in which the initial marking marks more than one place.

Example 6.7. In the net N_1 in Figure 6.7, discussed in Example 6.5, we have

$$\begin{aligned}\rho^1_{\max}(M,t) = \min\{ \quad & M(p_2),\ M(p_2) + M(p_3) - 1, \\ & M(p_2) + M(p_4) - 1,\ M(p_2) + M(p_3) + M(p_4) - 2\},\end{aligned}$$
$$\rho^1_{\min}(M,t) = 0.$$

Hence, when composing the module through transition t with another module N_2 the following inequalities should be considered

$$\mathcal{A}(N_1),$$
$$\mathcal{A}(N_2),$$
$$\rho^2_{\min}(M,t) \leq M(p_2),$$
$$\rho^2_{\min}(M,t) \leq M(p_2) + M(p_3) - 1,$$
$$\rho^2_{\min}(M,t) \leq M(p_2) + M(p_4) - 1,$$
$$\rho^2_{\min}(M,t) \leq M(p_2) + M(p_3) + M(p_4) - 2.$$

The case of a composition of two state machines through a set of transitions T_s is now considered.

Theorem 6.9. *When two state machines N_1 and N_2 are composed through a set T_s of transitions and these transitions are looped in one of the nets, say N_2, the set of markings reachable from the initial marking M_0 for the composed net $N = N_1 \parallel N_2$ is given by the following set of linear inequalities $\mathcal{A}(N)$:*

$$\begin{gathered}\mathcal{A}(N_1),\\ \mathcal{A}(N_2),\\ \sum_{p\in P^1_{T_s}\setminus P^1_{\overline{T}_s}} M(p) \le h \sum_{p\in P^2_{T_s}} M(p),\end{gathered}$$

where:

- *$\mathcal{A}(N_i)$ $(i = 1, 2)$, is the set of inequalities for the net N_i (as derived with Algorithm 6.1);*
- *the sets of places $P^i_{T_s}$ and $P^i_{\overline{T}_s}$ $(i = 1, 2)$, belongs to N_i, and are determined as in Definition 6.4;*
- *h is equal to the number of tokens contained in N_1.*

Proof: Given the special structure of N_2, any reachable marking M_2 of N_2 may be reached without firing any transition in T_s. Hence it is never possible, as suggested by the previous note, that a firing sequence on N_2 may require the firing of more transitions in T_s than a firing sequence on N_1. Also, if a marking M'_2 of N_2 may be reached by firing a transition in T_s, then any sequence of transitions in T_s may also be fired. Hence the only constraint imposed by the composition of the two modules is that for any marking $M = [M_1^T M_2^T]^T$, if reaching M_1 requires the firing of one or more transition in T_s, M_2 may be also be reached by firing a transition in T_s. This constraint is expressed by the third inequality of $\mathcal{A}(N)$. ◇

The following two theorems generalize Theorem 6.8 and Theorem 6.9 to the composition of two ECSM (not only state machines) along simple paths (not only single transitions). These theorems will be given without proof.

When two ECSM nets are composed along a simple path it is necessary to consider the possibility that the path may belong to more than one state machine module on each net. Also the places determined in Definition 6.4 are computed with respect to the first transition of the path.

Theorem 6.10. *Let N_1 and N_2 be two ECSM nets, i.e., $N_i = N_{i,1} \parallel \ldots \parallel N_{i,n_i}$ $(i = 1, 2)$, where $N_{i,j}$ is a state machine. Assume N_1 and N_2 are to be composed through a simple path of transitions $\theta = t_0 p_1 t_1 \ldots p_r t_r$ that belongs to modules $N_{1,q}$ $(q \in J_1)$ and to modules $N_{2,s}$ $(s \le J_2)$. The set of markings reachable from the initial marking M_0 for the composed net $N = N_1 \parallel N_2$ is given by the following set of linear inequalities $\mathcal{A}(N)$:*

$$\begin{gathered}\mathcal{A}(N_1),\\ \mathcal{A}(N_2),\end{gathered}$$

$$\sum_{p\in P^{1,q}_{t_0}\setminus P^{1,q}_{\bar{t}_0}} M(p) \le h_2^{q,s} \sum_{p\in P^{2,s}_{t_0}} M(p), \qquad (q \in J_1,\ s \in J_2),$$

$$\sum_{p\in P^{2,s}_{t_0}\setminus P^{2,s}_{\bar{t}_0}} M(p) \le h_1^{q,s} \sum_{p\in P^{1,q}_{t_0}} M(p), \qquad (q \in J_1,\ s \in J_2),$$

where:

- *$\mathcal{A}(N_i)$ is the set of inequalities for the net N_i;*

- *the sets of places $P_t^{i,j}$ and $P_{\bar{t}}^{i,j}$ $(i = 1, 2;\ j \in J_i)$,belongs to $N_{i,j}$, and are determined as in Definition 6.4;*

- *$h_1^{q,s} = 1$ $(h_2^{q,s} = 1)$ if there does not exist a cycle containing the path θ in the net $N_{1,q}$ $(N_{2,s})$. Else $h_1^{q,s}$ $(h_2^{q,s})$ is equal to the number of tokens contained in the net $N_{2,s}$ $(N_{1,q})$.*

When two ECSM nets are composed along k simple paths, the paths are looped in one of the nets, hence they belong to only one state machine module of the looped net. However, it is necessary to consider the possibility that each path may belong to more than one state machine module on the net that is not looped.

Theorem 6.11. *Let N_1 and N_2 be two ECSM nets, i.e., $N_i = N_{i,1} \parallel \ldots \parallel N_{i,n_i}$ $(i = 1, 2)$, where $N_{i,j}$ is a state machine. Assume N_1 and N_2 are to be composed through a k simple path of transitions $\theta_j = t_0^j p_1^j t_1^j \ldots t_{r_j}^j$ $(j = 1, \ldots, k)$, and let $T_s = \{t_0^1, \ldots, t_0^k\}$. Assume furthermore that path θ_j belongs to modules $N_{1,q}$ $(q \in J_j)$ and that all paths are looped in the module $N_{2,1}$. The set of markings reachable from the initial marking M_0 for the composed net $N = N_1 \parallel N_2$ is given by the following set of linear inequalities $\mathcal{A}(N)$:*

$$\begin{gathered}\mathcal{A}(N_1),\\ \mathcal{A}(N_2),\end{gathered}$$

$$\sum_{p \in P_1'} M(p) \leq h \sum_{p \in P_2'} M(p),$$

where:

- *$\mathcal{A}(N_i)$ is the set of inequalities for the net N_i;*

- *P_1' is the set of places in N_1 that may be marked only by firing a transition $t \in T_s$:*

$$P_1' = \bigcup_{j=1}^{k} \bigcup_{q \in J_j} \left(P_{t_0^j}^{1,q} \setminus P_{\bar{t}_0^j}^{1,q} \right),$$

  while P_2' is the set of places in N_2 that may be marked firing a transition $t \in T_s$:

$$P_2' = P_{T_s}^{2,1};$$

- *h is equal to the sum of the tokens contained in the nets $N_{1,q}$ $(q \in J_j)$ $(j = 1, \ldots, k)$.*

We conclude this section pointing out that when state machines modules are composed to form an ECSM, the number of equations of that defines the reachability set grows, in the worst case, more than linearly. In the case of Theorem 6.10 we have to add $2 \times |J_1| \times |J_2|$ equations.

6.4 Supervisory Verification

In this section the results developed so far are applied to the validation of supervisors for the control of discrete event systems (DES).

Consider m discrete event systems, represented by the state machines $N_1, \ldots, N_m$, working concurrently. It is generally assumed that the set of transitions of all these systems are disjoint. The specifications to be enforced on the joint behavior of these systems are represented by n different state machines $H_1, \ldots, H_n$, whose transitions are a subset of all the transitions of the N's. The procedure for determining a *monolithic supervisor* requires the construction, by concurrent

composition, of the net $E = N_1 \parallel \ldots \parallel N_m \parallel H_1 \parallel \ldots \parallel H_n$. Recall that there exists a set of final markings associated to the N's and H's.

E and the nets N_i $(i = 1, \ldots, n)$ will run in parallel, i.e., whenever a transition fires on one of the nets N_i, it is also fired in E. Furthermore, the transitions enabled in E at a given marking represent the transitions that are allowed to fire in the nets N_i, while all other transitions are disabled. The net E will represent a proper supervisor, if the following two properties are ensured.

- *Nonblockingness:* the reachability set of the net E does not contain blocking markings, i.e., markings from which a final marking cannot not be reached;
- *Controllability:* it is not possible to reach a marking from which an uncontrollable transition, belonging to the net N_i, is enabled in N_i but is not enabled in E.

In the following subsection it is discussed how these properties may be verified by Integer Programming techniques in the case that E is an ECSM net. Clearly these restrictions heavily limit the class of control problems that can be solved by our approach. However, it is possible to check for these properties in more efficient ways than by brute force state space search. Additionally, as will be shown below, it may be the case that the model may be validated by simple Linear Programming.

6.4.1 Blocking

Let $\langle N, M_0 \rangle$ be a marked net, and M_f be a final marking associated to it. For the sake of simplicity assume that M_f is the only final marking of the net. The net N will be blocking if and only if

$$(\exists M)\ [M \in R(N, M_0), M_f \notin R(N, M)]$$

Now the set $R(N, M_0)$ of an ECSM net can be given as a convex linear set, as shown in Section 3. Similarly the *coreachability set* of M_f, i.e., the set $\{M \mid M_f \in R(N, M)\}$, can be given in this form, according to the following proposition.

Proposition 6.5. *Let $N = (P, T, I, O)$ be a net. Given a marking M_f of N, the coreachability set of M_f is identical to the reachability set from M_f in the reversed net $N^R = (P, T, O, I)$, that is,*

$$\{M \mid M_f \in R(N, M)\} = R(N^R, M_f).$$

Finally it is possible to check for the existence of blocking markings as follows.

Proposition 6.6. *Let $\langle N, M_0 \rangle$ be a marked ECSM net and let M_f be the final marking associated with it. Assume the reachability set of M_0 is given by the set $PR^{A_0}(N, M_0) = \{M \in I\!N^{|P|} \mid A_0 \cdot M \geq A_0 \cdot M_0\}$, and the coreachability set of M_f is given by the set $PR^{A_f}(N, M_f) = \{M \in I\!N^{|P|} \mid A_f \cdot M \geq A_f \cdot M_f\}$, where*

$$A_f = \begin{bmatrix} \vec{a}_f^{\,1} \\ \ldots \\ \vec{a}_f^{\,k} \end{bmatrix}.$$

Then there exist a blocking marking M if and only if one or more of the following Constraint Sets has an integer feasible solution $(\forall i = 1, \ldots, k)$:

$$\begin{array}{c} A_0 \cdot M \geq A_0 \cdot M_0, \\ \vec{a}_f^{\,i} \cdot M \leq \vec{a}_f^{\,i} \cdot M_f - 1, \\ M \geq \vec{0}. \end{array} \qquad \mathrm{CS1_i}$$

Proof: M is a blocking marking $\iff M \in R(N, M_0) \wedge M_f \notin R(N, M) \iff [A_0 \cdot M \geq A_0 \cdot M_0] \wedge \neg[A_f \cdot M \geq A_f \cdot M_f] \iff [A_0 \cdot M \geq A_0 \cdot M_0] \wedge [\bigvee_{i=1}^{k} \vec{a}_f^{\,i} \cdot M < \vec{a}_f^{\,i} \cdot M_f] \iff (\exists i)[(A_0 \cdot M \geq A_0 \cdot M_0) \wedge (\vec{a}_f^{\,i} \cdot \vec{a}_f^{\,i} \cdot M_f - 1)]$. $\diamond$

Note that each constraint $\vec{a}_f^i \cdot M < \vec{a}_f^i \cdot M_f$ has been rewritten in the equivalent form $\vec{a}_f^i \cdot M \leq \vec{a}_f^i \cdot M_f - 1$.

It is possible to relax the constraint that the vector M be integer and obtain a sufficient condition for the validation of the net. In fact, if no real vector M satisfies the previous systems of inequalities no blocking marking may be reached.

Note also that some of the constraints $\vec{a}_f^i \cdot M \geq \vec{a}_f^i \cdot M_f$ may be trivially verified by any marking in $R(N, M_0)$. This is the case, for instance, for the constraints coming from the P-semiflows of the net that are satisfied by the reachable and coreachable markings. Thus, we need not solve the corresponding $CS1_i$, as it clearly has no feasible solution.

6.4.2 Controllability

Consider the net E constructed by concurrent composition of all the systems and specification modules. Let $t_q \in T_u$ be an uncontrollable transition and assume that t_q belongs to the system net N_i. (By the hypothesis that all the systems have disjoint transitions, a transition may belong to only one net N, although it may belong to more than one specification net H.) Let the preset of t_q be: ${}^\bullet t_q = \{p_q^0, \ldots, p_q^{k_q}\}$, where p_q^0 is a place of N_i and p_q^j $(j > 0)$ is a place of some specification net H.

E is not controllable if and only if it is possible to reach a marking M such that an uncontrollable transition t_q is enabled by $M \uparrow_i$ in N_i, but it is not enabled by M in E. In other words, E is not controllable if and only if

$$(\exists t_q \in T_u)\ (\exists M)\ [M \in R(E, M_0) \wedge M(p_q^0) \geq 1 \wedge (\bigvee_{j=1}^{k_q} M(p_q^j) \leq 0)].$$

Proposition 6.7. *Let the reachability set of E be given by the set $PR^A(E, M_0) = \{M \in \mathbb{N}^{|P|} \mid A \cdot M \geq A \cdot M_0\}$. Then E will not be controllable if and only if one or more of following Constraint Sets has an integer feasible solution ($\forall t_q \in T_u$ and $\forall p_q^j \in {}^\bullet t_q\ j > 0$)*

$$\begin{array}{cc} A \cdot M \geq A \cdot M_0, & \mathrm{CS2_{q,j}} \\ M(p_q^0) \geq 1, & \\ M(p_q^j) \leq 0, & \\ M \geq \vec{0}. & \end{array}$$

Proof: E is not controllable $\Longleftrightarrow$
$(\exists t_q \in T_u)\ (\exists M)\ [M \in R(E, M_0) \wedge M(p_q^0) \geq 1 \wedge (\bigvee_{j=1}^{k_q} M(p_q^j) \leq 0)] \Longleftrightarrow$
$(\exists t_q \in T_u)\ (\exists M)\ (\exists j)\ [A \cdot M \geq A \cdot M_0 \wedge M(p_q^0) \geq 1 \wedge M(p_q^j) \leq 0]$. $\diamond$

Also if the constraint that M be integer is relaxed, we may use Linear Programming to derive a sufficient condition for E to be controllable.

6.5 Conclusions

In this chapter we have defined a class of P/T net, called Elementary Composed State Machine nets. The reachability problem for this class can be solved by incidence matrix analysis. We have shown how it is possible to derive a set of linear inequalities that exactly define the set of reachable markings. Important properties of the net, such as the absence of blocking states or controllability, may be studied by Integer Programming techniques. This approach may be used to validate supervisors for the control of discrete event systems.

The main drawbacks of this approach is summarized as follows:

- Integer Programming problems may be not be solved, in general, in polynomial time. However we have also shown that it is possible to use Linear Programming techniques to derive sufficient conditions for supervisory validation.
- The model is limited, in the sense that there exist monolithic supervisors that cannot be modeled as ECSM nets.
- Although a procedure to validate a supervisor is given here, the control problem is not directly solved, in the sense that if the model does not have the desired properties the approach does not directly lead to the construction of a proper supervisor.

Chapter 7

GENERALIZED MUTUAL EXCLUSION CONSTRAINTS

7.1 Introduction

Mutual exclusion constraints are a natural way of expressing the concurrent use of a finite number of resources, shared among different processes. In the framework of Petri nets and from a very general perspective, we define a generalized mutual exclusion constraint (GMEC) as a condition that limits a weighted sum of tokens contained in a subset of places [Giua 92b]. Let $\langle N, M_0 \rangle$ be a net system with set of places P. A constraint $(\vec{w}, k)$ defines a set of *legal* markings:

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

where $\vec{w}$ is a weight vector of nonnegative integers, and k is a positive integer. Markings in $\mathbb{N}^{|P|}$ that are not legal will be denoted *forbidden* markings.

In the first part of this chapter we present a methodology, based on linear algebraic techniques [Silva 92], to compare and simplify GMEC. An equivalence notion among GMEC is introduced and studied from the point of view of structural net theory.

In traditional Petri net modeling all transitions are assumed to be *controllable*, i.e., may be prevented from firing by a control agent. A problem addressed for those systems with shared resources has been that of deadlock prevention or avoidance [Banaszak 90, Viswanadham 90, Zhou 91]. A single GMEC may be easily implemented by a *monitor*, i.e., a place whose initial marking represents the available units of a resource and whose outgoing and incoming transitions represent, respectively, the acquisition and release of units of the resource.

In the framework of Supervisory Control [Golaszewski 88], the complexity of enforcing a GMEC is enhanced by the presence of *uncontrollable* transitions, i.e., transitions that may be observed but not prevented from firing by a control agent. To enforce a given GMEC, it is necessary to prevent the system from reaching a superset of the forbidden markings, containing all those markings from which a forbidden one may be reached by firing a sequence of uncontrollable transitions. Unfortunately, in this case we prove that the set of legal markings cannot *always* be represented by a linear domain in the marking space, thus there exist problems which do not have a "monitor-based" solution.

In this context, the second part of this chapter discusses GMEC for systems represented as marked graphs. The goal is that of constructing a supervisor capable of enforcing the constraints. A solution to this problem has been given by Holloway and Krogh [Holloway 90, Holloway 92a, Krogh 91]. In their approach, which may be defined as *fully interpreted*, the control policy is efficiently computed by an on-line controller as a feedback function of the marking of the system. In this work, instead, we are interested in deriving a net based structure for the supervisor. We

will discuss and compare several solutions. Some of these models will be *fully compiled*, i.e., the corresponding supervisor is represented by a P/T net, others will be *partially compiled*, i.e., the corresponding supervisor is given as an interpreted net in which the firing of some transitions not only depends on the marking of the net but on the value of some boolean expressions as well.

The advantages of fully compiling the supervisor action in a net structure are:

- The computation of the control action is faster, since it does not require separate on-line computation.
- The same Petri net system execution algorithms may be used for both the original system and the supervisor.
- A closed-loop model of the system under control may be built with standard net composition constructions, and analyzed for properties of interest. Moreover, the structural theory of Petri nets may be used to prove, without an exhaustive state space search, that the system under control enjoys the properties.

On the other hand, partially compiled models are more flexible in the sense that some interpretations may be used to implement complex control policies, whose corresponding net structure may be exceedingly large.

The chapter is structured as follows. In Section 7.2, generalized mutual exclusions constraints are defined. If all transitions of the net are controllable, these constraints may be enforced by a monitor place. If some of the transitions are uncontrollable, it is possible to prove that a monitor-based solution may not exist. In Section 7.3, we study the properties of marked graphs with the addition of monitor places. In particular, we discuss a sub-class of marked graphs for which there is always a monitor-based solution, corresponding to a set of GMEC, even when some transitions are uncontrollable. In Section 7.4, we present two different P/T structures for the supervisor capable of enforcing GMEC for this sub-class of marked graphs. In Section 7.5, we present two different partially compiled net structures for the supervisor capable of enforcing GMEC for this sub-class of marked graphs. In Section 7.6, we compare the advantages and disadvantages of the different control structures.

7.2 GMEC and Monitors

In this section we define a *generalized mutual exclusion constraint* (GMEC) as a condition that limits the weighted sum of tokens in a set of places. We discuss the modeling power of this kind of constraint and prove that only for restricted classes of systems a *forbidden marking problem* may be expressed as a mutual exclusion problem. Then we study how GMEC may be enforced by adding additional control structure, in the form of new places called *monitors*, to the net.

7.2.1 Generalized Mutual Exclusion Constraints

Definition 7.1. *Let $\langle N, M_0\rangle$ be a net system with set of places P. A single* generalized mutual exclusion constraint *$(\vec{w}, k)$ defines a set of legal markings*

$$\mathcal{M}(\vec{w}, k) = \{M \in I\!N^{|P|} \mid \vec{w}^T \cdot M \leq k\},$$

where $\vec{w} : P \to I\!N$ is a weight vector, and $k \in I\!N^+$. The support *of $\vec{w}$ is the set $Q_w = \{p \in P \mid w(p) > 0\}$.*

A set of generalized mutual exclusion constraints $(W, \vec{k})$, with $W = [\vec{w}_1 \dots \vec{w}_m]$ and $\vec{k} = (k_1 \dots k_m)^T$, defines a set of legal markings

$$\mathcal{M}(W, \vec{k}) = \bigcap_{i=1}^{m} \mathcal{M}(\vec{w}_i, k_i) = \{M \in I\!N^{|P|} \mid W^T \cdot M \leq \vec{k}\}.$$

As a particular case, when $\vec{w} \leq \vec{1}$, i.e., $w(p) = 1$ $(\forall p \in Q_w)$, the *unweighted* GMEC $(\vec{w}, k)$ is reduced to the *set condition* considered in [Krogh 91].

In the following we will discuss the redundancy and equivalence between constraints.

Definition 7.2. *Let $\langle N, M_0\rangle$ be a system. A GMEC $(\vec{w}, k)$ is* redundant *with respect to (wrt) a set of markings $A \subseteq \mathbb{N}^{|P|}$ if $A \subseteq \mathcal{M}(\vec{w}, k)$.*

A GMEC $(\vec{w}, k)$ is redundant wrt a system $\langle N, M_0\rangle$ *if $R(N, M_0) \subseteq \mathcal{M}(\vec{w}, k)$.*

A set of GMEC $(W, \vec{k})$, where $W = [\vec{w}_1 \dots \vec{w}_m]$ and $\vec{k} = (k_1 \dots k_m)^T$, is redundant wrt $\langle N, M_0\rangle$ if $(\vec{w}_i, k_i)$ is redundant for all $i = 1, \dots, m$.

Linear programming techniques may be used to derive sufficient conditions for redundancy.

Proposition 7.1. *If the following Linear Programming Problem (LPP) has optimal solution $x^* < k + 1$ then the GMEC $(\vec{w}, k)$ is* redundant *wrt $\langle N, M_0\rangle$:*

$$\begin{array}{rll} x = & \max & \vec{w}^T \cdot M \\ & s.t. & M = M_0 + C \cdot \vec{\sigma}, \\ & & M, \vec{\sigma} \geq \vec{0}. \end{array}$$

Proof: If $x^* < k + 1$ then $PR(N, M_0) \subseteq \mathcal{M}(\vec{w}, k)$ and this implies that $R(N, M_0) \subseteq \mathcal{M}(\vec{w}, k)$. ◇

The proposition gives a sufficient condition for redundancy. There are classes of nets, such as marked graphs (see Proposition 7.3), for which the condition is necessary and sufficient [Colom 89]. Also for the nets for which $PR(N, M_0) = PR^B(N, M_0)$ we may equivalently check for redundancy solving the following linear programming problem:

$$\begin{array}{rll} x = & \max & \vec{w}^T \cdot M \\ & s.t. & B^T \cdot M = B^T \cdot M_0, \\ & & M \geq \vec{0}. \end{array}$$

where B is a basis of P-semiflows of the net.

Definition 7.3. *Two sets of GMEC $(W_1, \vec{k}_1)$ and $(W_2, \vec{k}_2)$ are* equivalent *wrt $\langle N, M_0\rangle$ if $R(N, M_0) \cap \mathcal{M}(W_1, \vec{k}_1) = R(N, M_0) \cap \mathcal{M}(W_2, \vec{k}_2)$.*

We may check for equivalence between constraints using the same approach we used to check for redundancy. In fact from the definition it follows that two sets of GMEC $(W_1, \vec{k}_1)$ and $(W_2, \vec{k}_2)$ are equivalent wrt $\langle N, M_0\rangle$ if and only if $(W_1, \vec{k}_1)$ is redundant wrt $R(N, M_0) \cap \mathcal{M}(W_2, \vec{k}_2)$, and $(W_2, \vec{k}_2)$ is redundant wrt $R(N, M_0) \cap \mathcal{M}(W_1, \vec{k}_1)$.

Example 7.1. Consider the system in Figure 7.1a whose set of reachable markings is $R(N, M_0) = \{M \mid \vec{1}^T \cdot M = 3\}$. Let $(\vec{w}_1, k_1)$ and $(\vec{w}_2, k_2)$ be two GMEC with: $\vec{w}_1 = (1330)^T, k_1 = 5$, and $\vec{w}_2 = (0110)^T, k_2 = 1$.

To prove that the two constraints are equivalent wrt the system considered we may proceed as follows. The LPP

$$\begin{array}{rll} x_1 = & \max & \vec{w}_1^T \cdot M \\ & s.t. & \vec{1}^T \cdot M = 3, \\ & & \vec{w}_2^T \cdot M \leq 1, \\ & & M \geq \vec{0}, \end{array}$$

has optimal value $x_1^* = 5 < k_1 + 1$, hence by Proposition 7.1 $R(N, M_0) \cap \mathcal{M}(\vec{w}_2, k_2) \subseteq \mathcal{M}(\vec{w}_1, k_1)$. The LPP

$$\begin{array}{rll} x_2 = & \max & \vec{w}_2^T \cdot M \\ & s.t. & \vec{1}^T \cdot M = 3, \\ & & \vec{w}_1^T \cdot M \leq 5, \\ & & M \geq \vec{0}, \end{array}$$

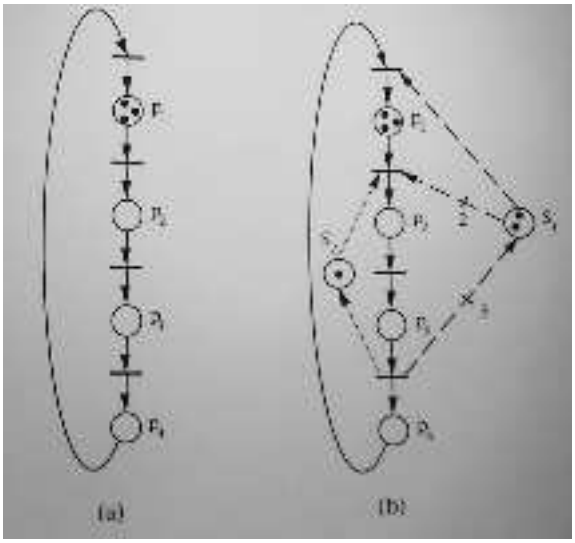


Figure 7.1: System in Example 7.1.

has optimal value $x_2^* = \frac{5}{3} < k_2 + 1$, hence $R(N, M_0) \cap \mathcal{M}(\vec{w}_1, k_1) \subseteq \mathcal{M}(\vec{w}_2, k_2)$. This proves that the two constraints are equivalent for the given system.

The equivalence between constraints leads to the idea of *simplification* of a constraint. Given a constraint $(\vec{w}, k)$, we may look for a simpler, but equivalent, constraint. A constraint $(\vec{w}', k')$ is simpler than $(\vec{w}, k)$ if $\vec{w}' < \vec{w}$. In the next subsection we will see that simpler constraints require simpler control structure to be enforced. The next example shows another advantage of simplifying constraints.

Example 7.2. For the system in Figure 7.1a consider, in addition to the two constraints discussed in Example 7.1, the constraint $(\vec{w}_3, k_3)$ with: $\vec{w}_3 = (0330)^T, k_3 = 5$. By definition, $(\vec{w}_2, k_2)$ is simpler than $(\vec{w}_3, k_3)$ that is simpler than $(\vec{w}_1, k_1)$. It is immediate to see that $\mathcal{M}(\vec{w}_3, k_3) = \mathcal{M}(\vec{w}_2, k_2)$, i.e., $(\vec{w}_3, k_3)$ is equivalent to $(\vec{w}_2, k_2)$ wrt $\langle N, M_0 \rangle$. Since we have proved in Example 7.1 that $(\vec{w}_1, k_1)$ is equivalent to $(\vec{w}_2, k_2)$, the equivalence between $(\vec{w}_1, k_1)$ and $(\vec{w}_3, k_3)$ also follows.

Note, however, that if we try to use a LPP to prove that $(\vec{w}_1, k_1)$ is redundant wrt $R(N, M_0) \cap (\vec{w}_3, k_3)$ we have an inconclusive answer. In fact the LPP

$$\begin{array}{rll} x_1 = & \max & \vec{w}_1^T \cdot M \\ & s.t. & \vec{1}^T \cdot M = 3, \\ & & \vec{w}_3^T \cdot M \leq 5, \\ & & M \geq \vec{0}, \end{array}$$

has optimal value: $x_1^* = \frac{19}{3} > 6$, hence we cannot conclude that $(\vec{w}_1, k_1)$ is redundant wrt $R(N, M_0) \cap (\vec{w}_3, k_3)$.

It is important, in the previous example, to pinpoint the advantage of using the simpler unweighted constraint $(\vec{w}_2, k_2)$ rather than the weighted one $(\vec{w}_3, k_3)$ to prove equivalence to $(\vec{w}_1, k_1)$. The system considered in the example is a live marked graph, and the constraint set that defines the set of reachable markings has integer extremal points, hence any optimal solution of the Linear Programming Program is also a solution of the corresponding Integer Programming Problem. This property is preserved if we add any number of unweighted constraints to the constraint set that defines the set of reachable markings.

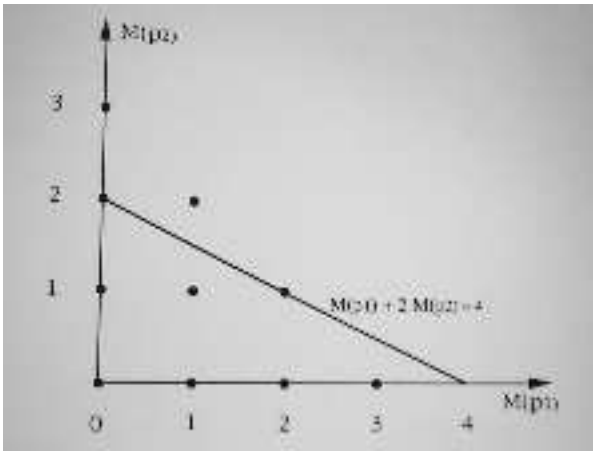


Figure 7.2: Reachable markings in Example 7.5.

7.2.2 Modeling Power of Generalized Mutual Exclusion Constraints

In this subsection we discuss the modeling power of GMEC, both weighted and unweighted.

The use of weights in the definition of $(\vec{w}, k)$ may be a useful way to compactly express more than one unweighted constraint, i.e., it may be the case that a weighted constraint may be decomposed into a set of unweighted ones.

Example 7.3. In the case of safe (i.e., 1-bounded) systems, the constraint $(\vec{w}, k)$ with $\vec{w} = (1234)^T$ and $k = 5$ is equivalent to the set of constraints $(W, \vec{k})$ with

$$W = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix},$$

and $\vec{k} = (211)^T$. In fact $\mathcal{M}(\vec{w}, k) \cap \{0,1\}^4 = \mathcal{M}(W, \vec{k}) \cap \{0,1\}^4$.

We point out that there does not always exist a set of unweighted constraints equivalent to a set of weighted ones.

Example 7.4. Consider again the system in Figure 7.1a. Let $(\vec{w}, k)$ be a GMEC with: $\vec{w} = (1200)^T, k = 4$. In Figure 7.2 we have represented the projection of the reachability set on the first two components, $M(p_1)$ and $M(p_2)$. Markings $M_1 = (2100)^T$ and $M_2 = (0210)^T$ are legal, while marking $M_3 = (1200)^T$ is forbidden by $(\vec{w}, k)$.

If there exists a set of unweighted constraints $(W, \vec{k})$ equivalent to $(\vec{w}, k)$, then one of the constraints in this set must be $(\vec{w}', k')$, with $\vec{w}' = \vec{1}$, such that $M_1, M_2 \in \mathcal{M}(\vec{w}', k')$, and $M_3 \notin \mathcal{M}(\vec{w}', k')$. We will prove, by contradiction, that no such $(\vec{w}', k')$ may exists. In fact, $\vec{w}' \cdot M_1 < \vec{w}' \cdot M_3 \Longrightarrow w'(p_1) < w'(p_2)$, and $\vec{w}' \cdot M_2 < \vec{w}' \cdot M_3 \Longrightarrow w'(p_3) < w'(p_1)$. That is, in order to have an unweighted constraint that forbids M_3 but that does not forbid M_1 and M_2 we need to chose a $\vec{w}'$ such that $(\forall p)\ w'(p) \in \{0,1\}$ and $w'(p_3) < w'(p_1) < w'(p_2)$. This is clearly impossible.

For safe systems, however, the following theorem proves that any weighted constraint is equivalent to a set of unweighted constraints.

Theorem 7.1. *Let $\langle N, M_0 \rangle$ be a safe system with set of places P and $(\vec{w}, k)$ a weighted GMEC. There exists a set of unweighted constraints $(W, \vec{k})$ equivalent to $(\vec{w}, k)$ wrt $\langle N, M_0 \rangle$.*

Proof: Consider the set of vectors V such that $\forall \vec{v} \in V$ the following conditions are verified:

C1. $\vec{v} \in \{0,1\}^{|P|}$;

C2. $Q_v \subseteq Q_w$;

C3. $\vec{w}^T \cdot \vec{v} > k$;

C4. $(\forall \vec{v}' \in \{0,1\}^{|P|}, \vec{v}' < \vec{v})\ \ \vec{w}^T \cdot \vec{v}' \leq k$.

Let $(W, \vec{k})$ be such that $W = (\vec{w}_1 \dots \vec{w}_r)$ and $\vec{k} = (k_1 \dots k_r)^T$, where

$$\bigcup_{i=1}^{r} \{\vec{w}_i\} = V,$$

and where $(i = 1, \dots, r)$ $k_i = |Q_{w_i}| - 1$.

a) Let us prove $R(N, M_0) \cap \mathcal{M}(\vec{w}, k) \subseteq \mathcal{M}(W, \vec{k})$.
$M \in R(N, M_0) \cap \mathcal{M}(\vec{w}, k) \Longrightarrow M \leq \vec{1} \wedge \vec{w}^T \cdot M \leq k \Longrightarrow M \leq \vec{1} \wedge (\forall \vec{v} \in V)\ \exists p \in Q_v \ni M(p) = 0 \Longrightarrow (\forall \vec{v} \in V)\ \vec{v}^T \cdot M \leq k_i \Longrightarrow M \in \mathcal{M}(W, \vec{k})$.

b) Let us prove $R(N, M_0) \cap \mathcal{M}(W, \vec{k}) \subseteq \mathcal{M}(\vec{w}, k)$. It is enough to prove that $M \in R(N, M_0) \wedge M \notin \mathcal{M}(\vec{w}, k) \Longrightarrow M \notin \mathcal{M}(W, \vec{k})$.
$M \in R(N, M_0) \wedge M \notin \mathcal{M}(\vec{w}, k) \Longrightarrow M \leq \vec{1} \wedge \vec{w}^T \cdot M > k$. Let $\vec{v}_0$ be defined as: if $p \in Q_w$ then $v_0(p) = M(p)$ else $v_0(p) = 0$. Clearly $\vec{v}_0$ satisfies conditions C1-C3 listed above. We will show that there exists vector $\vec{v}_j \leq \vec{v}_0$ and such that $\vec{v}_j \in V$. Consider $p' \in Q_{v_0} \ni (\forall p \in Q_{v_0}) w(p') \leq w(p)$. Let $\vec{v}_1$ be a new vector such that: if $p \neq p'$ then $v_1(p) = v_0(p)$ else $v_1(p) = 0$. If $\vec{w}^T \cdot \vec{v}_1 \leq k$ stop else repeating this procedure construct $\vec{v}_2, \cdots, \vec{v}_{j+1}$ such that $M \geq \vec{v}_0 > \vec{v}_1 > \cdots > \vec{v}_{j+1}$ and $\vec{w}^T \cdot \vec{v}_{j+1} \leq k$ while $\vec{w}^T \cdot \vec{v}_j > k$. This means that $\vec{v}_j \in V$, hence $\vec{v}_j^T \cdot M = |Q_{v_j}| \Longrightarrow M \notin \mathcal{M}(W, \vec{k})$. ◇

In the rest of this subsection we will compare GMEC with the most general kind of constraint that can be defined on the markings of a system, the *forbidden markings* constraint [Holloway 92a]. A forbidden marking constraint consists of an *explicit list* of markings F that we want to forbid.

Let us now consider a net system $\langle N, M_0 \rangle$ and let F be any set of forbidden markings. Is it possible to find a set of GMEC $(W, \vec{k})$ equivalent to F, i.e., such that $R(N, M_0) \backslash F = R(N, M_0) \cap \mathcal{M}(W, \vec{k})$? In general the answer is no. In fact given three markings $M_1, M_2, M_3 \in R(N, M_0)$ with $M_3 = (M_1 + M_2)/2$ we have that $M_1, M_2 \in \mathcal{M}(W, \vec{k}) \Longrightarrow M_3 \in \mathcal{M}(W, \vec{k})$, since $\mathcal{M}(W, \vec{k})$ is a convex set. However F may be chosen such that $M_1, M_2 \notin F$ and $M_3 \in F$. This proves that there may not exist an GMEC equivalent to a forbidden marking constraint.

However it is possible to prove that for some classes of nets there exists an GMEC equivalent to any forbidden marking constraint.

Theorem 7.2. *Let $\langle N, M_0 \rangle$ be a safe and conservative net system. Then given a set of forbidden markings F there exists a set of GMEC $(W, \vec{k})$ such that $R(N, M_0) \setminus F = R(N, M_0) \cap \mathcal{M}(W, \vec{k})$.*

Proof: Let us first state two obvious facts.

1. If a net is safe there does not exist two different markings with the same support, i.e., $M, M' \in R(N, M_0) \wedge Q_M = Q_{M'} \Longrightarrow M = M'$.

2. If a net is conservative no marking is covering another one, i.e., $(\forall M \in R(N, M_0))\ \nexists M' \in R(N, M_0) \ni M < M'$.

Then, given a set of forbidden markings F we may forbid any $M \in F$ with a constraint $(\vec{w}, k)$ where: $\vec{w}(p) = 1$ if $p \in Q_M$ else $\vec{w}(p) = 0$, and $k = |Q_M| - 1$. Clearly $M \notin \mathcal{M}(\vec{w}, k)$ and any other marking $M' \in R(N, M_0)$ is such that $M' \in \mathcal{M}(\vec{w}, k)$. Thus $(W, \vec{k})$ may be constructed as the union of all the GMEC constraints forbidding a marking in F. ◇

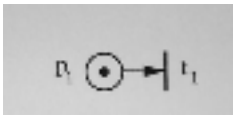


Figure 7.3: System in Example 7.6.

The requirement that the net be conservative may be shown necessary by the following example.

Example 7.5. Consider the 1-bounded but not conservative system in Figure 7.3. The two possible markings of the system are $M_1 = (1)$ and $M_2 = (0)$. Clearly for a set of forbidden markings $F = \{(0)\}$ it is not possible to find an equivalent GMEC since $(\forall \vec{w}, k)\ \vec{w}^T \cdot M_2 \leq \vec{w}^T \cdot M_1 \leq k$.

7.2.3 Monitors

A GMEC may be enforced on a system by a *monitor*.

Definition 7.4. *Given a system $\langle N, M_0 \rangle$, with $N = (P, T, Pre, Post)$, and a GMEC $(\vec{w}, k)$, the* monitor *that enforces this constraint is a new place S to be added to N. The resulting system is denoted $\langle N^S, M_0^S \rangle$, with*

$$N^S = (P \cup \{S\}, T, Pre^S, Post^S).$$

Let C be the incidence matrix of N. Then N^S will have incidence matrix

$$C^S = \begin{bmatrix} C \\ -\vec{w}^T \cdot C \end{bmatrix}.$$

We are assuming that there are no selfloops containing S in N^S, hence Pre^S and $Post^S$ may be uniquely determined by C^S. The initial marking of $\langle N^S, M_0^S \rangle$ is

$$M_0^S = \begin{pmatrix} M_0 \\ k - \vec{w}^T \cdot M_0 \end{pmatrix}.$$

We assume that the initial marking M_0 of the system satisfies the constraint $(\vec{w}, k)$.

As an example, in Figure 7.1b we have represented the two monitors corresponding to the two constraints discussed in Example 7.1.

We will use the following notation for system with monitors. Let $X : P \cup \{S\} \rightarrow I\!N$ be a vector. The *projection* of X on P is the restriction of X to P and will be denoted $X \uparrow_P$. This definition is extended in the usual way to the projection of a set of vectors $\mathcal{X}$, i.e., $\mathcal{X} \uparrow_P = \{X \uparrow_P | \ X \in \mathcal{X}\}$.

Proposition 7.2. *Let $\langle N, M_0 \rangle$ be a system, $(\vec{w}, k)$ a GMEC, and $\langle N^S, M_0^S \rangle$ the system with the addition of the corresponding monitor S.*

1. *The monitor S ensures that the projection on P of the reachability set of $\langle N^S, M_0^S \rangle$ is contained in the set of legal reachable markings of $\langle N, M_0 \rangle$, i.e.,*

$$R(N^S, M_0^S) \uparrow_P \subseteq R(N, M_0) \cap \mathcal{M}(\vec{w}, k).$$

2. *The monitor S ensures that the projection on P of the potentially reachable set of $\langle N^S, M_0^S \rangle$ is identical to the set of legal potentially reachable markings of $\langle N, M_0 \rangle$, i.e.,*

$$PR(N^S, M_0^S) \uparrow_P = PR(N, M_0) \cap \mathcal{M}(\vec{w}, k).$$

3. *The monitor S minimally restricts the behavior of $\langle N^S, M_0^S\rangle$, in the sense that it prevents only transition firings that yield forbidden markings.*

4. *The monitor S is a structurally implicit place in N^S if and only if all places in Q_w are structurally bounded.*

5. *The monitor S is an implicit place in $\langle N^S, M_0^S\rangle$ if and only if $(\vec{w}, k)$ is redundant in $\langle N, M_0\rangle$.*

Proof:

1] Clearly $R(N^S, M_0^S) \uparrow_P \subseteq R(N, M_0)$, since the addition of a place can only further constrain the behavior of a system. To prove $R(N^S, M_0^S) \uparrow_P \subseteq \mathcal{M}(\vec{w}, k)$, let $M^S \in R(N^S, M_0^S)$ and $M = M^S \uparrow_P$. Then $M^S = M_0^S + C^S \cdot \vec{\sigma}$ or, equivalently,

$$M = M_0 + C \cdot \vec{\sigma},$$

$$M^S(S) = M_0^S(S) - \vec{w}^T \cdot C \cdot \vec{\sigma} = k - \vec{w}^T \cdot (M_0 + C \cdot \vec{\sigma}) \geq 0.$$

Hence $\vec{w}^T \cdot M = \vec{w}^T \cdot (M_0 + C \cdot \vec{\sigma}) \leq k$, i.e., $M \in \mathcal{M}(\vec{w}, k)$.

2] With the same reasoning of the previous point we can immediately conclude that $PR(N^S, M_0^S) \uparrow_P \subseteq PR(N, M_0) \cap \mathcal{M}(\vec{w}, k)$. Let us prove the reverse inclusion. Let $M \in PR(N, M_0) \cap \mathcal{M}(\vec{w}, k)$, i.e., $\exists \vec{\sigma} \geq 0$ such that

$$M = M_0 + C \cdot \vec{\sigma},$$

$$\vec{w}^T \cdot M \leq k.$$

This implies that $\vec{w}^T(M_0 + C \cdot \vec{\sigma}) \leq k$, i.e., $k - \vec{w}^T \cdot (M_0 + C \cdot \vec{\sigma}) \geq 0$. Then we also have that

$$M^S = \begin{pmatrix} M \\ k - \vec{w}^T \cdot (M_0 + C \cdot \vec{\sigma}) \end{pmatrix}$$

is a non negative solution of $M^S = M_0^S + C^S \cdot \vec{\sigma}$, i.e., $M^S \in PR(N^S, M_0^S)$.

3] Let $\sigma t \in L(N, M_0)$ be such that: $M_0[\sigma\rangle M[t\rangle M'$ and $\sigma \in L(N^S, M_0^S)$ be such that: $M_0^S[\sigma\rangle M^S$. We need to prove that $\sigma t \notin L(N^S, M_0^S) \Longrightarrow \vec{w}^T \cdot M' > k$. Let $C(\cdot, t)$ be the column of C corresponding to transition t. Then $Pre^S(S, t) - Post^S(t, S) = -C^S(S, t) = \vec{w}^T \cdot C(\cdot, t)$. Since t is not enabled by marking M^S and since there are no selfloops containing S, it follows that

$$0 \leq M^S(S) < Pre^S(S, t) \Longrightarrow Post^S(t, S) = 0,$$

i.e., $Pre^S(S, t) = \vec{w}^T \cdot C(\cdot, t)$. Then

$$k - \vec{w}^T \cdot M = M^S(S) < Pre^S(S, t) = \vec{w}^T \cdot C(\cdot, t),$$

from which follows

$$\vec{w}^T \cdot M' = \vec{w}^T \cdot [M + C(\cdot, t)] > k.$$

4] As shown in [Colom 89], S is structurally implicit in N^S if and only if $\exists Y \geq \vec{0}$ such that $Y^T \cdot C^S \leq C^S(S, \cdot)$, where $C^S(S, \cdot) = -\vec{w}^T \cdot C$ is the incidence vector of S in C^S. Then: All places in Q_w are structurally bounded $\Longleftrightarrow \exists Y' \geq \vec{0}$ such that $Q_{Y'} \supseteq Q_w$, ${Y'}^T \cdot C \leq \vec{0} \Longleftrightarrow \exists a \in \mathbb{R}^+$ such that $Y = (aY' - \vec{w}) \geq \vec{0}$, $Y^T \cdot C^S \leq -\vec{w}^T \cdot C \Longleftrightarrow S$ is structurally implicit in N^S.

5] *Only if.* S is implicit $\Longleftrightarrow L(N, M_0) = L(N^S, M_0^S) \Longrightarrow R(N, M_0) = R(N^S, M_0^S) \uparrow_P \subseteq \mathcal{M}(\vec{w}, k) \Longleftrightarrow (\vec{w}, k)$ is redundant in $\langle N, M_0\rangle$.

If. We just need to prove that if $(\vec{w}, k)$ is redundant in $\langle N, M_0\rangle$ then $L(N, M_0) = L(N^S, M_0^S)$. Clearly $L(N, M_0) \supseteq L(N^S, M_0^S)$, so we will prove that redundancy implies $L(N, M_0) \subseteq L(N^S, M_0^S)$.

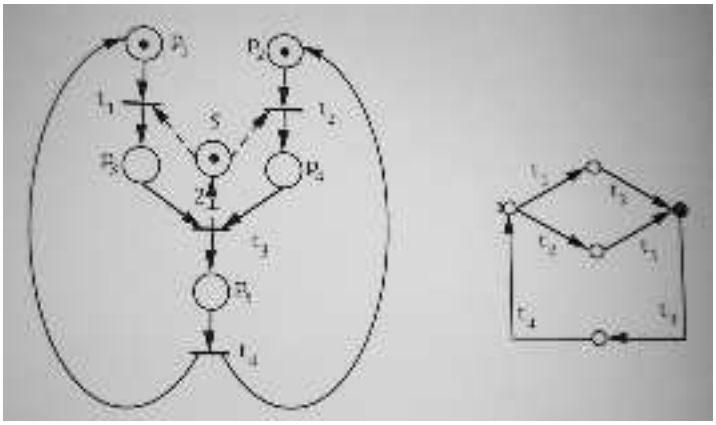


Figure 7.4: A system not live under constraint.

Let $\sigma t \in L(N, M_0)$ be such that: $M_0[\sigma\rangle M[t\rangle M'$ and $\sigma \in L(N^S, M_0^S)$ be such that: $M_0^S[\sigma\rangle M^S$. Note that redundancy implies $\vec{w}^T \cdot M' = \vec{w}^T \cdot [M + C(\cdot, t)] \leq k$. Then $M^S(S) = k - \vec{w}^T \cdot M \geq \vec{w}^T \cdot C(\cdot, t) = Pre^S(S, t) - Post^S(t, S)$ and by the hypothesis that S is not contained in a selfloop we have: $M^S(S) \geq Pre^S(S, t)$. This proves that $\sigma t \in L(N^S, M_0^S)$. $\diamond$

The addition of a monitor to the net structure modifies the behavior of a system, in order to avoid reaching markings that do not satisfy the corresponding GMEC. We pinpoint three facts:

- The addition of a monitor does not always preserve liveness of the system. In the system in Figure 7.4, e.g., a monitor has been added to enforce the constraint $M(p_3) + M(p_4) \leq 1$. The system reaches a deadlock after the firing of a single transition.
- Not all markings that satisfy the GMEC may be reached on the net with the addition of a monitor. In the net in Figure 7.4, e.g., the marking $(000011)^T$ may not be reached even if it satisfies the constraint $M(p_3) + M(p_4) \leq 1$ because the net reaches a deadlock. In the net in Figure 7.5, a monitor has been added to enforce the constraint $M(p_1) + M(p_3) \leq 1$. From the initial marking $M_0^S = (000111)^T$ the marking $M^S = (100010)^T$ will never be reached even if the net is live.
- Even if liveness is preserved, the system may lose reversibility, as shown in the system in Figure 7.5. In the same figure is shown the reachability graph of the original system (all arcs and states) and of the system with monitor (only continuous arcs). The initial marking, that satisfies the constraint, will never be reached again in the system with monitor.

7.2.4 Nets With Uncontrollable Transitions

We assume, now, that the set of transitions T of a net is partitioned into the two disjoints subsets T_u, the set of *uncontrollable* transitions, and T_c, the set of *controllable* transitions. A controllable transition may be disabled by the supervisor, a controlling agent which ensures that the behavior of the system is within a legal behavior. An uncontrollable transition represents an event which may not be prevented from occurring by a supervisor.

Given a system $\langle N, M_0\rangle$ and a set of GMEC $(W, \vec{k})$, the set of legal markings is given as a linear domain:

$$\mathcal{M}(W, \vec{k}) = \{M \in I\!N^{|P|} \mid W^T \cdot M \leq \vec{k}\}.$$

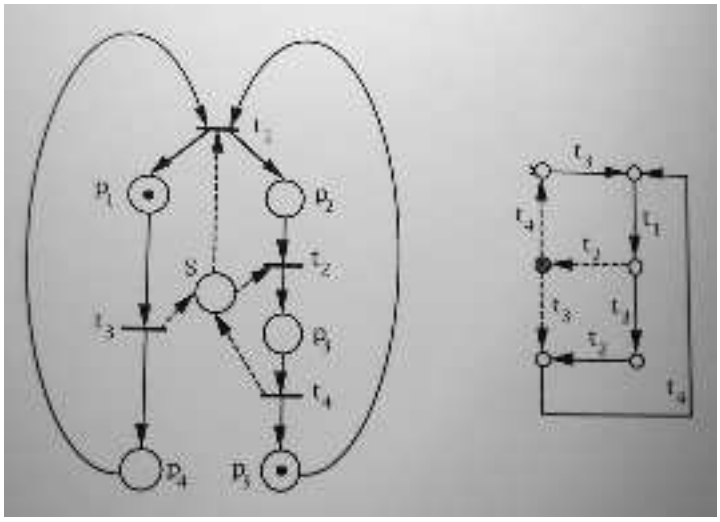


Figure 7.5: A system not reversible under constraint.

In the presence of uncontrollable transitions, we need to further restrict the behavior of the system, avoiding all those markings from which a forbidden marking may be reached by firing only uncontrollable transitions. The set of legal marking is in this case:

$$\mathcal{M}_c(W,\vec{k}) = \mathcal{M}(W,\vec{k}) \setminus \{M \in I\!N^{|P|} \mid \exists M' \notin \mathcal{M}(W,\vec{k}) \ni M[\sigma\rangle M' \wedge \sigma \in T_u^*\},$$

i.e., we do not consider legal the markings that satisfy $(W,\vec{k})$ but from which a forbidden marking may be reached by firing only uncontrollable transitions. We need to introduce this restriction because a firing sequence $\sigma \in T_u^*$ may not be prevented by a controlling agent.

It is possible to prove that there does not always exist a GMEC $(W,\vec{k})$ such that $R(N,M_0) \cap \mathcal{M}(W,\vec{k}) = R(N,M_0) \cap \mathcal{M}_c(\vec{w},k)$. Thus we may have cases in which does not exist a monitor-based solution to a given mutual exclusion problem.

Example 7.6. In the net in Figure 7.6, we have represented as empty boxes the controllable transitions t_1, t_2, t_5. Assume we want to enforce a constraint $(\vec{w},k)$ with $\vec{w} = (00100010)^T$ and $k = 1$, i.e., such that $M(p_5) + M(p_7) \leq 1$. Is is easy to see that the markings $M_1 = (2002001)^T$ and $M_2 = (0220001)^T$ are in $\mathcal{M}_c(\vec{w},k)$, but $M = (1111001)^T = (M_1 + M_2)/2$ is not. This proves that there does not exists a GMEC $(W,\vec{k})$ such that $R(N, M_0 \cap \mathcal{M}(W,\vec{k}) = R(N,M_0) \cap \mathcal{M}_c(\vec{w},k)$.

This shows that in presence of *uncontrollable* transitions, a problem of mutual exclusion is transformed into a more general *forbidden marking problem*, which is a qualitatively different problem, in the sense that it may not always be solved with the same techniques used in the case that all transitions are controllable. Note, however, that for safe and conservative systems the result of Theorem 7.2 ensures that, even if some transitions are not controllable, $(\vec{w},k)$ may be enforced by a set of monitors.

We also discuss the concept of *maximally permissible control policy* [Krogh 87]. When all transitions are controllable, we have proved that a monitor is capable of enforcing a given GMEC $(\vec{w},k)$, in the sense that it can ensure that only markings in $\mathcal{M}(\vec{w},k)$ will be reached by the system under control (Proposition 7.2, part 1). It is also the case that whenever a transition t is prevented from firing by the monitor at a given marking M then $M[t\rangle M' \wedge M' \notin \mathcal{M}(\vec{w},k)$ (Proposition 7.2, part 3). Thus any transition firing that yields a legal marking is not forbidden and the behavior of the system under control is the largest behavior that satisfies the constraint. We call this behaviour *maximally permissible* following [Krogh 87].

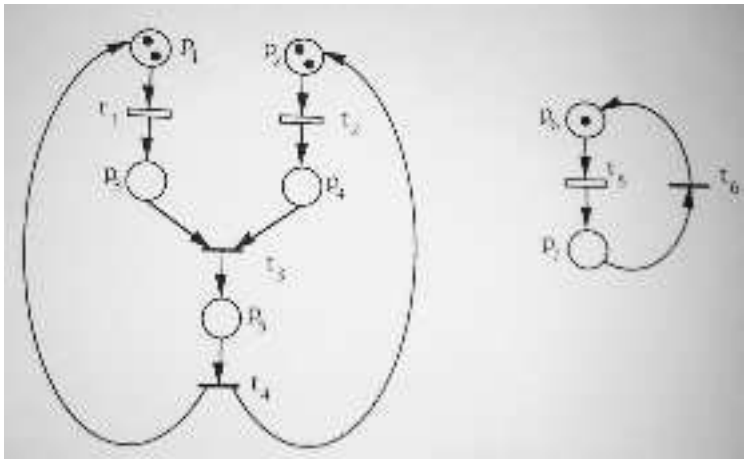


Figure 7.6: A mutual exclusion problem with uncontrollability that does not admit a monitor-based solution.

In the case of uncontrollable transitions, the maximally permissible control policy should ensure that: a) only markings in $\mathcal{M}_c(\vec{w}, k)$ will be reached by the system under control; b) all transition firings that yield a marking in $\mathcal{M}_c(\vec{w}, k)$ should be allowed.

7.3 Generalized Mutual Exclusion Constraints on Marked Graphs

In this section we focus on a class of nets, the marked graphs (MG). In the first subsection we study the properties of marked graphs with the addition of monitors. In the second subsection we present a general formalism for enforcing generalized mutual exclusion constraints on MG with uncontrollable transitions. In the third subsection we restrict our attention to a particular subclass of MG for which a solution to a mutual exclusion problem may be efficiently derived. In Section 7.4 and Section 7.5 we will show several control structure capable of enforcing generalized mutual exclusion constraints, in the presence of uncontrollable transitions, for this subclass.

7.3.1 Marked Graphs with Monitors

In the rest of this chapter, we will consider systems whose underlying net is a strongly connected marked graph (MG). Strongly connected MG are structurally live and bounded.

Proposition 7.3 ([Murata 89]). *Let $\langle N, M_0\rangle$ be a MG system. Then*

$$PR(N, M_0) = B(N, M_0).$$

If $\langle N, M_0\rangle$ is a live MG system, then we also have that

$$R(N, M_0) = PR(N, M_0).$$

Proposition 7.4. *Let $\langle N, M_0\rangle$ be an MG system, $(W, \vec{k})$ be a set of GMEC, and let $\langle N^S, M_0^S\rangle$ be the system with the addition of the corresponding monitors. Then $PR(N^S, M_0^S) \uparrow_P = B(N, M_0) \cap \mathcal{M}(W, \vec{k})$.*

Proof: Follows from Proposition 7.2, part 2, and Propositon 7.3. ◇

According to this proposition it is possible to represent the potential reachability set of $\langle N^S, M_0^S\rangle$ as a set of linear inequalities where the firing count vector $\vec{\sigma}$ does not appear.

Proposition 7.5 ([Murata 89]). *A connected MG has a single minimal t-semiflow,* $X = \vec{1}$.

The introduction of a set of monitors corresponding to a set of GMEC does not change this property. The resulting net will be a *mono-t-semiflow* net [Campos 91]. For mono-t-semiflow nets *deadlock-freeness* is equivalent to *liveness*. The following proposition gives a sufficient condition for liveness.

Proposition 7.6. *Let* $\langle N, M_0\rangle$ *be a live MG system,* $(W, \vec{k})$ *a set of GMEC, and* $\langle N^S, M_0^S\rangle$ *the system with the addition of the corresponding monitors.* $\langle N^S, M_0^S\rangle$ *is* live *if* $\forall M \in B(N, M_0) \cap \mathcal{M}(W, \vec{k})$, M *is not a dead marking.*

Proof: If no dead marking exists in $PR(N^S, M_0^S)$ then no dead marking exists in $R(N^S, M_0^S)$ and the system is deadlock-free. This in turn implies liveness, since the net is a mono-t-semiflow net. ◇

Note that there may exist dead markings in $B(N, M_0) \cap \mathcal{M}(W, \vec{k})$ that are not reachable under control. These markings are called *killing spurious markings* in structural jargon. Thus the previous proposition gives a sufficient but not necessary condition for liveness. The next example shows that, unfortunately, killing spurious markings may exists on marked graphs with monitors.

Example 7.7. Consider the system in Figure 7.7, whose reachability graph is also shown. Assume we want to enforce the two constraints: $3M(p_1) + M(p_3) \leq 9$, and $M(p_2) + 3M(p_4) \leq 9$. The legal marking of this system under the constraints are shown in Figure 7.8. Let the initial marking be $M_0 = (2130)^T$. The dead marking $M = (3003)^T$ is in $PR(N^S, M_0^S)\uparrow_P$, since $M \in B(N, M_0) \cap \mathcal{M}(W, \vec{k})$, but since it is never reachable the system is live.

Proving reversibility for a MG with monitors $\langle N^S, M_0^S\rangle$ is a harder task. Liveness and the existence of repetitive sequence firable from M_0 are not sufficient conditions for reversibility, as shown in the following example.

Example 7.8. In the system in Figure 7.7 we want to enforce the two constraints: $2M(p_1) + 3M(p_4) \leq 9$, and $2M(p_2) + 3M(p_3) \leq 9$. The legal marking of this system under the constraints are shown in Figure 7.9. The system is live from any legal initial marking. However, from initial marking $M_0 = (0303)^T$ the system is not reversible, even if there exist a repetitive sequence firable from M_0.

7.3.2 Control Subnet

In this subsection we discuss a general methodology for enforcing constraints on systems with uncontrollable transitions. To enforce a constraint we need to prevent some transition firing. Unfortunately, we cannot prevent the firing of an uncontrollable transition $t \in T_u$. We may, however, prevent the firing of a set of controllable transitions (called *control transitions of* t) whose firing is required prior to the firing of t.

Definition 7.5. *Let* $N = (P, T, Pre, Post)$ *be a net, and let* $t_i \in T_u$ *be an uncontrollable transition.*

The control subnet *for* t_i *is the subnet* $N_i = (P_i, T_i, Pre_i, Post_i)$ *where* $P_i \subseteq P$ *is the set of places connected to* t *by a path containing only uncontrollable transitions,* $T_i = {}^\bullet P_i \cap P_i^\bullet$, *and* $Pre_i = Pre \cap (P_i \times T_i)$, $Post_i = Post \cap (P_i \times T_i)$.

The set of control transitions *for* t *is the set* $A_i = {}^\bullet P_i \setminus P_i^\bullet$. *It is obvious that* $A_i \subseteq T_c$.

We may extend this definition to controllable transitions as well. Given a net $N = (P, T, Pre, Post)$ and a controllable transition $t_i \in T_c$, the control subnet for t_i is not defined but the set of control

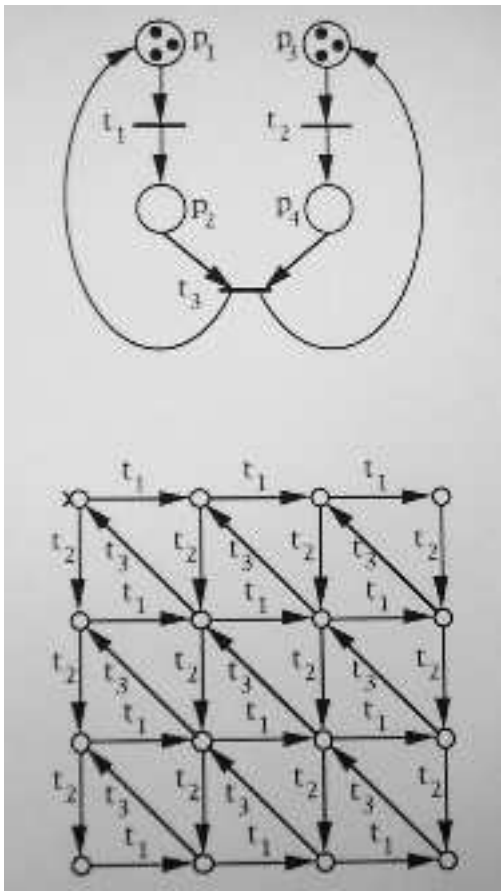


Figure 7.7: A marked graph system and its reachability graph.

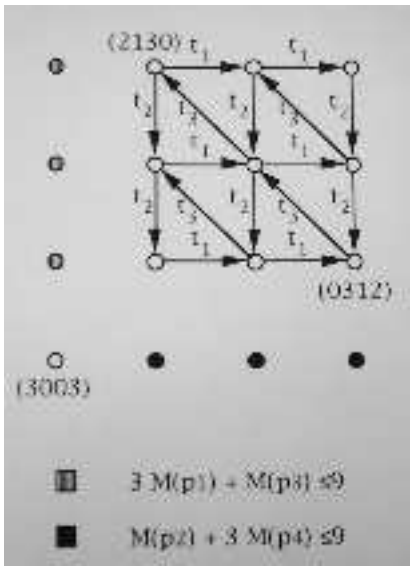


Figure 7.8: Legal markings for the system in Figure 7 under the set of constraints shown.

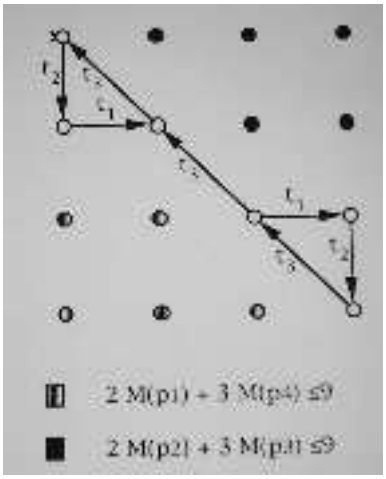


Figure 7.9: Legal markings for the system in Figure 7 under the set of constraints shown.

transitions for t_i is the set $A_i = \{t_i\}$, i.e., the transition itself. This will allows us, in the following, to use the same formalism for both controllable and uncontrollable transitions.

For marked graphs systems if it possible to analytically compute the dependency between the firing of an uncontrollable transition and the firing of its control transitions. The advantage is that this computation may be done on the structure of net, without resorting to the construction of the space of reachable markings.

Proposition 7.7. *Let $\langle N, M_0\rangle$ be an MG system and t_i one of its transitions. Let $N_i = (P_i, T_i, Pre_i, Post_i)$ be the control subnet for t_i, and let A_i be the set of control transitions for t_i.*

1. *$T_i \subseteq T_u$. This means that $A_i \cap T_i = \emptyset$.*
2. *Given a marking $M \in R(N, M_0)$, the maximum number of times we may fire t_i without firing any transition in A_i (the* deviation bound *between t_i and A_i) is*

$$DB(M, t_i, A_i) = \min\{td(M, t', t_i) \mid t' \in A_i\},$$

where $td(M, t', t_i)$ is the token distance between transitions t' and t_i, i.e., the minimum token content among all possible direct paths from t' to t_i at marking M.

Proof:

1] By contradiction, if $t' \in T_i \cap T_c$, then the input place of t' cannot be in P_i, as its only output transition is controllable, hence $t' \notin T_i$.

2] See Murata [Murata 89]. Note that the token distance $DB(M, t, A_i)$ may be computed solely from the analysis of N_i and its marking. Note also that if t_i is controllable $DB(M, t_i, A_i) = DB(M, t_i, t_i) = 0$. ◇

We want to apply the ideas developed so far to the problem of enforcing GMEC on marked graph systems. Given a constraint $(\vec{w}, k)$, the problem is that of controlling the firing of the input transition of all places in Q_w to ensure that the constraint is always verified. We will use the following notation. Given a place $p_i \in Q_w$ we will denote:

- t_i^o its output transition;
- t_i its input transition;
- $N_i = (P_i, T_i, Pre_i, Post_i)$ the control subnet for t_i;

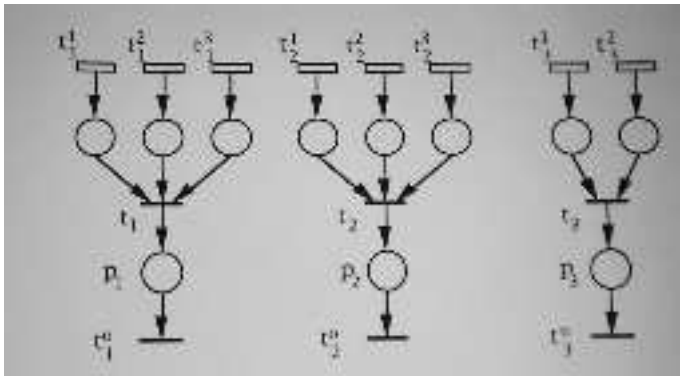


Figure 7.10: Control subnets for the input transitions of places p_1, p_2, p_3.

- $A_i = \{t_i^1, \cdots, t_i^{n_i}\}$ the set of control transitions for t_i.

Example 7.9. In Figure 7.10 places p_1, p_2, p_3 belong to the support of a GMEC. We have represented the control subnets for their input transitions, with the corresponding control transitions. The remaining structure of the marked graph is not shown.

The deviation bound between an uncontrollable transition and its set of control transitions may be used to define the set of legal markings $\mathcal{M}_c(\vec{w}, k)$ for a GMEC $(\vec{w}, k)$ on marked graphs, under the hypothesis discussed in the following.

Proposition 7.8. *Let $\langle N, M_0\rangle$ be an MG system and $(\vec{w}, k)$ a GMEC. For each place $p_i \in Q_w$, let its input transition be t_i. Assume that: $(\bigcup_{p_i \in Q_w} P_i) \cap Q_w = \emptyset$. Then the set of legal markings is*

$$\mathcal{M}_c(\vec{w}, k) = \{M \in I\!N^{|P|} \mid \vec{w}^T \cdot (M + D_M) \leq k\},$$

where $D_M(p_i) = DB(M, t^i, A^{t^i})$ **if** *$p_i \in Q_w$* **else** *$D_M(p_i) = 0$.*

Proof: Proposition 7.7, part 1, and the assumption that no place p_i belongs to the subnet of any t_i, implies that by firing only uncontrollable transitions we may reach a new marking M' such that $M'(p_i) = M(p_i) + D_M(p_i)$. ◇

The previous proposition may be used to define the *maximally permissible control policy* that enforces the constraint $(\vec{w}, k)$. Given a marked graph system $\langle N, M_0\rangle$, the maximally permissible control may be computed step by step as follows. Let $A = (\bigcup_{p_i \in Q_w} A_i)$. Then from any marking $M \in R(N, M_0)$:

- For each firable $t \in A$, compute $M' \ni M[t\rangle M'$. If $\vec{w}^T \cdot (M' + D_{M'}) \leq k$ then t should be left free to fire else it should be disabled;
- All transitions in $T_c \setminus A$ may be left free to fire.

7.3.3 Control Safe Places

We will consider in the following a special class of MG systems introduced in the next definition.

Definition 7.6. *A place p_i of an MG system $\langle N, M_0\rangle$ is said to be* control safe *if given its input transition t_i and its output transition t_i^o, the deviation bound between any control transition in A_i and both t_i and t_i^o is at most one for any reachable marking, i.e.,*

$$(\forall M \in R(N, M_0) \wedge \forall t' \in A_i)\ [DB(M, t', t_i) \leq 1 \wedge DB(M, t', t_i^o) \leq 1].$$

It is possible to check for control safeness of a place, based solely on the net structure.

Proposition 7.9. *Let $\langle N, M_0\rangle$ be a live and bounded MG system. A place p_i is control safe if and only if $\forall t \in A_i$ there exists a cycle that contains p_i and t and this cycle is marked with a single token.*

Given a GMEC $(\vec{w}, k)$, we will assume that the places in Q_w are control safe. We have seen that if the net is safe there exists a monitor-based solution to any GMEC, even if not all transitions are controllable. The restriction that the places in Q_w be control safe has a similar purpose. It will permit a simplification of the problem, in the sense that will allow us to derive other control structures, often more efficient than a set of monitors. The different solutions are shown in Section 7.4 and Section 7.5.

The idea here is that to check whether a place $p_i \in Q_w$ may be marked by a firing sequence of uncontrollable transitions we need to check only how many firings of transitions in A_i have occurred.

Proposition 7.10. *Let $\langle N, M_0\rangle$ be a MG system, and p_i a control safe place of N. Let t_i be the input transition of p_i, t_i^o the output transition of p_i, and A_i the set of control transitions of t_i, with $n_i = |A_i|$. Then $\forall M \in R(N, M_0)$:*

1. $(\forall t \in A_i)[td(M, t, t_i^o) \leq 1]$;
2. $M(p_i) + D_M(p_i) \leq 1$;
3. $M(p_i) + D_M(p_i) = 1 \Longleftrightarrow (\forall t \in A_i)[td(M, t, t_i^o) = 1]$.

Proof: Follows from the definition of control safe place. ◇

We discuss a particular case that may arise for a given constraint $(\vec{w}, k)$. Assume that $p_{j_1} \in Q_w$ is in the control subnet of place $p_{j_2} \in Q_w$. Then p_{j_1} and p_{j_2} are in structural mutual exclusion, i.e., by the hypothesis that their are control safe, they will never be marked simultaneously. Hence $\mathcal{M}(w, k) = \mathcal{M}(\vec{w}_1, k) \cap \mathcal{M}(\vec{w}_2, k)$, where: $\vec{w}_i(p) = 0$ **if** $p = p_{j_1}$ **else** $\vec{w}_i(p) = \vec{w}(p)$, i.e., the constraint $(\vec{w}, k)$ is equivalent to the set of constraints $(W, k \cdot \vec{1}) = \{(\vec{w}_1, k), (\vec{w}_2, k)\}$. *Proof:* Trivially $\mathcal{M}(\vec{w}, k) \subseteq \mathcal{M}(\vec{w}_1, k) \cap \mathcal{M}(\vec{w}_2, k)$. To prove the reverse inclusion, assume $M \in \mathcal{M}(\vec{w}_1, k) \cap \mathcal{M}(\vec{w}_2, k)$. Then: **if** $M(p_{j_2}) > 0$ **then** $M(p_{j_1}) = 0$, and $M \in \mathcal{M}(\vec{w}_1, k) \Longrightarrow M \in \mathcal{M}(\vec{w}, k)$ **else** $M \in \mathcal{M}(\vec{w}_2, k) \Longrightarrow M \in \mathcal{M}(\vec{w}, k)$. ◇

In the next section we will assume, without loss of generality, that no place p in Q_w, p belongs to the control subnet of another place p' in Q_w.

7.4 Fully Compiled Models

In this section we present two different ways of enforcing a GMEC on MG systems with control safe places. Both solutions are fully compiled, i.e., the corresponding control structure is a net system as well.

7.4.1 Model 1: Monitor-based Controller

Definition 7.7. *Let $\langle N, M_0\rangle$ be a MG system and $(\vec{w}, k)$ be an unweighted mutual exclusion constraint, i.e., $\vec{w} \leq \vec{1}$, defined on it. We assume that $M_0 \in \mathcal{M}_c(\vec{w}, k)$. Let $Q_w = \{p_1, \ldots, p_r\}$ be a set of control safe places of N and assume that $r = |Q_w| = k + 1$. Given $p_i \in Q_w$ let $A_i = \{t_i^1, \ldots, t_i^{n_i}\}$ be the set of control transitions for t_i and t_i^o be the output transition of p_i. The* monitor *that enforces this constraint consists of a place S to be added to the original net with arcs as follows:*

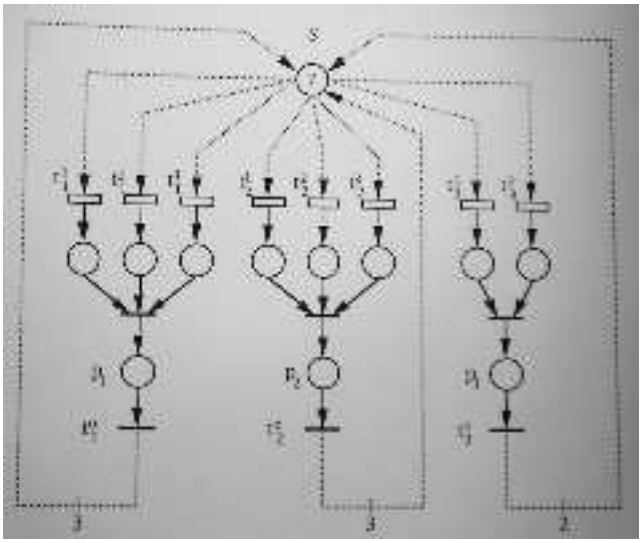


Figure 7.11: Example of monitor.

- $Pre(S,t) = 1$ **if** $t \in A_i$ **else** $Pre(S,t) = 0$*;*
- $Post(S,t) = n_i$ **if** $t = t_i^o$ **else** $Post(S,t) = 0$*.*

The initial marking will assign $s - 1$ *tokens to place* S *where*

$$s =\mid \{t \mid t \in A_i \wedge td(M_0, t, t_i^o) = 0\} \mid$$

i.e., s *counts the number of control transitions that must fire prior to the marking of all places in* Q_w*.*

The previous definition assume that no transition will be selflooped with the monitor place. E.g., a transition t will be selflooped if $\exists i, j \ni t = t_i^o \wedge t \in A_j$. In this case we need to eliminate the arcs in selfloop from the pre-incidence and post-incidence matrices.

Example 7.10. In Figure 7.11 we want to enforce the constraint

$$M(p_1) + M(p_2) + M(p_3) \leq 2$$

over the net in Figure 7.10. We have added a place S to the original system with the arcs shown in dotted lines.

It is easy to see that a monitor constructed as in Definition 7.7 enforces the maximally permissible policy that ensures that the constraint $(\vec{w}, k)$ will be satisfied. In fact the monitor prevents only transition firings that lead to all markings M such that

$$(\forall i = 1, \ldots, r)(\forall t \in A_i)[td(M, t, t_i^o) = 1],$$

which by Proposition 7.10 are the only illegal markings for unweighted constraints of this kind.

A set of constraints $(W, \vec{k})$ of this form may be enforced by adding several monitors.

Assume now $(\vec{w}, k)$, with $\vec{w} \leq \vec{1}$, is such that $|Q_w| > k + 1$. The previous construction may not be used. However the original constraint may be rewritten as a set of constraints according to the following proposition.

Proposition 7.11. Let $(\vec{w}, k)$ be a mutual exclusion constraint, with $\vec{w} \leq \vec{1}$, and $|Q_w| > k+1$. Then,

$$\mathcal{M}(\vec{w}, k) = \bigcap_{\vec{w}' \in I_{k+1}} \mathcal{M}(\vec{w}', k),$$

where $I_{k+1} = \{\vec{w} \in \{0,1\}^P \mid \vec{w}' \leq \vec{w}, |Q_{w'}| = k+1\}$.

Proof: ($\subseteq$) is trivial. Let us prove ($\supseteq$). Any marking $M \in \bigcap_{\vec{w}' \in I_{k+1}} \mathcal{M}(\vec{w}', k)$ marks at most k places in Q_w, otherwise there exists $\vec{w}'' \in I_{k+1} \ni Q_{w''} \subseteq \{p \mid M(p) > 0\}$ and $M \notin \mathcal{M}(\vec{w}'', k)$. Let $\vec{w}''' \in I_{k+1}$, be a weight vector whose support contains all the places marked by M. Clearly $M \in \mathcal{M}(\vec{w}''', k) \Longrightarrow M \in \mathcal{M}(\vec{w}, k)$. $\diamond$

The previous proposition shows that the unweighted constraint $(\vec{w}, k)$, with $\vec{w} \leq \vec{1}$ and $|Q_w| > k+1$, is equivalent to the set of constraints $(W, k \cdot \vec{1}) = \{(\vec{w}', k) \mid \vec{w}' \in I_{k+1}\}$, hence may be enforced by a set of monitors. However the problem is that there are $\begin{pmatrix} |Q_w| \\ k+1 \end{pmatrix}$ different subsets of Q_w of cardinality $k+1$. Thus in the worst case the number of monitors is exponential with respect to the cardinality of Q_w.

The monitor based construction may also be used, when the weight of the places is not unitary, explicitly rewriting the set of unweighted constraints equivalent to the single weighted constraint. We have proved in Theorem 7.1 that this is always possible for the class of nets considered here.

7.4.2 Model 2: Compiled Supervisor

In this section we consider a net supervisor, capable of enforcing a set of GMEC. We assume that the supervisor observes the execution of the unconstrained system and at any given instant provides a control pattern, i.e., specifies which controllable transitions are allowed to fire. The control pattern is implicit in the transition structure of the supervisor, in the sense that a controllable transition that belongs to the supervisor structure is enabled by the control pattern if and only if it is enabled by the marking of the supervisor net.

Definition 7.8. *Let $\langle N, M_0 \rangle$ be a MG system and $(\vec{w}, k)$ be a GMEC. We assume that $M_0 \in \mathcal{M}_c(\vec{w}, k)$. Let $Q_w = \{p_1, \ldots, p_r\}$ be a set of control safe places. Given $p_i \in Q_w$ let $A_i = \{t_i^1, \ldots, t_i^{n_i}\}$ be the set of control transitions for t_i and t_i^o be the output transition of p_i. The* compiled supervisor *that enforces this constraint is $S = (P_S, T_S, Pre_S, Post_S)$ with:*

- $P_S = \{p_0, p_1', p_1'', p_1''', p_2', \ldots, p_r', p_r'', p_r'''\}$;
- $T_S = A_1' \cup A_1'' \cup A_2' \ldots \cup A_r' \cup A_r'' \cup \{t_1^o, \ldots, t_r^o\}$ *where A_i' and A_i'' are sets of transitions synchronized with A_i;*
- Pre_S *and* $Post_S$ *are such that:*
  - $Pre_S(p_0, t) = w(p_i)$ **if** $t \in A_i''$ **else** $Pre_S(p_0, t) = 0$;
  - $Post_S(p_0, t) = w(p_i)$ **if** $t = t_i^o$ **else** $Post_S(p_0, t) = 0$;
  - $Pre_S(p_i', t) = 1$ **if** $t \in A_i'$ **else** $Pre_S(p_i', t) = 0$;
  - $Post_S(p_i', t) = n_i - 1$ **if** $t = t_i^o$ **else** $Post_S(p_i', t) = 0$;
  - $Pre_S(p_i'', t) = n_i - 1$ **if** $t \in A_i''$ **else** $Pre_S(p_i'', t) = 0$;
  - $Post_S(p_i'', t) = 1$ **if** $t \in A_i'$ **else** $Post_S(p_i'', t) = 0$;
  - $Pre_S(p_i''', t) = 1$ **if** $t = t_i^o$ **else** $Pre_S(p_i''', t) = 0$;
  - $Post_S(p_i''', t) = 1$ **if** $t \in A_i''$ **else** $Post_S(p_i''', t) = 0$.

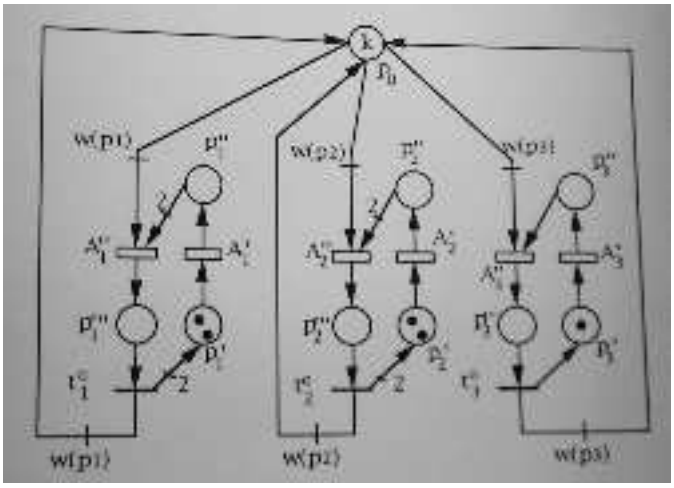


Figure 7.12: Example of compiled supervisor.

The initial marking of S is M_0^S such that, $\forall i = 1, \ldots, r$,

$$
\begin{array}{ll}
\textbf{if} & [M_0(p_i) + D_{M_0}(p_i) = 1] \\
\textbf{then} & [M_0^S(p_i') = M_0^S(p_i'') = 0 \wedge M_0^S(p_i''') = 1] \\
\textbf{else} & [M_0^S(p_i') =\mid \{t \in A_i \mid td(M_0, t, t_i^o) = 0\} \mid -1 \\
 & \wedge M_0^S(p_i'') = n_i - 1 - M_0^S(p_i') \\
 & \wedge M_0^S(p_i''') = 0],
\end{array}
$$

and $M_0^S(p_0) = k - \sum_{i=1}^{r} w(p_i) M_0^S(p_i''')$.

Example 7.11. In Figure 7.12 we show the supervisor that may be used to enforce the constraint $w(p_1)M(p_1) + w(p_2)M(p_2) + w(p_3)M(p_3) \leq k$ over the net shown in Figure 7.10.

In the example in Figure 7.12 we have represented the set of parallel transitions A_i' and A_i'' as a single transition. Whenever the system executes a transition $t \in A_i$, the corresponding transition in A_i' or A_i'' will fire. Note that the behavior is deterministic, since if a transition in A_i' is enabled, the corresponding transition in A_i'' is not, and conversely. For the computation of the control pattern, a transition in A_i is enabled by the control pattern if the corresponding transition in A_i' or in A_i'' is enabled.

In the previous definition we have assumed that:

- $(\forall i = 1, \ldots, r)\ n_i > 1$. If $n_i = 1$ we may remove the places p_i' and p_i'' and the set of transitions A_i'.
- $(\forall i \neq j)\ A_i \cap A_j = \emptyset$. If this is not the case, we need to slightly change the structure of the supervisor by merging the transitions of A_i' and A_i'' in common with A_j' and A_j''

We want to show that the supervisor constructed according to Definition 7.8 enforces the required maximally permissible policy. We note first that given a marking M of the system and a corresponding marking M^S of the supervisor we have that $(\forall i = 1, \ldots, r)[M(p_i) + D_M(p_i) = M^S(p_i''')]$. Since the place p_0 is enforcing the constraint $\sum_{i=1}^{r} w(p_i) M^S(p_i''') \leq k$ we have, by Proposition 7.8, that the supervisor enforces the required policy.

In the case of a set of constraints $(W, \bar{k}) = \{(\vec{w}_1, k_1), \ldots, (\vec{w}_m, k_m)\}$ we need to construct a supervisor for each single constraint $(\vec{w}_i, k_i)$. Should a controllable transition belong to more that one supervisor, say S_{i_1} and S_{i_2}, it will be enabled by the control pattern if and only if it is enabled on both S_{i_1} and S_{i_2}.

7.5 Partially Compiled Models

The net structure of the supervisor may be further simplified. For instance we may avoid repeating the set of transitions A_i. However we need to add additional control structure by introducing transitions with an associated interpretation. This partially destroys the possibility of analyzing the net with traditional PN techniques. The advantage, however, is that the "partially interpreted" supervisors that we discuss here may be easily modified to enforce constraints on any kind of MG system. We will not discuss these modifications.

7.5.1 Model 3: Non-Deterministic Partially Compiled Supervisor

Definition 7.9. *Let $\langle N, M_0\rangle$ be a MG system and $(\vec{w}, k)$ be a GMEC. We assume that $M_0 \in \mathcal{M}_c(\vec{w}, k)$. Let $Q_w = \{p_1, \ldots, p_r\}$ be a set of control safe places. Given $p_i \in Q_w$ let $A_i = \{t_i^1, \ldots, t_i^{n_i}\}$ be the set of control transitions for t_i and t_i^o be the output transition of p_i. The* non-deterministic partially compiled supervisor *that enforces this constraint is $S = (P_S, T_S, Pre_S, Post_S)$ with:*

- $P_S = \{p_0, p_1', p_1'', p_1''', p_2', \ldots, p_r', p_r'', p_r'''\}$;
- $T_S = A_1 \cup A_2 \cup \ldots \cup A_r \cup \{t_1^o, \ldots, t_r^o\} \cup \{\pi_1, \pi_2, \ldots, \pi_r\}$ *where each π_i is a transition to which a predicate is associated;*
- Pre_S *and* $Post_S$ *are such that:*
  - $Pre_S(p_0, t) = w(p_i)$ **if** $t = \pi_i$ **else** $Pre_S(p_0, t) = 0$;
  - $Post_S(p_0, t) = w(p_i)$ **if** $t = t_i^o$ **else** $Post_S(p_0, t) = 0$;
  - $Pre_S(p_i', t) = 1$ **if** $t = \pi_i$ **else** $Pre_S(p_i', t) = 0$;
  - $Post_S(p_i', t) = 1$ **if** $t = t_i^o$ **else** $Post_S(p_i', t) = 0$;
  - $Pre_S(p_i'', t) = 1$ **if** $t \in A_i$ **else** $Pre_S(p_i'', t) = 0$;
  - $Post_S(p_i'', t) = n_i - 1$ **if** $t = t_i^o$
    else $[Post_S(p_i'', t) = 1$ **if** $t = \pi_i$ **else** $Post_S(p_i'', t) = 0]$;
  - $Pre_S(p_i''', t) = n_i$ **if** $t = t_i^o$ **else** $Pre_S(p_i''', t) = 0$;
  - $Post_S(p_i''', t) = 1$ **if** $t \in A_i$ **else** $Post_S(p_i''', t) = 0$;

The initial marking of S is M_0^S such that, $\forall i = 1, \ldots, r$,

$$\begin{array}{ll} \textbf{if} & [M_0(p_i) + D_{M_0}(p_i) = 1] \\ \textbf{then} & [M_0^S(p_i') = M_0^S(p_i'') = 0 \wedge M_0^S(p_i''') = n_i] \\ \textbf{else} & [M_0^S(p_i') = 1 \\ & \wedge M_0^S(p_i'') = \mid \{t \in A_i \mid td(M_0, t, t_i^o) = 0\} \mid -1 \\ & \wedge M_0^S(p_i''') = n_i - M_0^S(p_i') - M_0^S(p_i'')], \end{array}$$

and $M_0^S(p_0) = k - \sum_{j \in J} w(p_i)$, where $J = \{i \mid M_0^S(p_i') = 0\}$.

The predicate associated to the transitions π_i are used to implement the control policy. The firing of transition π_i will allow place p_i to be marked.

Example 7.12. In Figure 7.13 we show the supervisor that may be used to enforce the constraint $w(p_1)M(p_1) + w(p_2)M(p_2) + w(p_3)M(p_3) \leq k$ over the net shown in Figure 7.10.

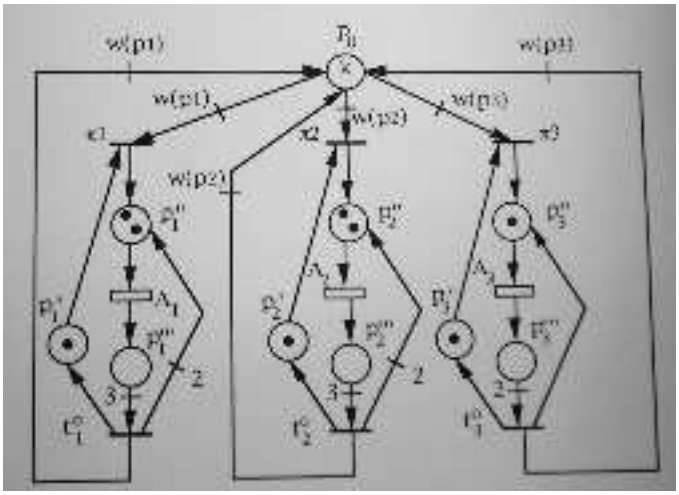


Figure 7.13: Example of non-deterministic partially compiled supervisor.

The supervisor constructed according to Definition 7.9 will permit the system to reach only legal markings. In fact, given a marking M of the system and a corresponding marking M^S of the supervisor we have that $(\forall i = 1, \ldots, r)[M(p_i) + D_M(p_i) = 1 \iff M^S(p_i''') = n_i]$. Since the place p_0 is enforcing the constraint $\sum_{i \in J} w(p_i) \leq k$, with $J = \{i \mid M^S(p_i'') + M^S(p_i''') = n_i\}$, we have, by Proposition 7.8, that the supervisor will allow only legal markings to be reached. This control policy is also maximally permissible, as shown in [Krogh 91], if two or more control transitions may fire simultaneously.

The supervisor has been called *non-deterministic* because we decide a priori which place in Q_w will be allowed to be marked.

7.5.2 Model 4: Deterministic Partially Compiled Supervisor

Definition 7.10. *Let $\langle N, M_0 \rangle$ be a MG system and $(\vec{w}, k)$ be a GMEC. We assume that $M_0 \in \mathcal{M}_c(\vec{w}, k)$. Let $Q_w = \{p_1, \ldots, p_r\}$ be a set of control safe places. Given $p_i \in Q_w$ let $A_i = \{t_i^1, \ldots, t_i^{n_i}\}$ be the set of control transitions for t_i and t_i^o be the output transition of p_i. The* deterministic partially compiled supervisor *that enforces this constraint is $S = (P_S, T_S, Pre_S, Post_S)$ with:*

- $P_S = \{p_0, p_1', p_1'', p_2', \ldots, p_r', p_r''\}$;
- $T_S = A_1 \cup A_2 \cup \ldots \cup A_r \cup \{t_1^o, \ldots, t_r^o\} \cup \{\pi_1', \pi_1'', \pi_2', \ldots, \pi_r', \pi_r''\}$ *where π_i' and π_i'' are transitions to which a predicate is associated;*
- Pre_S *and* $Post_S$ *are such that:*
  - $Pre_S(p_0, t) = 1$ **if** $t = \pi_i'$ **else** $Pre_S(p_0, t) = 0$;
  - $Post_S(p_0, t) = 1$ **if** $[t = t_i^o$ **or** $t = \pi_i'']$ **else** $Post_S(p_0, t) = 0$;
  - $Pre_S(p_i', t) = 1$ **if** $t \in A_i$ **else** $Pre_S(p_i', t) = 0$;
  - $Post_S(p_i', t) = n_i - 1$ **if** $t = t_i^o$
    else $[Post_S(p_i', t) = 1$ **if** $t = \pi_i'$ **else** $Post_S(p_i', t) = 0]$;
  - $Pre_S(p_i'', t) = n_i$ **if** $t = t_i^o$ **else** $Pre_S(p_i'', t) = 0$;
  - $Post_S(p_i'', t) = 1$ **if** $t \in A_i$ **else** $Post_S(p_i'', t) = 0$;

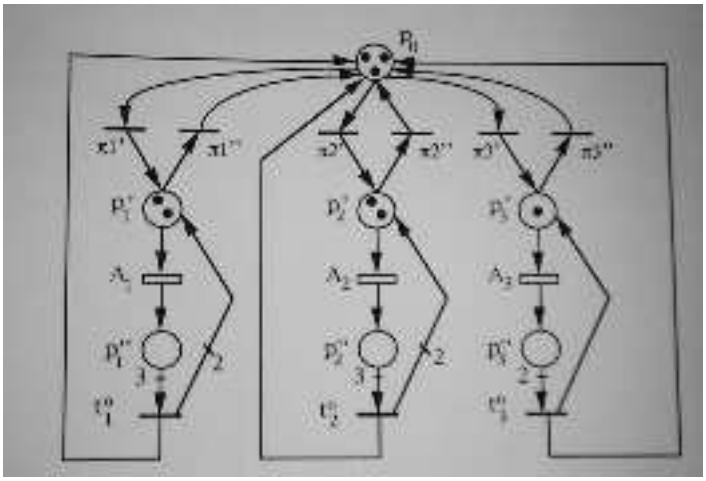


Figure 7.14: Example of deterministic partially compiled supervisor.

The initial marking of S *is* M_0^S *such that,* $\forall i = 1, \ldots, r$*,*

$$\begin{array}{ll} \textbf{if} & [M_0(p_i) + D_{M_0}(p_i) = 1] \\ \textbf{then} & [M_0^S(p_i') = 0 \wedge M_0^S(p_i'') = n_i] \\ \textbf{else} & M_0^S(p_i') = \mid \{t \in A_i \mid td(M_0, t, t_i^o) = 0\} \mid -1 \\ & \wedge M_0^S(p_i'') = n_i - M_0^S(p_i')], \end{array}$$

and $M_0^S(p_0) = r - \mid \{i \mid M_0^S(p_i'') = n_i\} \mid$.

The predicate associated to the transitions π_1' *and* π_1'' *are used to implement the control policy. Let* $J = \{i \mid M(p_i'') = n_i\}$ *and let* $v = \sum_{i \in J} w(p_i)$*. Then:*

- $\pi_i' =$ *TRUE* **if** $[M(p_i') + M(p_i'') = n_i - 1] \wedge [v + w(p_i) \leq k]$ **else** $\pi_i' =$ *FALSE;*
- $\pi_i'' =$ *TRUE* **if** $[M(p_i') + M(p_i'') = n_i] \wedge [v + w(p_i) > k]$ **else** $\pi_i'' =$ *FALSE.*

Example 7.13. In Figure 7.14 we show the supervisor that may be used to enforce the constraint $w(p_1)M(p_1) + w(p_2)M(p_2) + w(p_3)M(p_3) \leq k$ over the net shown in Figure 7.10. Note that the structure of the supervisor does not depend on the weight vector $\vec{w}$ and on the integer k, that affect only the value of the predicates π_1' and π_1''.

It is difficult to show that this supervisor is actually implementing the maximally permissible control policy because of the predicates associated to the transitions π_i' and π_i''. Note however that given a marking M of the system and a corresponding marking M^S of the supervisor, we have that $(\forall i = 1, \ldots, r)[M(p_i) + D_M(p_i) = 1 \Longleftrightarrow M^S(p_i'') = n_i]$. The transitions π_i'' will remove the token necessary to reach $M^S(p_i'') = n_i$ whenever the marking of place p_i would violate the constraint.

7.6 Comparison of the Models

The *monitor-based controller* is an extension to systems with uncontrollable transitions of the controller studied in Section 7.2 for nets with only controllable transitions. Thus all the structural properties of monitors may be used to analyze the system under control. The drawback is that it may require an exceedingly large number of monitors. However, in those cases in which it may be used efficiently, it is the simpler and most straightforward solution.

The *compiled supervisor* has the advantage of always requiring a compact structure that grows linearly with the number of places in the support of the weight vector. However, since it requires all control transitions to be represented twice, it leads to a closed-loop model which is difficult to analyze.

The *non-deterministic partially compiled supervisor* implements a policy slightly different from all other models. This policy is the same derived by Holloway and Krogh [Holloway 92a]. However our model does not require on-line computation, since the structure of the supervisor ensures that only legal markings may be reached. The predicates associated to the transitions may be used to implement different run-time policies, as suggested in [Krogh 91].

The *deterministic partially compiled supervisor* has the simpler structure but its behavior strongly depends on the predicates associated to the transitions. Thus it is at the same time the simpler model to implement and the most difficult to analyze.

7.7 Conclusions

We have presented and studied a class of specifications, called generalized mutual exclusion constraints. These specifications on a net system where all transitions are controllable may be easily enforced by a set of places called monitors. Unfortunately, we have shown that this technique is not always applicable when some of the transitions of the net are uncontrollable.

For some classes of nets, we have proved that GMEC may always be enforced by monitors, even in the presence of uncontrollable transitions. For one of these particular classes, marked graphs with control safe places, we compared a monitor-based solution of mutual exclusion problems with several supervisory based solutions.

Chapter 8

CONCLUSIONS AND FUTURE RESEARCH

8.1 Original Contributions

The thesis has discussed the use of Petri nets as discrete event models for Supervisory Control. The approach we have followed is different from previous Petri net based approaches [Holloway 90, Ushio 89]. In fact, we use Petri net models to represent not only the system under control but the supervisor as well.

The original contributions consist of four major parts.

1. Petri net languages for Supervisory Control.
2. Design of supervisors.
3. Validation of Petri net supervisors.
4. Efficient construction of control structure.

8.1.1 Petri Net Languages for Supervisory Control

We have explored the closure properties of Petri net languages under the operators used in Supervisory Control: prefix closure, and supremal controllable sublanguage. Our results show that Petri net languages are not closed under these operators.

It was known from previous work [Wonham 87] that regular languages are closed under these operators. Thus, we face a trade-off. If we restrict ourselves to the use of bounded or conservative PN models, which have a finite reachability set and generate regular languages, we are sure to have a Petri net supervisor for any given control problem. If we want to use of the full language power of Petri nets models, we may consider unbounded Petri net models, but now not all control problems may be solved by a Petri net supervisor.

We have also defined a new class of Petri net languages, called DP-closed languages and denoted $\mathcal{L}_{DP}$. The languages in this class are those L-type Petri net languages (terminal Petri net languages) whose prefix language is a P-type Petri net language (prefix Petri net language). Unlike all other known classes of Petri net languages, the class $\mathcal{L}_{DP}$ is not closed under intersection.

As a final result, we have given necessary and sufficient conditions for the existence of Petri net supervisors when the system's behavior and the legal behavior are deterministic Petri net languages.

The closed behavior $L(\mathbf{G})$ of system $\mathbf{G}$ may be restricted to a legal behavior $L \subseteq L(\mathbf{G})$ if and only if L is: a) a prefix Petri net language; b) deterministic; c) controllable. The marked behavior

$L_m(\mathbf{G})$ of system $\mathbf{G}$ may be restricted to a legal behavior $L \subseteq L_m(\mathbf{G})$ if and only if L is: a) a DP-closed Petri net language; b) $L_m(\mathbf{G})$-closed; c) controllable.

8.1.2 Supervisory Design

Based on the monolithic supervisory design, as formulated in [Wonham 88a], we have presented a design based on the concurrent composition operator. This design is well suited for Petri nets models.

The design requires two steps. In the first step, a coarse structure for a supervisor is synthesized by means of concurrent composition of different modules, representing the system and the specifications we want to enforce on the system's behavior.

This composition may be easily performed, using Petri net models, by merging transitions labeled by the same symbols. Since the composition is performed on the structure of the net, the same finite procedure may be used to compose nets with finite or infinite reachability sets. The final Petri net model represents a closed loop model of the controlled system in which it is still possible to identify the composed modules.

The generator constructed in the first step of the algorithm will be a proper supervisor if it is nonblocking and if the language it generates is controllable. These properties may be easily expressed in term of net properties, as we have seen.

In the second step, the generator constructed in the first step of the algorithm is refined to obtain a nonblocking and controllable generator. In particular, we have discussed how to refine a coarse structure for a supervisor, by introducing new arcs and possibly duplicating transitions, to avoid reaching undesirable markings.. This procedure may always be applied when the net is conservative. In this refinement, the modular structure of the net is preserved, but in some cases it may be necessary to introduce a large number of transitions.

We have also compared Petri nets and state machines models, showing the advantages of using Petri nets for the design of supervisors.

8.1.3 Validation of Petri Net Supervisors

A well known Petri net analysis technique is based on the incidence matrix analysis. We have studied how this technique may be used to validate supervisors obtained by concurrent composition of several modules.

We have defined a class of P/T net, called Elementary Composed State Machine (ECSM) nets, showing how to derive a set of linear inequalities that exactly define the set of reachable markings. ECSM nets are nets obtained by concurrent composition of state machine modules. We restrict the type of compositions considered, in order to guarantee some important properties. The two basic compositions are: a) compositions of nets along a simple path; b) compositions of nets along a set of simple paths that are looped in one of the nets. The class of Elementary Composed State Machines can model both choice and concurrent behavior.

We have also discussed the difference of our approach with respect to other classical methods based on incidence matrix analysis. We use a set of linear inequalities to define the reachability set; these inequalities are obtained by the computation of the basic traps of the state machine modules that compose an ECSM, and by the firing bounds of the composed transitions.

Important properties of the net, such as the absence of blocking states or controllability, may be studied by Integer Programming techniques. We have shown how this approach may be used to validate ECSM supervisors.

8.1.4 Efficient Construction of Control Structure

We have presented and studied a class of specifications for Petri net models, called generalized mutual exclusion constraints (GMEC), that may be used to express the concurrent use of a finite number of resources, shared among different processes.

A GMEC limits a weighted sum of tokens contained in a subset of places; all markings that do not satisfy the constraint are called forbidden. GMEC are a mild generalization of mutual exclusion constraints considered by other authors [Krogh 91, Zhou 91], due to the fact that the weights we consider are defined in $I\!N$, rather than in $\{0, 1\}$. We have presented a methodology, based on linear algebraic techniques, to compare and simplify GMEC. An equivalence notion among GMEC has been introduced and studied from the point of view of structural net theory.

A single GMEC may be easily implemented by a *monitor*, i.e., a place whose initial marking represents the available units of a resource and whose outgoing and incoming transitions represent, respectively, the acquisition and release of units of the resource.

In the framework of Supervisory Control, the complexity of enforcing a GMEC is enhanced by the presence of *uncontrollable* transitions, i.e., transitions that may be observed but not prevented from firing by a control agent. To enforce a given GMEC, it is necessary to prevent the system from reaching a superset of the forbidden markings, containing all those markings from which a forbidden one may be reached by firing a sequence of uncontrollable transitions. Unfortunately, in this case we have proven that the set of legal markings cannot *always* be represented by a linear domain in the marking space, thus there exist problems which do not have a "monitor-based" solution.

In this context, we have discussed GMEC for systems represented as marked graphs with control safe places. The goal is that of constructing a supervisor capable of enforcing the constraints. A solution to this problem has been given by Holloway and Krogh [Krogh 91]. In their approach, which may be defined as *fully interpreted*, the control policy is efficiently computed by an on-line controller as a feedback function of the marking of the system. We have studied, instead, how a net structure for the supervisor may be constructed. We have discussed and compared several solutions. Some of these models are *fully compiled*, i.e., the corresponding supervisor is represented by a P/T net, others are *partially compiled*, i.e., the corresponding supervisor is given as an interpreted net in which the firing of some transitions depends not only on the marking of the net but on the value of some boolean expressions as well.

The advantages of fully compiling the supervisor action in a net structure are:

- The computation of the control action is faster, since it does not require separate on-line computation.
- The same Petri net system execution algorithms may be used for both the original system and the supervisor.
- A closed-loop model of the system under control may be built with standard net composition constructions and analyzed for properties of interest. Moreover, structure theory of Petri nets may be used to prove, without an exhaustive state space search, that the system under control enjoys the properties.

On the other hand, partially compiled models are more flexible in the sense that some interpretations may be used to implement complex control policies, whose corresponding net structure may be exceedingly large.

8.2 Future Research

It may be possible to extend this research in several parts. We give a list of the problems that may be addressed in the future.

8.2.1 Petri Net Languages for Supervisory Control

Find a characterization of the class $\mathcal{L}_{DP}$ in terms of some pumping lemma, or other particular properties of these languages.

Characterize the class of nonregular Petri net languages that are closed under the supremal controllable sublanguage operator.

8.2.2 Supervisory Design

Determine a refinement procedure based on the coverability tree of a Petri net, rather than on its reachability tree. Since the coverability tree is always finite, this may permit the refinement of nets with infinite reachability set in all those cases in which the refined model may be given as a Petri net.

8.2.3 Validation of Petri Net Supervisors

Extend the method used to study the reachability set of ECSM to a larger class of nets. In particular, the method may be extended to other types of compositions, following the approach presented by [Koh 92] where the composition of partially overlapping paths is considered. Additionally, it may be interesting to consider the composition Macroplace/Macrotransition nets [Jungnitz 92], which are a superset of state machines.

8.2.4 Efficient Construction of Control Structure

Derive the supervisory structure for enforcing GMEC on classes of nets more general than control safe place marked graphs. Here the use of interpreted nets should prove a key factor in deriving a simple supervisory structure.

Consider classes of specification different from GMEC, e.g., *sequencing specifications* [Holloway 92b], and derive the corresponding supervisor using a structural based approach.

Bibliography

[Aveyard 74] R.L. Aveyard, "A Boolean Model for a Class of Discrete Event Systems," *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. SMC-4, No. 3, pp. 249–258, May, 1974.

[Avrunin 91] G.S. Avrunin, U.A. Buy, J.C. Corbett, L.K. Dillon, J.C. Wileden, "Automated Analysis of Concurrent Systems with the Constrained Expression Toolset," *IEEE Trans. Software Engineering*, Vol. 17, No. 11, pp. 1204–1222, November, 1991.

[Baker 72] H. Baker, Jr., "Petri Nets and Languages," *Computation Structures Group Memo 68*, Project MAC, Massachusetts Institute of Technology (Cambridge, Massachusetts), May, 1972.

[Baker 73] H. Baker, Jr., "Equivalence Problems of Petri Nets," *Master's thesis*, Dept. of Electrical Engineering, Massachusetts Institute of Technology (Cambridge, Massachusetts), May, 1973.

[Banaszak 90] Z.A. Banaszak, B.H. Krogh, "Deadlock Avoidance in Flexible Manufacturing Systems with Concurrently Competing Process Flows," *IEEE Trans. on Robotics and Automation*, Vol. RA-6, No. 6, pp. 724–734, December, 1990.

[Beck 86] C.L Beck, B.H. Krogh, "Models for Simulation and Discrete Control of Manufacturing Systems," *IEEE Int. Conf. Robotics and Automation*, pp. 305–310, April, 1986.

[Berthelot 86] G. Berthelot, "Checking Properties of Nets Using Transformations," *Advances in Petri Nets 1985*, G. Rosenberg (ed.), Lecture Notes in Computer Sciences, Vol. 85, pp. 19–40, Springer-Verlag, 1986.

[Berthelot 87] G. Berthelot, "Transformations and Decompositions of Nets," *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986*, W. Brauer, W. Reisig and G. Rosenberg (eds.), Lecture Notes in Computer Sciences, Vol. 254-I, pp. 359–376, Springer-Verlag, 1987.

[Berthomieu 87] B. Berthomieu, "Methods for Carrying Proofs on Petri Nets Using their Structural Properties," *Technical Report*, Laboratoire d'Automatique et d'Analyse des Systèmes du CNR (Toulouse, France), January, 1987.

[Best 86] E. Best, C. Fernández, "Notation and Terminology on Petri Net Theory," *Arbeitspapiere der Gesellschaft für Math. und Datenverarbeitung*, No. 195, 1986.

[Best 91] E. Best, "Design Methods Based on Nets: Esprit Basic Research Action DEMON" *Advances in Petri nets, 1990*, G. Rozenberg (ed.), pp. 487–506, Springer-Verlag, 1991.

[Brave 90] Y. Brave, M. Heymann, "Stabilization of Discrete-Event Processes," *Int. J. Control*, Vol. 51, No. 50, pp. 1101–1117, 1990.

[Campos 91] J. Campos, G. Chiola, M. Silva, "Ergodicity and Throughput Bounds of Petri Nets with Unique Consistent Firing Count Vector," *IEEE Trans. on Software Engineering*, Vol. SE-17, No. 2, pp. 117–125, February, 1991.

[Cieslak 91] R. Cieslak, C. Desclaux, A.S. Fawaz, P. Varaiya, "Supervisory Control of Discrete-Event Processes with Partial Observations," *IEEE Trans. on Automatic Control*, Vol. AC-33, No. 3, pp. 249–260, March, 1991.

[Chen 91] E. Chen, S. Lafortune, "Dealing with Blocking in Supervisory Control of Discrete Event Systems," to appear in *IEEE Trans. on Automatic Control*, 1991.

[Colom 89] J.M. Colom, "Análisis Estructural de Redes de Petri, Programación Lineal y Geometria Convexa," *Tesis Doctoral*, Universidad de Zaragoza (Zaragoza, Spain), 1989.

[Crockett 87] D. Crockett, A. Desrochers, F. DiCesare, T. Ward, "Implementation of a Petri Net Controller for a Machining Workstation," *Proc. IEEE Int. Conf. on Robotics and Automation* (Raleigh, North Carolina), pp. 1861–1867, April, 1987.

[Datta 84] A.K. Datta, S. Gosh, "Synthesis of a Class of Deadlock-Free Petri Nets," *Jour. of the Association for Computing Machinery*, Vol. 31, No. 3, pp. 486-506, July, 1984.

[Datta 86] A.K. Datta, S. Gosh, "Modular Synthesis of Deadlock-Free Control Structures," *Foundation of Software Technology and Theoretical Computer Science*, Vol. 241, G. Goos and J. Hartmanis (eds.), Springer-Verlag, pp. 288–318,1986.

[De Cindio 82] F. De Cindio, G. De Michelis, L. Pomello, C. Simone, "Superposed Automata Nets," *Application and Theory of Petri Nets*, C. Girault and W. Reisig (eds.), Informatik-Fachberichte, Vol. 52, pp. 269–279, Springer-Verlag, 1982.

[De Cindio 83] F. De Cindio, G. De Michelis, L. Pomello, C. Simone, "Milner's Communicating Systems and Petri Nets," *Application and Theory of Petri Nets*, A. Pagnoni and G. Rozenberg (eds.), Informatik-Fachberichte, Vol. 66, pp. 40–59, Springer-Verlag, 1983.

[DiCesare 91] F. DiCesare, A. Fanni, A. Giua, "Controllo dei sistemi ad eventi discreti mediante reti di Petri," *Ricerca Operativa*, No. 58, pp. 2–47, 1991.

[Giua 90a] A. Giua, "Petri Net Languages for the Control of Discrete Event Systems," *Master's Thesis*, Dept. Electrical Computer and Systems Engineering, Rensselaer Polytechnic Institute (Troy, New York), 1990.

[Giua 90b] A. Giua, F. DiCesare, "Easy Synchronized Petri Nets as Discrete Event Models," *Proc. 29th IEEE Int. Conf. on Decision and Control* (Honolulu, Hawaii), pp. 2839–2844, December, 1990.

[Giua 91] A. Giua, F. DiCesare, "Supervisory Design Using Petri Nets," *Proc. 30th IEEE Int. Conf. on Decision and Control* (Brighton, England), pp. 385–390, December, 1991.

[Giua 92a] A. Giua, F. DiCesare, "GRAFCET and Petri Nets in Manufacturing," will appear in *Programming Environments for Computer Integrated Manufacturing*, W.A. Gruver and J.C. Boudreaux (Eds.), Springer-Verlag, 1992.

[Giua 92b] A. Giua, F. DiCesare, M. Silva, "Generalized Mutual Exclusion Constraints for Petri Nets with Uncontrollable Transitions," will appear in *Proc. 1992 IEEE Int. Conf. on Systems, Man, and Cybernetics* (Chicago, Illinois), October, 1992.

[Giua 92c] A. Giua, F. DiCesare, "On the Existence of Petri Nets Supervisors," submitted to *31th IEEE Int. Conf. on Decision and Control* (Tucson, Arizona), December, 1992.

[Giua 92d] A. Giua, F. DiCesare, "Petri Net Incidence Matrix Analysis for Supervisory Control," *Working Paper*, Rensselaer Polytechnic Institute (Troy, New York), 1992.

[Golaszewski 88] C.H. Golaszewski, P.J. Ramadge, "Mutual Exclusion Problems for Discrete Event Systems with Shared Events," *Proc. IEEE 27th Int. Conf. on Decision and Control* (Austin, Texas), pp. 234–239, December, 1988.

[Hack 72] M. Hack, "Analysis of Production Schemata by Petri Nets," *Technical Report 94*, Project MAC, Massachusetts Institute of Technology (Cambridge, Massachusetts), February, 1972.

[Hack 74] M. Hack, "Extended State Machine Allocatable Nets, an Extension of Free Choice Petri Nets Results," *Computation Structures Group Memo 78-1*, Project MAC, Massachusetts Institute of Technology (Cambridge, Massachusetts), 1974.

[Hack 75a] M. Hack, "Petri Nets Languages," *Computation Structures Group Memo 124*, Project MAC, Massachusetts Institute of Technology (Cambridge, Massachusetts), June, 1975.

[Hack 75b] M. Hack, "Decidability Questions for Petri Nets," *Ph.D. dissertation*, Dept. of Electrical Engineering, Massachusetts Institute of Technology (Cambridge, Massachusetts), December, 1975.

[Hailpern 83] B.T. Hailpern, S.S. Owicki, "Modular Verification of Computer Communication Protocols," *IEEE Trans. on Communication*, Vol. COM-31, No. 1, pp. 56–68, January, 1983.

[Heymann 90] M. Heymann, "Concurrency and Discrete Event Control," *IEEE Control Systems Magazine*, pp. 103–112, June, 1990.

[Holloway 90] L.E. Holloway, B.H. Krogh, "Synthesis of Feedback Control Logic for a Class of Controlled Petri Nets," *IEEE Trans. on Automatic Control*, Vol. AC-35, No. 5, pp. 514–523, May, 1990.

[Holloway 92a] L.E. Holloway, B.H. Krogh, "On Closed-loop Liveness of Discrete Event Systems Under Maximally Permissible Control," *IEEE Trans. on Automatic Control*, Vol. AC-37, No. 5, pp. 692–697, May, 1992.

[Holloway 92b] L.E. Holloway, F. Hossain, "Feedback Control for Sequencing Specifications in Controlled Petri Nets," *3rd Int. Conf. on Computer Integrated Manufacturing* (Troy, New York), pp. 242–251, May, 1992.

[Hoare 85] C. A. R. Hoare, "Communicating Sequential Processes," Prentice-Hall, 1985.

[Ichikawa 85] A. Ichikawa, K. Yokoyama, S. Kurogy, "Reachability and Control of Discrete Event Systems Represented by Conflict-Free Petri Net," *Proc. IEEE Int. Symp. Circuits and Systems* (Kyoto, Japan), pp. 487–490, May, 1985.

[Ichikawa 88a] A. Ichikawa, K. Hiraishi, "Analysis and Control of Discrete Event Systems Represented by Petri Nets," *Discrete Events Systems: Models and Applications*, P. Varaiya and A.B. Kurzhanski (eds.), Lecture Notes in Control and Information Sciences, Vol. 103, pp. 115–134, Springer-Verlag, 1988.

[Ichikawa 88b] A. Ichikawa, K. Hiraishi, "A Class of Petri Nets That a Necessary and Sufficient Condition for Reachability is Obtainable," *Trans. Society of Instrument and Control Engineers, SICE* (in Japanese), Vol. 24, No. 6, 1988.

[Inan 88] K. Inan, P. Varaiya, "Finitely Recursive Process Models for Discrete Event Systems," *IEEE Trans. on Automatic Control*, Vol. AC-33, No. 7, pp. 626–639, July, 1988.

[Johnen 87] C. Johnen, "Analyse Algorithmique des Réseaux de Petri: Verification d'Espace d'Accueil, Systemès de Réécriture," *Thèse Doctoral*, Université Paris–Sud (Paris, France), 1987.

[Jantzen 87] M. Jantzen, "Language Theory of Petri Nets," *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986*, W. Brauer, W. Reisig and G. Rosenberg (eds.), Lecture Notes in Computer Sciences, Vol. 254-I, pp. 397–412, Springer-Verlag, 1987.

[Jungnitz 92] H. Jungnitz, "Approximation Methods for Stochastic Petri Nets," *Ph.D. Thesis*, Rensselaer Polytechnic Institute (Troy, New York), May, 1992.

[Kasturia 88] E. Kasturia, F. DiCesare, A. Desrochers, "Real Time Control of Multilevel Manufacturing Systems Using Colored Petri Nets," *Proc. IEEE Int. Conf. on Robotics and Automation* (Philadelphia, Pennsylvania), pp. 1114–1119, April, 1988.

[Koh 92] I. Koh, "A Transformation Theory for Petri Nets and Their Applications to Manufacturing Automation," *Ph.D. Thesis*, Rensselaer Polytechnic Institute (Troy, New York), December, 1991.

[Krogh 86] B.H. Krogh, C.L Beck, " Synthesis of Place/Transition Nets for the Simulation and Control of Manufacturing Systems," *Proc. 4th IFAC/IFORS Symp. on Large Scale Systems* (Zurich, Switzerland), August, 1986.

[Krogh 87] B.H. Krogh, "Controlled Petri Nets and Maximally Permissible Feedback Logic," *Proc. 25th Allerton Conf. on Communications, Control, and Computing* (Urbana-Champaign, Illinois), pp. 317–326, September, 1987.

[Krogh 91] B.H. Krogh, L.E. Holloway, "Synthesis of Feedback Control Logic for Discrete Manufacturing Systems," *Automatica*, Vol. 27, No. 4, pp. 641–651, July-August, 1991.

[Lafortune 90a] S. Lafortune, E. Chen, "The Infimal Closed Controllable Superlanguage and Its Application in Supervisory Control," *IEEE Trans. on Automatic Control*, Vol. AC-35, No. 4, pp. 398–405, April, 1990.

[Lafortune 90b] S. Lafortune, H. Yoo, "Some Results on Petri net Languages," *IEEE Trans. on Automatic Control*, Vol. AC-35, No. 4, pp. 482–485, April, 1990.

[Lafortune 91] S. Lafortune, F. Lin, "On Tolerable and Desirable Behaviors in Supervisory Control of Discrete Event Systems," *J. of Discrete Event Dynamic Systems*, January, 1991.

[Li 88] Y. Li, W.M. Wonham, "Controllability and Observability in the State-Feedback Control of Discrete-Event Systems," *Proc. 27th IEEE Conf. Decision and Control* (Austin, TX), pp. 203–208, December, 1988.

[Lin 88] F. Lin, W.M. Wonham, "Decentralized Control and Coordination of Discrete-Event Systems," *Proc. 27th IEEE Conf. Decision and Control* (Austin, TX), pp. 1125–1130, December, 1988.

[Lin 90] F. Lin, W.M. Wonham, "Decentralized Control and Coordination of Discrete-Event Systems with Partial Observation," *IEEE Trans. on Automatic Control*, Vol. AC-35, No. 12, pp. 1330–1337, December, 1990.

[Milne 79] G. Milne, R. Milner, "Concurrent Processes and Their Syntax," *Jour. ACM*, Vol. 26, No. 2, pp. 302–321, April, 1979.

[Milner 80] R. Milner, "A Calculus of Communicating Systems," Lecture Notes in Computer Science 92, Springer-Verlag, 1980.

[Murata 77] T. Murata, "Circuit Theoretic Analysis and Synthesis of Marked Graphs," *IEEE Trans. on Circuits and Systems*, Vol. CAS-24, No. 7, pp. 400–405, July, 1977.

[Murata 86] T. Murata, N. Komoda, K. Matsumoto, K. Haruna, "A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Application in Factory Automation," *IEEE Trans. on Industrial Electronics*, Vol. IE-33, No. 1, pp. 1–8, February, 1986.

[Murata 89] T. Murata, " Petri Nets: Properties, Analysis and Applications," *Proceedings IEEE*, Vol. PROC-77, No. 4, pp. 541–580, April, 1989.

[Narahari 85] Y. Narahari, N. Viswanadham, "A Petri Net Approach to the Modeling and Analysis of Flexible Manufacturing Systems," *Annals of Operations Research*, Vol. 3, pp. 449–472, 1985.

[Ostroff 90a] J.S. Ostroff, W.M. Wonham, "A Framework for Real-Time Discrete Event Control," *IEEE Trans. on Automatic Control*, Vol. AC-35, No. 4, pp. 386–397, April, 1990.

[Ostroff 90b] J.S. Ostroff, "A logic for Real-Time Discrete Event Processes," *IEEE Control Systems Magazine*, pp. 95–102, June, 1990.

[Özveren 90] C.M. Özveren, A.S. Willsky, "Observability of Discrete Event Dynamic Systems," *IEEE Trans. on Automatic Control*, Vol. AC-35, No. 7, pp. 797–806, July, 1990.

[Parigot 86] M. Parigot, E. Peltz, "A Logical Formalism for the Study of the Finite Behavior of Petri Nets," *Advances in Petri Nets 1985*, Lecture Notes in Computer Science 222, G. Rozenberg (ed.), Springer-Verlag, pp. 346–361, 1986.

[Peltz 86] E. Peltz, "Infinitary Languages of Petri Nets and Logical Sentences," *Proc. 7th European Workshop on Application and Theory of Petri Nets* (Oxford, England), Sheffield City Polytech., pp. 224–237, 1986.

[Peterson 81] J.L. Peterson, "Petri Net Theory and the Modeling of Systems," Prentice-Hall, 1981.

[Pnueli 79] A. Pnueli, "The Temporal Semantics of Concurrent Programs," *Proc. Int. Symposium on Semantics of Concurrent Computation* (Evian, France), Lecture Notes in Computer Science 70, Springer-Verlag, pp. 1–20, 1979.

[Ramadge 83] P.J. Ramadge, "Control and Supervision of Discrete Event Processes," *Ph.D. Thesis*, Dept. Electrical Engineering, Univ. Toronto (Toronto, Ontario), 1983.

[Ramadge 86] P.J. Ramadge, W.M. Wonham, "Modular Supervisory Control of Discrete-Event Systems," *Proc. 7th Int. Conf. Analysis and Optimization of Systems* (Antibes, France), pp. 202–214, June, 1986.

[Ramadge 87] P.J. Ramadge, W.M. Wonham, "Supervisory Control of a Class of Discrete-Event Processes," *SIAM Jour. Control and Optimization*, Vol. 25, No. 1, pp. 206–230, January, 1987.

[Ramadge 88] P.J. Ramadge, "Supervisory Control of Discrete Event Systems: A Survey and Some New Results," *Discrete Event Systems: Models and Applications*, Varaiya and Kurzhanski (eds.), Lecture Notes in Control and Information Sciences, N. 103, pp. 69-80, Springer-Verlag, 1988.

[Ramadge 89a] P.J. Ramadge, "Some Tractable Supervisory Control Problem for Discrete-Event Systems Modeled by Büchi Automata," *IEEE Trans. on Automatic Control*, Vol. AC-34, No. 1, pp. 10–19, January, 1989.

[Ramadge 89b] P.J. Ramadge, W.M. Wonham, "The Control of Discrete Event Systems," *Proceedings IEEE*, Vol. PROC-77, No. 1, pp. 81–98, January, 1989.

[Reisig 85] W. Reisig, "Petri Nets: An Introduction," *EATCS Monographs on Theoretical Computer Science*, Vol. 4, W. Brauer, G. Rozenberg and A. Salomaa (eds.), Springer-Verlag, 1885.

[Silva 80] M. Silva, "Simplification des rèseaux de Petri par elimination des places implicites," *Digital Processes*, Vol. 6, pp. 245–256, 1980.

[Silva 83] M. Silva, S. Velilla, "Programmable Logic Controllers and Petri nets: a comparative study," *IFAC Conference on Software for Computer Control*, E.A. Puente and Ferrate (Eds.), pp. 83–88, Pergamon Press, 1983.

[Silva 85] M. Silva, *Las redes de Petri en la Automatica y la Informatica*, Ed. AC, Madrid, Spain, 1985.

[Silva 89a] M. Silva, J.M. Colom, "On the Computation of Structural Synchronic Invariants in P/T Nets" *Advances in Petri nets, 1988*, G. Rozenberg ed., pp. 386–417, Springer-Verlag, 1989.

[Silva 89b] M. Silva, "Logic Controllers," *Proc. IFAC Int. Symp. on Low Cost Automation* (Milan, Italy), pp. 157–165 bis, November, 1989.

[Silva 92] M. Silva, J.M. Colom, J. Campos, "Linear Algebraic Techniques for the Analysis of Petri Nets," *Proc. Int. Symp. on Mathematical Theory of Networks and Systems*, MITA Press (Tokyo, Japan), (to appear) 1992.

[Sreenivas 92] R.S. Sreenivas, B.H. Krogh, "On Petri Net Models of Infinite State Supervisors," *IEEE Trans. on Automatic Control*, Vol. AC-37, No. 2, pp. 274–277, February, 1992.

[Tadmor 89] G. Tadmor, O. Maimon, "Control of Large Discrete Event Systems: Constructive Algorithms," *IEEE Trans. on Automatic Control*, Vol. AC-34, No. 11, pp. 1164–1168, November, 1989.

[Tsitsiklis 87] J.N. Tsitsiklis, "On the Control of Discrete-Event Dynamical Systems," *Proc. 26th IEEE Int. Conf. Decision and Control* (Los Angeles, California), pp. 419–422, December, 1987.

[Ushio 88] T. Ushio, R. Matsumoto, "State Feedback and Modular Control Synthesis in Controlled Petri Nets," *Proc. 27th Int. Conf. Decision and Control* (Austin, Texas), pp. 1502–1507, December, 1988.

[Ushio 89] T. Ushio, "On the Controllability of Controlled Petri Nets," *Control-Theory and Advanced Technology*, Vol. 5, No. 3, pp. 265–275, September, 1989.

[Ushio 90] T. Ushio, "On The Existence of Finite State Supervisors in Discrete-Event Systems," *Proc. 29th IEEE Int. Conf. Decision and Control* (Honolulu, Hawaii), pp. 2857–2860, December, 1990.

[Viswanadham 90] N. Viswanadham, Y. Narahari, T.J. Johnson, "Deadlock Prevention and Deadlock Avoidance in Flexible Manufacturing Systems Using Petri Net Models," *IEEE Trans. on Robotics and Automation*, Vol. RA-6, No. 6, pp. 713–723, December, 1990.

[Wonham 87] W.M. Wonham, P.J. Ramadge, "On the Supremal Controllable Sublanguage of a Given Language," *SIAM Jour. Control and Optimization*, Vol. 25, No. 3, pp. 637–659, May, 1987.

[Wonham 88a] W.M. Wonham, "A Control Theory for Discrete-Event Systems," *Advanced Computing Concepts and Techniques in Control Engineering*, M.J. Denham and A.J. Laub (eds.), Springer-Verlag, pp. 129–169, 1988.

[Wonham 88b] W.M. Wonham, P.J. Ramadge, "Modular Supervisory Control of Discrete-Event Systems," *Math. Control Signals Systems*, Vol. 1, No. 1, pp. 13–30, 1988.

[Zhou 90] M.C. Zhou, "A Theory for the Synthesis and Augmentation of Petri Nets in Automation," *Ph.D. Thesis*, Rensselaer Polytechnic Institute (Troy, New York), May, 1990.

[Zhou 91] M.C. Zhou, F. DiCesare, "Parallel and Sequential Mutual Exclusions for Petri Net Modeling of Manufacturing Systems with Shared Resources," *IEEE Trans. on Robotics and Automation*, Vol. RA-7, No. 4, pp. 515–527, August, 1991.

Appendix A

PETRI NET LANGUAGES

A.1 Petri Net Generators

A *labeled Petri net* (or *Petri net generator*) is a 4-tuple $\mathbf{G} = (N, \ell, M_0, F)$ where [Jantzen 87, Peterson 81]:

- $N = (P, T, Pre, Post)$ is a Petri net structure;
- $\ell : T \rightarrow \Sigma \cup \{\lambda\}$ is a *labeling function* that assigns to each transition a label from the alphabet of events Σ or assigns the empty string as a label;
- M_0 is an initial marking;
- F is a finite set of final markings.

We will use Petri net generators to represent *discrete event systems*. Thus we will use the word *system* to refer to a Petri net generator.

Three different type of *labeling functions* are usually considered.

- In a *free-labeled* PN all transitions are labeled distinctly and none is labeled λ, i.e., $(\forall t, t' \in T)\ [t \neq t' \Longrightarrow \ell(t) \neq \ell(t')]$ and $(\forall t \in T)\ [\ell(t) \neq \lambda]$.
- In a *λ-free labeled* PN no transition is labeled λ.
- In a *arbitrary labeled* PN no restriction is posed on ℓ.

The labeling function may be extended to a function $\ell : T^* \rightarrow \Sigma^*$ defining: $\ell(\lambda) = \lambda$ and $(\forall t \in T, \forall \sigma \in T^*)\ [\ell(\sigma t) = \ell(\sigma)\ell(t)]$.

Four languages are associated with $\mathbf{G}$ depending on the different notions of terminal strings.

- The *L-type* or *terminal language*[1] is defined as the set of strings generated by firing sequences that reach a final marking, i.e.,

$$L_L(\mathbf{G}) = \{\ell(\sigma) \mid M_0\ [\sigma\rangle\ M_f \in F\}.$$

- The *G-type* or *covering language* or *weak language* is defined as the set of strings generated by firing sequences that reach a marking M covering a final marking, i.e.,

$$L_G(\mathbf{G}) = \{\ell(\sigma) \mid M_0\ [\sigma\rangle\ M \geq M_f \in F\}.$$

  The *D-type* or *deadlock language* is defined as the set of strings generated by firing sequences that reach a marking at which no transition is enabled, i.e.,

$$L_D(\mathbf{G}) = \{\ell(\sigma) \mid M_0\ [\sigma\rangle\ M \wedge (\forall t \in T) \neg M\ [t\rangle\}.$$

[1] This language is called marked behavior in the framework of Supervisory Control and is denoted $L_m(\mathbf{G})$.

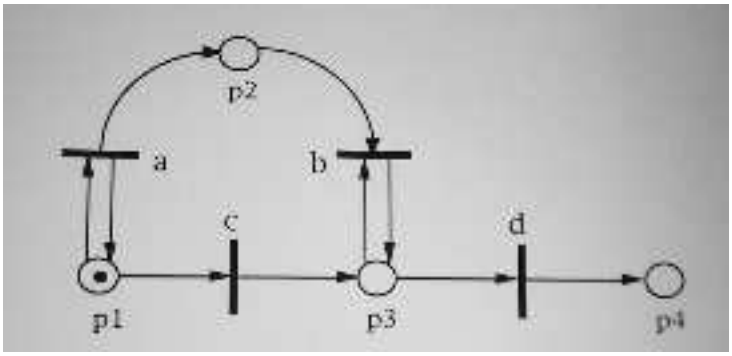


Figure A.1: Free-labeled system $\mathbf{G}$ in Example A.1.

- The *P-type* or *prefix language*[2] is defined as the set of strings generated by any firing sequence, i.e.,
$$L_P(\mathbf{G}) = \{\ell(\sigma) \mid M_0\ [\sigma\rangle\ \}.$$

Example A.1. The system $\mathbf{G}$ in Figure A.1 is free-labeled. The initial marking, also shown in the figure, is $M_0 = (1000)^T$. The set of final markings is $F = \{(0010)^T\}$. The languages of this system are:

$$L_L(\mathbf{G}) = \{a^m c b^m \mid m \geq 0\};$$
$$L_G(\mathbf{G}) = \{a^m c b^n \mid m \geq n \geq 0\};$$
$$L_D(\mathbf{G}) = \{a^m c b^n d \mid m \geq n \geq 0\};$$
$$L_P(\mathbf{G}) = \{a^m \mid m \geq 0\} \cup \{a^m c b^n \mid m \geq n \geq 0\} \cup \{a^m c b^n d \mid m \geq n \geq 0\}.$$

A.2 Classes of Petri Net Languages

The classes of Petri net languages are denoted as follows.

- $\mathcal{L}^f$ (resp. $\mathcal{G}^f$, $\mathcal{D}^f$, $\mathcal{P}^f$) denotes the class of terminal (resp. covering, deadlock, prefix) languages generated by free-labeled PN generators.
- $\mathcal{L}$ (resp. $\mathcal{G}$, $\mathcal{D}$, $\mathcal{P}$) denotes the class of terminal (resp. covering, deadlock, prefix) languages generated by λ-free labeled PN generators.
- $\mathcal{L}^\lambda$ (resp. $\mathcal{G}^\lambda$, $\mathcal{D}^\lambda$, $\mathcal{P}^\lambda$) denotes the class of terminal (resp. covering, deadlock, prefix) languages generated by arbitrary labeled PN generators.

The following table shows the relationship among these classes. Here $\rightarrow$ represents set inclusion $\supseteq$.

Some of these relations are easily proved. As an example, any *P-type* language of a generator $\mathbf{G}$ may also be obtained as a *G-type* language defining as a set of final markings $F = \{\vec{0}\}$.

[2]This language is called closed behavior in the framework of Supervisory Control and is denoted $L(\mathbf{G})$.

$$
\begin{array}{ccccc}
\mathcal{L}^{\lambda} & \rightarrow & \mathcal{L} & \rightarrow & \mathcal{L}^{f} \\
\downarrow\uparrow & & \downarrow\uparrow & & \\
\mathcal{D}^{\lambda} & \rightarrow & \mathcal{D} & \rightarrow & \mathcal{D}^{f} \\
\downarrow & & \downarrow & & \\
\mathcal{G}^{\lambda} & \rightarrow & \mathcal{G} & \rightarrow & \mathcal{G}^{f} \\
\downarrow & & \downarrow & & \downarrow \\
\mathcal{P}^{\lambda} & \rightarrow & \mathcal{P} & \rightarrow & \mathcal{P}^{f}
\end{array}
$$

Table A.1: Known relations among classes of Petri net languages. An arc $\rightarrow$ represents the set inclusion.

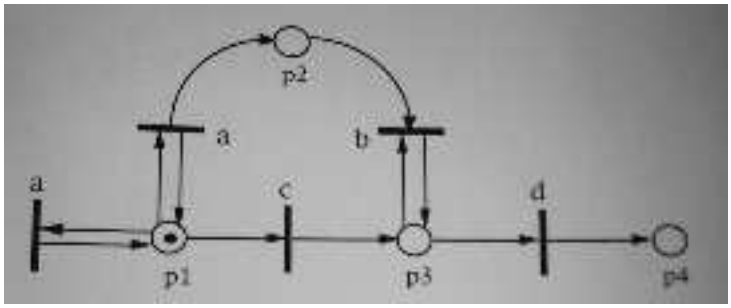


Figure A.2: Nondeterministic system $\mathbf{G}'$ in Example A.2.

A.3 Deterministic Languages

A PN generator $\mathbf{G}$ is *deterministic* if the labeling function ℓ and the behavior of $\mathbf{G}$ are such that $\forall t, t' \in T$, with $t \neq t'$ and $\forall M \in R(N, M_0),\ \ M\,[t\rangle \wedge M\,[t'\rangle \Longrightarrow [\ell(t) \neq \ell(t'), \ell(t) \neq \lambda, \ell(t') \neq \lambda]$.

This means that the knowledge of the string generated by the system from the initial marking is sufficient to determine the actual marking of the system.

We can then define twelve new classes of PN languages. For each of the previously defined classes we will denote with the subscript d the corresponding deterministic subclass. Thus $\mathcal{L}_d$ will denote the subclass of $\mathcal{L}$ generated by *deterministic* Petri nets, etc.

The relations among the subclasses of deterministic languages are different from those reported in Table A.1 for the general classes. Table A.1 shows that $\mathcal{P} \subset \mathcal{L}$. However, the next example shows that $\mathcal{P}_d \not\subset \mathcal{L}_d$.

Example A.2. Consider again the deterministic system $\mathbf{G}$ in Figure A.1 with set of final markings $F = \{(0001)^T\}$. Here $L_P(\mathbf{G}) = \{a^m \mid m \geq 0\} \cup \{a^m c b^n \mid m \geq n \geq 0\} \cup \{a^m c b^n d \mid m \geq n \geq 0\}$ is a deterministic $P - type$ PN language. We want to construct a new system $\mathbf{G}'$ such that $L_L(\mathbf{G}') = L_P(\mathbf{G})$. A possible solution is the system in Figure A.2 with set of final states $F = \{(1000)^T, (0010)^T, (0001)^T\}$. However $\mathbf{G}'$ is nondeterministic, since the two transitions labeled a are both enabled at the initial marking. It is intuitively clear that no deterministic Petri net may generate $L_P(\mathbf{G})$ as a terminal language if F is finite.

In the framework of Supervisory Control, we will consider the $L - type$ and $P - type$ language generated by deterministic systems. A new subclass of $\mathcal{L}$, called *deterministic P-closed* is discussed in Chapter 4. This class, denoted $\mathcal{L}_{DP}$, will play a major role in deriving necessary and sufficient conditions for the existence of supervisors represented as deterministic Petri net generators.

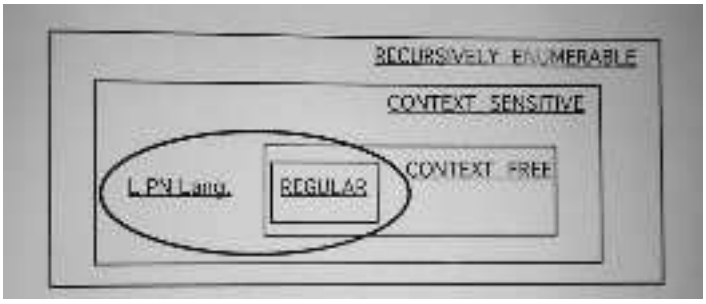


Figure A.3: Relations among the class $\mathcal{L}$ and other classes of formal languages.

A.4 Closure Properties and Relations with Other Classes

Parigot and Peltz have defined PN languages as regular languages with the additional capability of determining if a string of parenthesis is well formed. If we consider the class $\mathcal{L}$ of PN languages, it is possible to prove that $\mathcal{L}$ is a strict superset of regular languages and a strict subset of context-sensitive languages. In the diagram in Figure A.3, the set $\mathcal{L}$ is denoted "L PN languages".

It is possible to prove that $\mathcal{L}$ and the class of context-free languages are not commensurable. An example of a language in $\mathcal{L}$ that is not context-free: $L = \{a^m b^m c^m \mid m \geq 0\}$. An example of a language that is context-free but is not in $\mathcal{L}$: $L = \{ww^R \mid w \in \Sigma^*\}$[3] if $|\Sigma| > 1$.

The class $\mathcal{L}$ is closed under: concatenation, union, intersection, concurrent composition. The class $\mathcal{L}$ is not closed under: Kleene star, prefix closure[4].

A.5 Concurrent Composition and System Structure

In this section we will review the counterpart of language operators on the structure of a PN generator. We will consider only the concurrent composition operator, since other operators of interest, such as the intersection and shuffle operators may be considered as a special case of concurrent composition.

Let $\mathbf{G}_1 = (N_1, \ell_1, M_{0,1}, F_1)$ and $\mathbf{G_2} = (N_2, \ell_2, M_{0,2}, F_2)$ be two PN generators. Their *concurrent composition*, denoted also $\mathbf{G} = \mathbf{G}_1 \parallel \mathbf{G}_2$, is the system $\mathbf{G} = (N, \ell, M_0, F)$ that generates $L_L(\mathbf{G}) = L_L(\mathbf{G}_1) \parallel L_L(\mathbf{G}_2)$ and $L_P(\mathbf{G}) = L_P(\mathbf{G}_1) \parallel L_P(\mathbf{G}_2)$.

The structure of $\mathbf{G}$ may be determined as follows. Let P_i, T_i and Σ_i $(i = 1, 2)$ be the place set, transition set, and the alphabet of $\mathbf{G}_i$.

- The place set P of N is the union of the place sets of N_1 and N_2, i.e., $P = P_1 \cup P_2$.
- The transition set T of N and the corresponding labels are computed as follows.
  - Let $a \in \Sigma_1 \setminus \Sigma_2$ ($a \in \Sigma_2 \setminus \Sigma_1$) be the label of a transition $t \in T_1$ ($t \in T_2$). Then a labels a transition in T with the same input and output bag of t.
  - Let $a \in \Sigma_1 \cap \Sigma_2$ be labeling m_1 transitions in T_1 and m_2 transitions in T_2. Then $m_1 \times m_2$ transitions in T will be labeled a. The input (output) bag of each of these transitions is the sum of the input (output) bags of one transition in T_1 and of one transition in T_2.
- $M_0 = [M_{0,1}^T \; M_{0,2}^T]^T$.

[3] The string w^R is the reversal of string w.

[4] As proved in Section 4.2.

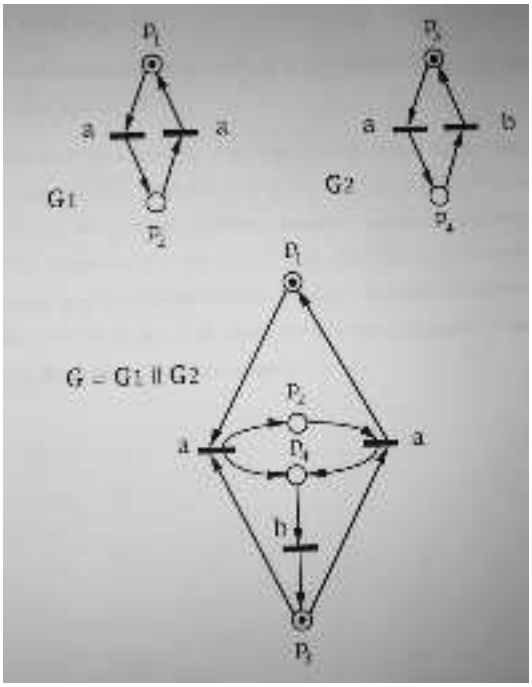


Figure A.4: Two systems $\mathbf{G}_1$, $\mathbf{G}_2$ and their concurrent composition $\mathbf{G}$ in Example A.3.

- F is the cartesian product of F_1 and F_2, i.e., $F = \{[M_1^T \; M_2^T]^T \mid M_1 \in F_1, M_2 \in F_2\}$.

Note that while the set of places grows linearly, the set of transitions and of final markings may grow faster. The composition of more than two systems may computed by repeated application of the procedure for composing two systems.

Example A.3. Let $\mathbf{G}_1 = (N_1, \ell_1, M_{0,1}, F_1)$ and $\mathbf{G_2} = (N_2, \ell_2, M_{0,2}, F_2)$ be two systems in Figure A.4. Here $F_1 = \{(10)^T\}$ and $F_2 = \{(10)^T, (01)^T\}$. Their concurrent composition $\mathbf{G} = \mathbf{G}_1 \parallel \mathbf{G}_2$ is also shown in Figure A.4. The initial marking of $\mathbf{G}$ is $M_0 = (1010)^T$ and its set of final markings is $F = \{(1010)^T, (1001)^T\}$.

We will use the following structural notation for composed systems.

Let $\mathbf{G}_1 = (N_1, \ell_1, M_{0,1}, F_1)$ and $\mathbf{G_2} = (N_2, \ell_2, M_{0,2}, F_2)$ be two PN generators and $\mathbf{G} = \mathbf{G}_1 \parallel \mathbf{G}_2 = (N, \ell, M_0, F)$ their concurrent composition. Let $N = (P, T, Pre, Post)$ and $N_i = (P_i, T_i, Pre_i, Post_i)$, $(i = 1, 2)$. We define:

- The *projection of P on net N_i*, $(i = 1, 2)$, as $P \uparrow_i = P_i$.
- The *projection of T on net N_i*, $(i = 1, 2)$, as $T \uparrow_i = \{t \in T \mid ({}^\bullet t \cup t^\bullet) \cap P \uparrow_i \neq \emptyset\}$. Note that $T \uparrow_i$ may be different from T_i, since additional transitions may have been introduced by composing systems in which the same symbol is labeling more than one transition.
- The *projection of Pre on net N_i*, $(i = 1, 2)$, denoted $Pre \uparrow_i$, as the restriction of Pre to $P \uparrow_i \times T \uparrow_i$.
- The *projection of $Post$ on net N_i*, $(i = 1, 2)$, denoted $Post \uparrow_i$, as the restriction of $Post$ to $P \uparrow_i \times T \uparrow_i$.

Also let M be a marking, $\vec{\sigma}$ be a firing count vector, and σ a firing sequence defined on net N. The *projection of M on N_i*, $(i = 1, 2)$, denoted $M \uparrow_i$, is the vector obtained from M by removing

all the components associated to places not present in N_i. The *projection of* $\vec{\sigma}$ *over* N_i, $(i = 1, 2)$, denoted $\vec{\sigma} \uparrow_i$, is the vector obtained by $\vec{\sigma}$ removing all the components associated to transitions not present in N_i. The *projection of* σ *on* N_i, $(i = 1, 2)$, denoted $\sigma \uparrow_i$, is the firing sequence obtained by σ removing all the transitions not present in N_i.

Appendix B

AN INTRODUCTION TO SUPERVISORY CONTROL

B.1 Discrete Event Systems and Properties

In the supervisory control theory developed by Ramadge, Wonham, et al., a DES is simply a generator of a formal language, defined on an alphabet Σ. The two languages associated with a DES **G** are:

- The *closed behavior* $L(\mathbf{G}) \subseteq \Sigma^*$, a prefix-closed language that represents the possible evolutions of the system.
- The *marked behavior* $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$, that represents the evolutions corresponding to the completion of certain tasks.

Although Ramadge and Wonham have used in their seminal papers a state machine based representation for DES, the theory they built is very general. Any other formalism that can represent the closed and marked behavior of a system may be used in this framework. In the examples presented in this section we will use state machine models to be consistent with the original notation. When relevant, however, we will mention Petri nets.

Example B.1. The state machine in Figure B.1 represents a discrete event system $\mathbf{G}$. The initial state is denoted by an arrow, the final states (just one in this example) are denoted by a double circle. The languages associate to $\mathbf{G}$ are:

$$L_m(\mathbf{G}) = \{abc^n \mid n \geq 0\} = abc^*;$$

$$L(\mathbf{G}) = \{\lambda\} \cup \{a\} \cup \{abc^n \mid n \geq 0\} = \overline{abc^*} = \overline{L_m(\mathbf{G})},$$

where the overline bar represent the prefix closure operator.

We have discussed in Appendix A how the closed and marked behavior may be associated with a Petri net.

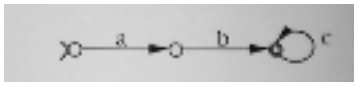


Figure B.1: A discrete event system modeled by a finite state machine.

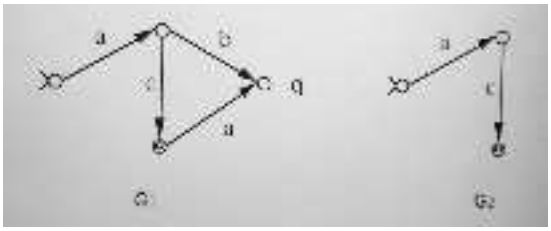


Figure B.2: A blocking discrete event system and its trimmed structure in Example B.1.

A DES is said to be *nonblocking* if any string $w \in L(\mathbf{G})$ can be completed into a string $wx \in L_m(\mathbf{G})$, i.e., into a string that belongs to the marked language. More formally, a DES $\mathbf{G}$ is nonblocking if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$.

This property may be restated for state machines as follows. Let us define as *reachable* a state that may be reached from the initial state, and as *coreachable* a state from which it is possible to reach a final state. Then a state machine is nonblocking if and only if all reachable states are coreachable.

Example B.2. In Figure B.2 we have a blocking system $\mathbf{G}_1$. The strings "ab" and "aca" cannot be completed into a string that belongs to the marked language. Equivalently, from state q it is not possible to reach a final marking.

Note that the DES may be *trimmed* removing the state q and the transitions entering it. In this case we obtain the new DES $\mathbf{G}_2$ with

$$L_m(\mathbf{G}_2) = L_m(\mathbf{G}_1),$$

$$L(\mathbf{G}_2) = \overline{L_m(\mathbf{G}_2)} = \overline{L_m(\mathbf{G}_1)} \subset L(\mathbf{G}_1).$$

Similarly, a Petri net is nonblocking if from any reachable marking it is possible to reach a final marking. Note that this property is significantly different from *deadlock-freeness* and *liveness*. Deadlock-freeness means that the reachability set does not contain a dead marking, i.e., a marking that enables no transition. However a nonblocking system may have a dead marking (if it is a final marking). Liveness implies that from any reachable marking there exists a firing sequence containing all transitions. The following example shows nets with different combinations of liveness and nonblocking properties (deadlock-freeness is implied by liveness, so we will not discuss it).

Example B.3. In Figure B.3 are shown: a) a live and nonblocking system; b) a non-live and nonblocking system; c) a live and blocking system; d) a nonlive and blocking system. The set of final markings F is also given in the figure for each net.

B.2 Controllable Events and Supervisor

The events labels in Σ are partitioned into two disjoint subsets: the set Σ_c of *controllable events* (that can be disabled if desired), and the set Σ_u of *uncontrollable events* (that cannot be disabled by an external agent).

It often required to restrict the behavior of a system within the limits of a *specification language*. We may only enable and disable the controllable events to achieve this behavior. The agent who specifies which events are to be enabled and disabled is called a *supervisor*.

Let us define a *control input* as a subset $\gamma \subseteq \Sigma$ satisfying $\Sigma_u \subseteq \gamma$ (i.e., all the uncontrollable events are present in the control input). If $a \in \gamma$, the event a is enabled by γ (permitted to occur),

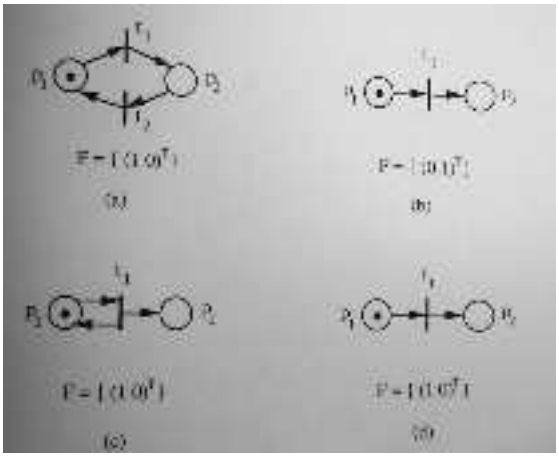


Figure B.3: Combinations of nonblocking and liveness properties for Petri nets with set of final markings F in Example B.2.

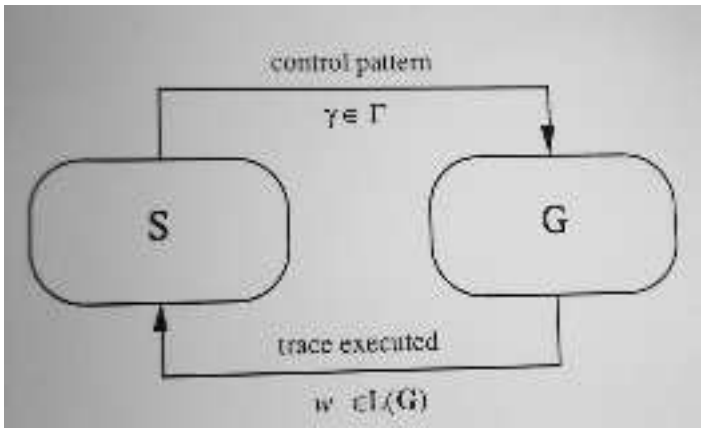


Figure B.4: A closed loop controlled discrete event system.

otherwise a is disabled by γ (prohibited from occurring); the uncontrollable events are always enabled. Let $\Gamma \subseteq 2^{\Sigma}$ denote the set of all the possible control inputs.

A supervisor controls a DES **G** by switching the control input through a sequence of elements $\gamma_1, \gamma_2, \ldots \in \Gamma$, in response to the observed string of previously generated events.

A closed loop block diagram showing a system under supervision is shown in Figure B.4. A point that should be stressed is the fact that the supervisor is implementing a *trace feedback*, rather than a *state feedback*. That is, the control input is not a function of the present state of the system, but of the string of events generated. If the system is deterministic, as generally assumed, it is possible to reconstruct its state from the string of events generated. Thus trace feedback is more general than state feedback.

Example B.4. Consider the system in Figure B.5, where $\Sigma_c = \{a, c\}$ and $\Sigma_u = \{b\}$. The controllable transitions are denoted with a ":". Assume we want a terminal string to contain the event b exactly twice. Table B.1 summarizes the required control input γ after a string w has been generated, and the corresponding state of the system. Note that in state q_0 the control input is different depending on the value of w.

A supervisor may be given as a function $f : L(\mathbf{G}) \rightarrow \Gamma$ specifying the control input $f(w)$

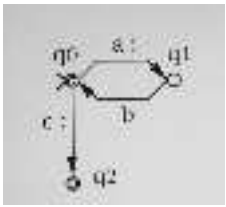


Figure B.5: System in Example B.4

state	w	γ
q_0	λ	$\{a, b\}$
q_1	a	$\{a, b, c\}$
q_0	ab	$\{a, b\}$
q_1	aba	$\{a, b, c\}$
q_0	$abab$	$\{b, c\}$
q_2	$ababc$	$\{b, c\}$

Table B.1: Control inputs in Example B.4.

to be applied for each possible string of events w generated by the system **G**. It also possible to represent a supervisor as another DES **S**, that runs in parallel with the system **G**, i.e., whenever an event occurs in **G** the same event will be executed by **S**. The events enabled at a given instant on **S** determine the control input.

Example B.5. In Example B.4 the supervisor has been described in tabular format. A transition structure for the same supervisor is shown in Figure B.6.

The closed loop system under control will be denoted $\mathbf{S}/\mathbf{G}$ and may be considered as a new discrete event system [Ramadge 87]. Let us assume in the following that the supervisor is given as a DES **S**. Then, the closed behavior of $\mathbf{S}/\mathbf{G}$ is

$$L(\mathbf{S}/\mathbf{G}) = L(\mathbf{G}) \cap L(\mathbf{S}),$$

i.e., the subset of the uncontrolled behavior that survives under supervision. For what regards the language accepted by $\mathbf{S}/\mathbf{G}$ there are two possible definitions. If **S** has a set of final states we may define as marked behavior of $\mathbf{S}/\mathbf{G}$ as

$$L_m(\mathbf{S}/\mathbf{G}) = L_m(\mathbf{G}) \cap L_m(\mathbf{S}),$$

i.e., all marked strings of **G** that are also marked by **S**. If we assume that the supervisor does not mark strings, i.e., it has no final states, $L_m(\mathbf{S})$ is not defined. In this case, we call the supervisor *non marking* and we define $L_m(\mathbf{S}/\mathbf{G})$ as

$$L_m(\mathbf{S}/\mathbf{G}) = L_m(\mathbf{G}) \cap L(\mathbf{S}/\mathbf{G}),$$

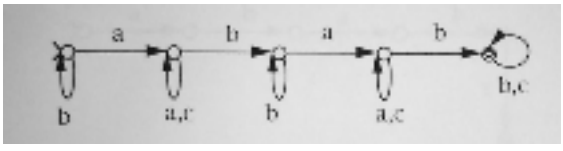


Figure B.6: Supervisor in Example B.5

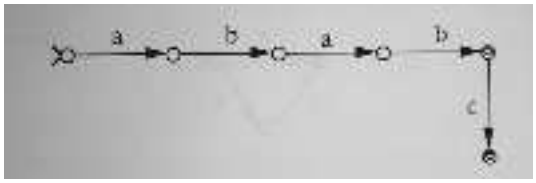


Figure B.7: Closed loop system $\mathbf{S}/\mathbf{G}$ in Example B.6.

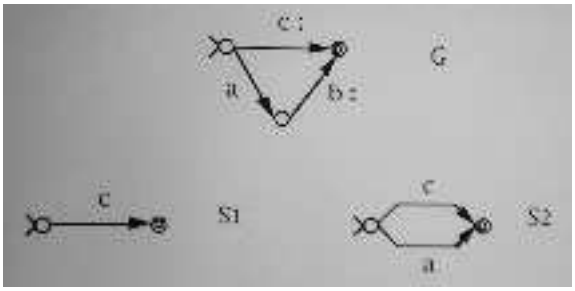


Figure B.8: Systems $\mathbf{G}$, $\mathbf{S_1}$, and $\mathbf{S_2}$ in Example B.7.

i.e., the subset of the uncontrolled marked behavior that survives under supervision, and call it *controlled behavior*[1].

Example B.6. The DES in Figure B.7 represented the closed loop system $\mathbf{S}/\mathbf{G}$ for the system and supervisor discussed in Example B.5. Here the closed and marked behavior of $\mathbf{S}/\mathbf{G}$ are $L(\mathbf{S}/\mathbf{G}) = \overline{ababc}$, and $L_m(\mathbf{S}/\mathbf{G}) = abab + ababc$, as can be derived from the figure. If we had considered a supervisor with no final states, we would have considered, in place of the marked behavior, the controlled behavior $L_m(\mathbf{S}/\mathbf{G}) = \lambda + ab + abab + ababc$.

Now let us assume that we are given two DES $\mathbf{G}$ and $\mathbf{S}$. $\mathbf{S}$ qualifies as a proper supervisor for $\mathbf{G}$ if the two following properties are ensured.

- *controllability*: $\mathbf{S}$ does not disable any uncontrollable event that may occur in $\mathbf{G}$, i.e., $(\forall w \in L(\mathbf{S}/\mathbf{G}), \forall a \in \Sigma_u)\ [wa \in L(\mathbf{G}) \Longrightarrow wa \in L(\mathbf{S}/\mathbf{G})]$.
- *nonblockingness* (or *properness*): the behavior of the system under supervision must be nonblocking, i.e., $L(\mathbf{S}/\mathbf{G}) = \overline{L_m(\mathbf{S}/\mathbf{G})}$.

Example B.7. Let $\mathbf{G}$, $\mathbf{S_1}$, and $\mathbf{S_2}$ be the DES in Figure B.8. Here $\Sigma_c = \{b, c\}$. $\mathbf{S_1}$ is not a supervisor for $\mathbf{G}$ because in the initial state it disables the uncontrollable event a that is enabled in the initial state of $\mathbf{G}$. $\mathbf{S_2}$ is not a proper supervisor for $\mathbf{G}$ because if the event a is executed it prevents the system from reaching the final state (in fact the event b is never enabled).

B.3 Supervisory Control Problem

Controlling a DES consists in restricting its open loop behavior within a given *specification* language. Thus we may state the two following Supervisory Control Problems.

[1]The notation is unfortunately ambiguous. In [Ramadge 87] $L_m(\mathbf{S}/\mathbf{G})$ denotes the marked behavior of $\mathbf{S}/\mathbf{G}$, while $L_c(\mathbf{S}/\mathbf{G})$ denotes the controlled behavior. In [Ramadge 89b] and in the work of other authors, however, $L_m(\mathbf{S}/\mathbf{G})$ has been used to denote the controlled behavior of $\mathbf{S}/\mathbf{G}$, since only non marking supervisors are considered. We will use $L_m(\mathbf{S}/\mathbf{G})$ to denote the language accepted by the closed loop system, specifying whenever necessary if it represents the marked or controlled behavior.

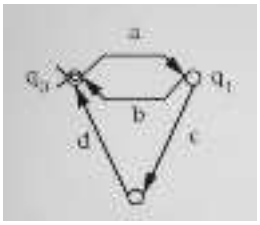


Figure B.9: System to control in Example B.8.

SCP1 Given a DES $\mathbf{G}$ and a specification language $L \subseteq L(\mathbf{G})$, there exists a supervisor $\mathbf{S}$ such that $L(\mathbf{S}/\mathbf{G}) = L$?

SCP2 Given a DES $\mathbf{G}$ and a specification language $L \subseteq L_m(\mathbf{G})$, there exists a nonblocking supervisor $\mathbf{S}$ such that $L_m(\mathbf{S}/\mathbf{G}) = L$? (Here we assume that $\mathbf{S}$ does not mark strings, i.e., $L_m(\mathbf{S}/\mathbf{G})$ is the controlled behavior of $\mathbf{S}/\mathbf{G}$.)

The solution of these two problems is based on the notions of controllable language and $L_m(\mathbf{G})$-closure, defined in the following.

A language $K \subset \Sigma^*$ is said to be *controllable* (with respect to $L(\mathbf{G})$ and Σ_u) if $\overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}$. This means that it is always possible, preventing the occurrence of controllable transitions, to restrict $L(\mathbf{G})$ within K.

A language $K \subset L_m(\mathbf{G})$ is said to be $L_m(\mathbf{G})$*-closed* if $K = \overline{K} \cap L_m(\mathbf{G})$. This means that any marked string of $\mathbf{G}$ that is the prefix of some string of K is also a string of K.

The following two theorems, due to Ramadge and Wonham [Ramadge 87, Ramadge 89b], give necessary and sufficient conditions for the existence of a supervisor.

Theorem B.1 ([Ramadge 87], P. 5.1). *For nonempty $L \subseteq L(\mathbf{G})$ there exists a supervisor $\mathbf{S}$ such that $L(\mathbf{S}/\mathbf{G}) = L$ if and only if L is prefix closed and controllable.*

Theorem B.2 ([Ramadge 87], T. 6.1). *For nonempty $L \subseteq L_m(\mathbf{G})$ there exists a nonblocking supervisor $\mathbf{S}$ such that $L_m(\mathbf{S}/\mathbf{G}) = L$ if and only if L is $L_m(\mathbf{G})$-closed and controllable.*

Example B.8. The DES $\mathbf{G}$ in Figure B.9 has closed behavior $L(\mathbf{G}) = \overline{(a(b+cd))^*}$, and marked behavior $L_m(\mathbf{G}) = (a(b+cd))^*$.

1. Let $L = \overline{(ab)^*}$ be the desired closed behavior of the closed loop system.

   (a) If $\Sigma_u = \{a, d\}$, L is controllable and may be enforced by a supervisor. The supervisor simply needs to disable the controllable event c whenever the system is in state q_1.

   (b) If $\Sigma_u = \{b, c\}$, L is not controllable and cannot be enforced by a supervisor. In fact the supervisor cannot disable the event c, which is now uncontrollable, when the system is in state q_1.

2. Let $L = (abab)^*$ be the desired controlled behavior of the closed loop system and assume $\Sigma_u = \{a, d\}$. L is controllable, but is not $L_m(\mathbf{G})$-closed. Hence there does not exit a supervisor $\mathbf{S}$ such that $L_m(\mathbf{S}/\mathbf{G}) = L$.

   This can be also be shown by contradiction in this particular example. By definition of controlled language, $L_m(\mathbf{S}/\mathbf{G}) = L_m(\mathbf{G}) \cap L(\mathbf{S}/\mathbf{G})$. Now assume $L_m(\mathbf{S}/\mathbf{G}) = L$; then $w = abab \in L_m(\mathbf{S}/\mathbf{G}) \subseteq L(\mathbf{S}/\mathbf{G})$ and $w' = ab \notin L_m(\mathbf{S}/\mathbf{G})$. However $w \in L(\mathbf{S}/\mathbf{G}) \Longrightarrow w' \in L(\mathbf{S}/\mathbf{G})$, i.e., $w' \in L_m(\mathbf{G}) \cap L(\mathbf{S}/\mathbf{G}) = L_m(\mathbf{S}/\mathbf{G})$, a contradiction.

Another important result due to Ramadge and Wonham is concerned with the existence of finite state machine supervisors.

Theorem B.3 ([Wonham 87], T. 3.1). *Let* $\mathbf{G}$ *be a finite state machine (i.e., its closed and marked behavior are* regular *languages) and let* L *be a* regular *specification language. If* L *satisfies the conditions of Theorem B.1, the supervisor* $\mathbf{S}$ *such that* $L(\mathbf{S}/\mathbf{G}) = L$ *may be represented as a finite state machine. If* L *satisfies the conditions of Theorem B.2, the supervisor* $\mathbf{S}'$ *such that* $L_m(\mathbf{S}'/\mathbf{G}) = L$ *may be represented as a finite state machine.*

B.4 Supremal Controllable Sublanguage

Assume we want to restrict the closed behavior of a system $\mathbf{G}$ within the limits of a legal language L that is not controllable. By Theorem B.1 a supervisor $\mathbf{S}$ such that $L(\mathbf{S}/\mathbf{G}) = L$ does not exist. However we may consider a controllable sublanguage $K \subseteq L$ that may be enforced by a supervisor $\mathbf{S}'$. Thus the behavior of the closed loop system is now $L(\mathbf{S}'/\mathbf{G}) = K \subset L$, i.e., it is a subset of the legal behavior.

We also want to minimally restrict the behavior of the system, i.e., we want to construct the supervisor which allows the largest behavior of the system within the limits given by the specification L. This behavior is called the *supremal controllable sublanguage*.

Let us define $\mathcal{C}(\mathbf{G}) = \{K \mid \overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}\}$ as the set of all languages controllable with respect to $L(\mathbf{G})$ and Σ_u. Given a language L we will define the supremal controllable sublanguage of L as

$$L^{\uparrow} = \sup\{K \mid K \subseteq L, K \in \mathcal{C}(\mathbf{G})\}$$

Theorem B.4 ([Ramadge 87], P. 7.1). *The supremal controllable sublanguage for a given supervisory control problem exists and is unique.*

However it may well be the case that $L^{\uparrow}$ contains only the empty string or is even the empty language.

Example B.9. For the problem in Example B.8.1b $L^{\uparrow} = \{\lambda\}$.

Since the supremal controllable sublanguage may be too restrictive, other controllable approximations to L have been defined. An extension by Lafortune and Chen [Lafortune 90a] to the standard Supervisory Control Problem, called SCPB (SCP with Blocking) considers a different approximation of the desired behaviour in terms of *Infimal Controllable Superlanguage* defined as

$$L^{\downarrow} = \inf\{K \mid K \supseteq L, K \in \mathcal{C}(\mathbf{G})\}.$$

This may, however, generate blocking. We will not consider SCPB in this thesis.

Ramadge and Wonham have also shown that in the regular case (i.e., when both the system behavior and the specification behavior are regular languages) the supremal controllable sublanguage is a regular language and can be computed in a finite number of steps.

As a final remark, we note that the supremal controllable sublanguage is optimal from a logical point of view but not necessarily from the performance point of view.

Example B.10. The Petri net system in Figure B.10 has a delay associated with each event (i.e., each transition). The place p_0 represents the effect of the supervisor that ensures that places p_2 and p_6 are never marked at the same time. In order to minimally restrict the behavior we allow the initial firing of either "ab" or "de". Assume the delays are the following.

event	a	b	c	d	e	f
delay [s]	0	10	10	0	8	4

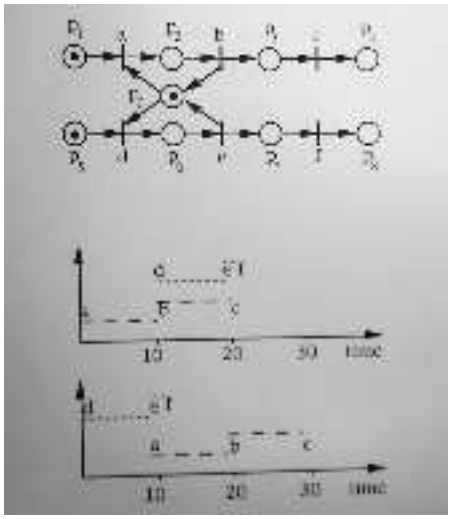


Figure B.10: A timed Petri net model in Example B.10.

If the sequence "ab" is initially executed, we have that the required time to reach the final marking $(0001001)^T$ is 22 seconds, while if the sequence "de" is initially executed the final marking is reached in 28 seconds. Clearly, if it is required that the system reaches the final marking in the shortest time, it would be better to further restrict the system's behavior allowing only the firing of the sequence "ab" from the initial marking.

Appendix C

NOTATION

$a, b, c, \ldots$	Symbols of an alphabet, events.
B	A basis of nonnegative P-invariants.
$C = Post - Pre$	The incidence matrix of a P/T net.
$\mathcal{C}(\mathbf{G})$	The set of languages controllable with respect to $L(\mathbf{G})$ and Σ_u, i.e., the set $\{K \mid \overline{K}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{K}\}$.
$\mathbf{E} = \mathbf{G} \parallel \mathbf{H}$	Discrete event system $\mathbf{G}$ constrained by specification $\mathbf{H}$
F	A set of final markings of a P/T net.
$\mathbf{G}$	A discrete event system. A Petri net model of a discrete event system is $\mathbf{G} = (N, \ell, M_0, F)$.
$\mathbf{H}$	A discrete event system representing a specification.
K	A controllable language.
$\ell : T \rightarrow \Sigma$	A labeling function that assigns to each transition of a net a label from the alphabet Σ.
L	A language.
$\overline{L}$	The prefix-closure of language L.
$L^{\uparrow}$	The supremal controllable sublanguage of language L.
$L(N, M_0)$	The set of firing sequence of $\langle N, M_0 \rangle$.
$L(\mathbf{G})$	The closed behavior of the discrete event system $\mathbf{G}$. If $\mathbf{G}$ is given as a Petri net model, the closed behavior is also called P-type Petri net language of $\mathbf{G}$.
$L_m(\mathbf{G})$	The marked behavior of the discrete event system $\mathbf{G}$. If $\mathbf{G}$ is given as a Petri net model, the marked behavior is also called L-type Petri net language of $\mathbf{G}$.
$\mathcal{L}, \mathcal{L}_d, \mathcal{L}_{DP}$	The sets of L-type, deterministic L-type, and DP-closed Petri net languages.
$M : P \rightarrow \mathbb{N}$	A marking of a P/T net (M_0 is an initial marking, M_f is a final marking).
$\mathcal{M}(\vec{w}, k)$	The set of marking satisfying a generalized mutual exclusion constraint, i.e., the set $\{M \in \mathbb{N}^{\lvert P \rvert} \mid \vec{w}^T \cdot M \leq k\}$.
$N = (P, T, Pre, Post)$	A P/T net.
N^R	The reversal of a P/T net N.

N^S	A P/T net N with the addition of a monitor place S, i.e. the net $(P \cup \{S\}, T, Pre^S, Post^S)$.
$\mathbb{N}$	The set of nonnegative integers.
$\langle N, M_0 \rangle$	A marked net, i.e., a net N with initial marking M_0.
p	A place of a P/T net.
${}^\bullet p, p^\bullet$	The sets of input and output transitions of place p.
P	A set of places of a P/T net.
$Post : P \times T \to \mathbb{N}$	The post-incidence matrix of a P/T net.
$Pre : P \times T \to \mathbb{N}$	The pre-incidence matrix of a P/T net.
$PR(N, M_0)$	The set of markings $\{M \in \mathbb{N}^{\lvert P \rvert} \mid (\exists \vec{\sigma} \in \mathbb{N}^{\lvert T \rvert})[M = M_0 + C \cdot \vec{\sigma}\}$ (potentially reachable set).
$PR^A(N, M_0)$	The set of markings $\{M \in \mathbb{N}^{\lvert P \rvert} \mid A \cdot M \geq A \cdot M_0\}$.
$PR^B(N, M_0)$	The set of markings $\{M \in \mathbb{N}^{\lvert P \rvert} \mid B^T \cdot M = B^T \cdot M_0\}$.
$\mathcal{P}, \mathcal{P}_d$	The sets of P-type, and deterministic P-type Petri net languages.
Q_X	The support of a vector $X : A \to \mathbb{N}$, i.e., the set $\{a \in A \mid X(a) > 0\}$.
$R(N, M_0)$	The set of reachable markings of $\langle N, M_0 \rangle$ (reachability set).
$\mathbf{S}$	A supervisor.
$\mathcal{S}$	A siphon of a P/T net.
$\mathbf{S}/\mathbf{G}$	The closed-loop system composed by the system $\mathbf{G}$ under the action of supervisor $\mathbf{S}$.
t	A transition of a P/T net.
${}^\bullet t, t^\bullet$	The sets of input and output places of transition t.
T	A set of transitions of a P/T net.
$\mathcal{T}$	A trap of a P/T net.
w, x, y, z	Strings of a language, sequences of events.
$(\vec{w}, k)$	A generalized mutual exclusion constraint.
$(W, \vec{k})$	A set of generalized mutual exclusion constraints.
$X : A \to \mathbb{N}$	A vector.
Y	A nonnegative P-invariant (P-semiflow) of a P/T net.
γ	A control input determined by a supervisor.
Γ	The set of all possible control inputs determined by a supervisor.
θ	A simple path of a P/T net.
λ	The empty string.
σ	A firing vector of transitions.
$\vec{\sigma} : T \to \mathbb{N}$	A firing count vector of a marked net.
Σ	An alphabet. Σ_c is an alphabet of controllable events; Σ_u is an alphabet of uncontrollable events.
Σ^*	The Kleene star of alphabet Σ, i.e., the language containing all strings composed with symbols in Σ and the empty string λ.

$\uparrow$	The projection operator.
$\|$	The concurrent composition operator.
$\|_d$	The shuffle operator (concurrent composition over disjoint alphabets).
$\diamond$	Denotes the end of a proof.