# Computational Procedures for Robust Nonblocking Supervisory Control of Discrete-Event Systems

Farid Yari

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

June 2015

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By : **Farid Yari**

Entitled : **Computational Procedures for Robust Nonblocking Supervisory Control of Discrete-Event Systems**

and submitted in partial fulfillment of the requirements for the degree of

**Master of Applied Science**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee :

_____ Chair
Dr. R. Raut

_____ Examiner, External
Dr. F. Nasiri (BCEE) To the Program

_____ Examiner
Dr. K. Khorasani

_____ Supervisor
Dr. S. Hashtrudi Zad

Approved by _____
Dr. W. E. Lynch, Chair
Department of Electrical and Computer Engineering

_____ 2015.

_____
Dr. Amir Asif, Dean
Faculty of Engineering and Computer Science

## ABSTRACT

### Computational Procedures for Robust Nonblocking Supervisory Control of Discrete-Event Systems

Farid Yari

The concept of robust control arises in control theory in dealing with modeling uncertainties and model changes. In the study of supervisory control of discrete-event systems (DES), one approach to robustness is to assume that the exact plant model is unknown but it belongs to a finite family of DES models. The design objective is to find a supervisor such that any of the plant DES models in the aforementioned family, under the supervision of the designed supervisor, meets its design specifications. The set of solutions of the robust nonblocking supervisory control problem (RNSCP) is available in the literature in terms of a class of languages. For the case of control with full event observation, RNSCP has an optimal (maximally permissive) solution. In the case of control under partial event observation (RNSCP-PO), a maximally permissive solution does not necessarily exist; however suboptimal solutions (in terms of normal languages) that are generally more suitable for computational procedures have been identified.

In this thesis, computational algorithms are developed for finding the solution of RNSCP in the form of finite-state automaton. First, a computational algorithm for supremal $\boldsymbol{G}$-nonblocking languages is presented. Next this algorithm is used to develop an iterative algorithm that obtains the maximally permissive solution of RNSCP as the largest fixed point of a suitable operator. It is shown that the algorithm converges in a bounded number of steps for finite-state plant models and

regular specification languages. The computational complexities of the algorithms are also derived. The resulting algorithms have been implemented in MATLAB environment using Discrete Event Control Kit (DECK) and applied to solve a problem of control and fault recovery in a simplified spacecraft propulsion system. Next the computational algorithm for RNSCP is extended to the case of control under partial event observation. In this case the maximally permissive solution of RNSCP-PO among the solutions that have the normality property is obtained.

**Acknowledgments**

It is hard to overstate my gratitude to my supervisor Dr. Shahin Hashtrudi Zad. I really appreciate the time and energy he devoted to this work. Completion of my masters would not have been possible without his never-ending support, patience and guidance. I feel fortunate to work under his supervision.

I would also thank Dr. Siamak Tafazoli for his valuable guidance in the spacecraft propulsion system analysis.

I owe sincere gratitude to Farzam Boroomand, Hamid Mahboubi, and Nazanin Hashemi for their kind help during research and thesis completion.

Special thanks to Sepide Movaghati for her continued support, understanding and never failing faith in me in the past years.

I dedicate this thesis to my parents for their unconditional love, support and encouragement throughout my life.

# Contents

# List of Figures

# Chapter 1

# Introduction

In this thesis computational algorithms are developed for the robust nonblocking supervisory control problem of discrete-event systems. In this problem, it is assumed that the true model of the system is unknown. However, it belongs to a set of possible models. We show that for finite-state models and regular language design specifications, the algorithms terminate in a finite number of steps.

In this chapter, discrete-event systems and conventional (non-robust) supervisory control problem are reviewed in Section 1.1. The robust supervisory control with model uncertainty under full observation and partial observation of events is discussed in Section 1.2. Some important features of computational algorithms are examined in Section 1.3 followed by a literature review on conventional supervisory control, robust supervisory control and computational algorithms in Section 1.4. Next the the objectives and contributions of this thesis are presented in Section 1.5, and the organization of the thesis is described in Section 1.6.

## 1.1 Supervisory Control of Discrete-event Systems

A discrete-event system (DES) is a dynamic system with discrete state-space and event-driven state transition. Therefore, the dynamics of these systems can be modeled as sequences of events. Consider a spacecraft propulsion system as an example.

The engine can be in one of the two states ON or OFF and this engine state can be changed by commands START-UP or SHUTDOWN (both events).

However, not all the sequences generated by a system are desirable (safe). For instance, there could be some illegal states in the system that must be avoided. In this case, the behavior of the DES has to be modified by means of a controller in order to ensure that the system's behavior is admissible.

In the supervisory control theory of discrete-event systems proposed by Ramadge and Wonham [33], a supervisor (controller) is responsible for restricting the system (plant) behavior so that certain specifications are satisfied. In this theory, the events are assumed to be either *controllable* or *uncontrollable*. Supervisory control is described as a feedback control structure where the supervisor (placed in the feedback loop) commands the control action by enabling or disabling certain events after monitoring the event sequences generated by the plant. Such control framework is shown in Figure 1.1. In this problem the supervisor is only allowed to disable the controllable events.

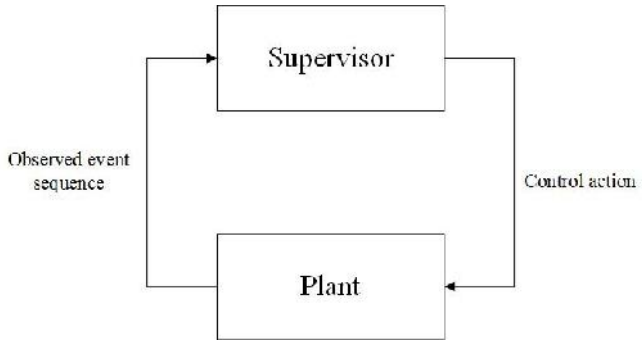



Figure 1.1: Supervisory control structure.

The specifications can be in terms of a set of legal sequences (also known as legal language) or they can be characterized using automata. In automata theory and many practical cases, finite-state automata are used to describe the system's behavior, specifications and the supervisor (to be designed). The supervisor has to restrict the behavior of the system under supervision to a legal behavior (which satisfies the design specifications) by disabling controllable events whenever necessary.

## 1.2 Robust Supervisory Control

Although it is assumed that the system model is known in the conventional supervisory control problem, the system dynamics are prone to changes due to failures or new conditions during its operation. In order to tackle modeling uncertainty or changes to the system, the notion of robust control is developed.

Several approaches have been proposed to deal with robust supervisory control. We consider the setup in which the system model is among a finite family of plant models. The main goal is to design a robust supervisor such that it works for all plant models. In other words, it can restrict every plant model to its legal behavior.

Fault recovery is a problem that can be regarded as a robust control problem. In a fault recovery problem, the system can operate either in normal mode or in several faulty modes. In each of these different modes, there are certain design specifications which the system under supervision is expected to follow. Thus, a robust supervisor is to be designed to guarantee that system under supervision satisfies all of the design specifications in every mode of operation.

As an example for this kind of problem, consider the propulsion system shown in Figure 1.2. Now, suppose that the valves may fail stuck-open or stuck-closed. Therefore, every valve has two *failure* modes in addition to the *normal* mode. Thus, the overall system has several possible modes of operation including a normal mode where there is no failure in the system, and failure modes in which at least one of the valves is failed, either stuck-open or stuck-closed.

Robust control can also be used to find solutions to problems that do not involve model uncertainties or changes. One such example is the problem of *Supervisory Control with Multiple sets of Marked states* [9, 14]. In supervisory control, the set of states in which the goals are met are "marked". It is desirable that these marked states can always be reached in which case the system is said to be nonblocking. In problems with multiple marked sets, nonblocking property is desired with respect to more than one set of marked states. The study of this problem is motivated by the

fact that many systems have multiple operational modes (each corresponding to a set of marked states) and it is desirable to be able to steer the system to any one of these modes. For instance, in a propulsion system with two engines $E1$ and $E2$, it is desirable to have the ability to take the system to any of the three operational modes of $OFF$, $E1{:}ON$ and $E2{:}ON$. In other industries, examples can be easily found of systems that operate in different modes or configurations. In all these cases, nonblocking with respect to more than one set of marked states is required.

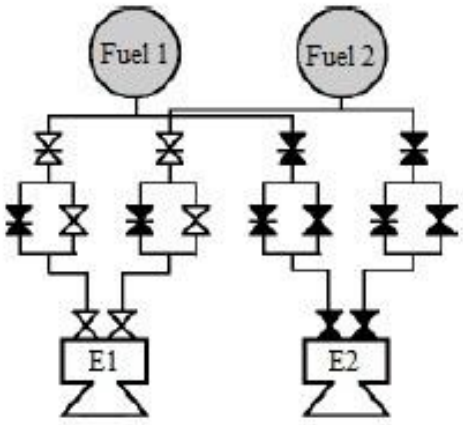



Figure 1.2: Spacecraft propulsion system.

## 1.3 Computational Procedures

In supervisory control theory, discrete-event systems are modeled by automata. However, design specifications are usually described using natural language (e.g. avoid buffer overflow). In order to tackle the supervisor synthesis, we have to find an automaton representation for the specification languages. We assume that the specification languages can be represented by finite-state automata. In the supervisory control problem, the question of existence and the synthesis of a minimally restrictive supervisor is reduced to finding the supremal controllable sublanguage of the specification language. This calculation is done by a recursive algorithm and the supremal

controllable sublanguage is characterized as the largest fixed point of a monotone operator. This iterative algorithm is shown to be finitely convergent if the plant is finite-state automaton and the specifications are regular languages (hence represented by finite-state automata).

Computational algorithms for calculating other supremal sublanguages of regular languages are also developed as recursions on an initial automaton. The most challenging part in solving such problems is to come up with an automaton that has enough information about the states and transitions of the plants and the legal behaviors. In each iteration of these recursive algorithms, certain illegal states and transitions which do not satisfy the desired property are removed. In this framework, the finite convergence is evident since no state or transition is added to the starting automaton and the pruning of the initial automaton terminates in a finite number of steps.

In this thesis, we take a similar approach to find computational algorithms for solving the robust nonblocking supervisory control problem.

## 1.4 Literature Review

Robust supervisory control and computational algorithms for (non-robust) supervisory control problem have been studied by several researchers so far. We briefly review some relevant studies and discuss their results.

### 1.4.1 Supervisory Control

Supervisory control of discrete-event systems was first introduced in [32, 33]. In this framework, the event set is partitioned into two disjoint sets of controllable and uncontrollable events. A supervisor can disable or enable events in each state to restrict the behavior of the plant within a desired behavior called specification. In such framework, a supervisor is called "admissible" if no uncontrollable event

is disabled. In [32, 33], the concept of controllability is introduced to tackle the supervisor synthesis problem.

[11, 25] extend the theory of supervisory control of discrete-event systems [32] to the problem of supervisory control under partial observation. There the set of events is partitioned into two disjoint sets of observable and unobservable events. The notion of language observability is introduced to solve the partial observation supervisory control problems. The solution of these supervisory control problems are obtained in terms of controllable and observable languages. A subset of the solutions have also been identified in terms of controllable and normal languages. This subset lends itself better to computational algorithms and in some cases, it includes the optimal solution to the original supervisory control problem.

The supervisory control problem in [32, 44] has been extended to many areas. In the following section, we review one of these extensions, robust supervisory control, which deals with model uncertainties or changes.

### 1.4.2 Robust Supervisory Control

The concept of robust control arises in control theory in dealing with modeling uncertainties or model changes. Several approaches have been explored for robust supervisory control of discrete-event systems (DES) (see, e.g. [12], [13], [24], [31]). In [24], the plant model belongs to a finite family of DES models $\boldsymbol{G}_1$,...,$\boldsymbol{G}_n$, and the objective is to design a supervisor such that all plant models under supervision satisfy a common design specification $K$ which is assumed to be a sublanguage of the marked behavior of all plant models, i.e. $K \subseteq \bigcap_i L_m(\boldsymbol{G}_i)$.

In [36] the results of [24] are extended to timed discrete-event systems and assumes separate design specifications for each plant model which are obtained by taking the intersection of a language $E$ with each plant's closed behavior. The discussion in [36] is limited to closed languages.

The results of [24] are generalized in [3] to the case involving separate non-prefix

closed design specification for each plant model and considers the nonblocking property. In this framework, the overall legal behavior is obtained from each plant's specification. The nonblocking property is guaranteed through the (sufficient condition of) *nonconflicting* property. [3] considers that all of the events are observable and also provides an algorithm for finding the optimal solution of the aforementioned robust nonblocking supervisory control problem.

In [35], the results of [3] are extended to the case of control under partial observation. Furthermore, [35] replaces the *nonconflicting* property with the $\boldsymbol{G}_i$*-nonblocking* property. This results in a characterization of the solution of nonblocking robust control in the form of a set of necessary and sufficient conditions of *controllability, observability,* $L_m(\boldsymbol{G})$*-closure* and $\boldsymbol{G}_i$*-nonblocking.*

Some of the extensions of this framework include modular implementation in closed languages [15], control with communication delays [30], networked DES [40], limited lookahead policies [2], robust diagnosis [37] and robust failure prognosis [38].

As mentioned earlier, robust control is used to deal with model changes. For instance in fault recovery problems, the plant model can be in normal or a set of faulty modes and a robust supervisor can be designed to meet the design specification of each mode [34], [35]. Robust control can also be used to find solutions to problems that do not involve model uncertainties or changes. One such example is the problem of *Supervisory Control with Multiple sets of Marked states* [9].

Other approaches for fault detection and recovery problems in DES have been also studied. [29] uses a diagnoser for fault detection and after the failure it switches to another supervisor. The approach in [29] is to safely detect faults and switch over from one supervisor to another from a set of reconfigured control laws. In [22], it is considered that there is a post-fault specification and it has to be satisfied after the failure. Furthermore, the supervisor is reconfigured only after detecting the fault. [42] develops a fault tolerant supervisor that enforces the behavior of the system to a non-faulty behavior in a finite number of steps after the failure. [41] studies the synthesis of the fault tolerant supervisor in [42].

In this thesis we model the system uncertainty as done in [24] and develop computational algorithms for obtaining the solution of robust nonblocking supervisory control problem presented in [35]. In the following section we review some of the works in the area of computational algorithms for obtaining supremal sublanguages in supervisory control problems.

### 1.4.3 Computational Procedures

In the supervisory control theory for discrete-event systems, the notion of controllable languages plays a key role in supervisor design. Specifically the main issue is to calculate the supremal controllable sublanguage of a given specification (or legal behavior). [44] characterizes the supremal controllable sublanguage as the largest fixed point of an operator and provides a recursive computational algorithm for calculating the resulting maximally permissive solution of the supervisory control problem. The finite convergence of this algorithm is also shown, provided that the plant and specification can be modeled as finite-state automata.

[17] develops a general unifying framework for computational algorithms for calculating supremal elements. The computational algorithm for obtaining the supremal controllable sublanguage in [44] is a special case of the algorithm in [17].

The class of observable sublanguages of a given language need not have a supremal element. [10] extends the results of [11, 25] to develop algorithms for the computation of the supremal normal and controllable sublanguage. The concept of normality is stronger than observability; therefore the obtained solution is not necessarily optimal. [39] comes up with a subclass of observable sublanguages which has a supremal element and develops a computational algorithm for finding the supremal language of the subclass of observable sublanguage that are controllable and $L_m(\boldsymbol{G})$-closed. This solution is in general a superset of the supremal $L_m(\boldsymbol{G})$-closed, controllable and normal sublanguage of that specified language. The algorithm in this work is a modification of the algorithm presented in [10].

[8] studies the class of nonconflicting sublanguages of a given language and presents closed-form expressions as well as computational algorithms for computing the supremal nonconflicting sublanguage of a given language. It also investigates computational algorithms for obtaining the supremal controllable and nonconflicting sublanguage.

[4] provides closed-form formulas for computing the supremal controllable and normal sublanguage provided that all the languages are prefix-closed. Next, [21] extends this result to develop closed-form expressions for calculating the supremal normal and supremal closed and normal sublanguages of given non-closed languages and arbitrary masks. [45] introduces a new iterative algorithm for computing the supremal controllable and normal sublanguage of a given non-closed language using the formula provided in [4] for calculating the supremal controllable and normal sublanguage of a closed language. The presented algorithm in this work does not iterate between the supremal controllable and supremal normal operations (as done in [10]). [18] presents a non-iterative algorithm for computation of supremal controllable and normal sublanguage provided that the plant and the projection mapping satisfy certain assumptions.

[20] provides computational algorithms for supremal controllable and normal sublanguage to solve the supervisory control under partial observation from an optimal control point of view.

In this thesis, we develop computational algorithms for solving the robust control problems of [35].

## 1.5 Thesis Contributions

The major contributions of this thesis can be summarized as follows.

- We develop computational algorithms for obtaining the optimal (maximally permissive) solution to the Robust Nonblocking Supervisory Control Problem

(RNSCP) with full event observation presented in [35]. Computational algorithms are presented for finding the supremal $L_m(\boldsymbol{G})$-closed and supremal $\boldsymbol{G}_i$-nonblocking sublanguages of a given legal language. We show that the algorithm converges in a bounded number of steps assuming finite-state plant models and regular specification languages.

- We extend the iterative computational algorithm for the RNSCP with full observation to the case of partial event observation by developing a new computational algorithm for obtaining the supremal normal sublanguage and combining it with the solution of RNSCP. The finite convergence of the proposed algorithm is also shown in this thesis.

- All of the algorithms for computing the solution of RNSCP with full observation are implemented in MATLAB using Discrete Event Control Kit (DECK) [46].

- The proposed computational algorithm is used to solve a problem of supervisory control and fault recovery in a simplified spacecraft propulsion system. We will see that non-robust supervisory control does not provide a satisfactory solution to this problem and using a robust supervisor is essential.

## 1.6 Thesis Outline

In Chapter 2, we review the required background on supervisory control (conventional and robust) and on existing computational algorithms. In Chapter 3, we develop an iterative algorithm for obtaining the optimal solution of RNSCP with full observation and provide the related computational algorithms. We also discuss the computational complexity of the proposed algorithms. We also formulate and solve a problem of robust supervisory control of a spacecraft propulsion system. Chapter 4 extends the computational algorithms of RNSCP with full observation to the case of partial event observation. Chapter 5 summarizes the thesis contributions and discusses possible future research.

# Chapter 2

# Background

In this thesis we aim to develop computational procedures for robust nonblocking supervisory control problem under full observation and extend the results to deal with partial event observation. Therefore in this chapter, we briefly review some background materials and preliminaries on supervisory control theory of discrete-event systems which are required for presenting our work. The first section introduces the discrete-event system models and language theory used to describe them. The conventional supervisory control problem is discussed in Section 2.2 followed by a review of robust nonblocking supervisory control problem for full observation and then partial event observation in Section 2.3. Finally, computational procedures for supervisory control problem are reviewed in Section 2.4.

## 2.1 Discrete-Event Systems

A discrete-event system (DES) is defined as an event-driven system with a discrete state space. The state transition mechanism in DES depends on occurrence of asynchronous discrete events. Discrete-event systems cover a wide range of systems such as manufacturing systems, computer and communication networks, traffic systems and robotics. Among many approaches for modeling discrete-event systems, we use the automata theory. In this work, discrete-event systems are modeled by finite-state

automata. Before discussing automata, we review some basic definitions related to sequences (words) and languages.

### 2.1.1 Languages

Let $\Sigma$ be a finite set of events (symbols), also known as *alphabet*. The set $\Sigma^+$ denotes the set of all finite event sequences (strings or words) over alphabet $\Sigma$ [43, 7].

$$\Sigma^+ := \{\sigma_1, ..., \sigma_k \mid k \geq 1, \sigma_i \in \Sigma\}.$$

The empty string is defined as a sequence with no events and denoted by $\epsilon$. Then we write

$$\Sigma^* := \Sigma^+ \cup \{\epsilon\}.$$

A *language* over $\Sigma$ is defined as any subset of $\Sigma^*$. The catenation of two strings $u$ and $v$ is the new string $uv$. Furthermore, a string $u$ is a *prefix* of $v$ if

$$\exists w \in \Sigma^* \quad (v = uw).$$

### 2.1.2 Operations on Languages

Since a language is defined as a set of strings, basic set operations can be applied to languages, namely intersection $L_1 \cap L_2$, union $L_1 \cup L_2$, complement $L_1^{co}$, and subtraction $L_2 - L_1$.

**Definition 2.1.** ***Catenation:***
*Let* $L_1, L_2 \subseteq \Sigma^*$, *then* $L_1, L_2 := \{u \in \Sigma^* \ : \ u = vw, v \in L_1, w \in L_2\}$.

In other words, $L_1 L_2$ is the result of catenation of the strings in $L_1$ with the strings in $L_2$.

**Definition 2.2.** ***Prefix-closure:***
*Let* $L \subseteq \Sigma^*$. *The prefix-closure of* $L$ *is* $\overline{L} := \{s \in \Sigma^* \mid \exists t \in \Sigma^*, \ st \in L\}$.

In other words, $\overline{L}$, the prefix-closure of $L$, consists of all the prefixes of the strings in $L$.

**Definition 2.3.** ***Natural Projection:***
*Let $\Sigma$ be an alphabet and $\Sigma_s \subseteq \Sigma$ be a subset of $\Sigma$. The natural projection onto $\Sigma_s^*$ can be defined as a map $P : \Sigma^* \longmapsto \Sigma_s^*$ in which*

$$P(\epsilon) = \epsilon$$

$$P(\sigma) = \begin{cases} \sigma, & if \ \ \sigma \in \Sigma_s \\ \epsilon, & otherwise \end{cases}$$

$$P(s\sigma) = P(s)P(\sigma) \quad \text{for} \quad s \in \Sigma^*, \sigma \in \Sigma$$

.

The following lemma is used later in this thesis.

**Lemma 2.1.** *[4] Consider $A, B \subseteq \Sigma^*$. If $A = \overline{A}$, then $A - B\Sigma^*$ must be closed.*

### 2.1.3 Automata

In order to develop computational algorithms, we need to use other modeling methods such as automata. Formally, a (deterministic) automaton is a five-tuple

$$\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$$

where $X$ is the state set, $\Sigma$ is the finite set of events (alphabet), $\eta : X \times \Sigma \longrightarrow X$ is the partial transition function, $x_0$ is the initial state and $X_m \subseteq X$ is the set of marked states. The term generator may also be used instead of automaton. In a state $x$, if a $\sigma$-transition is possible, then we write $\eta(x, \sigma)!$.
$L(\boldsymbol{G})$ and $L_m(\boldsymbol{G})$ denote the closed and marked behavior of $\boldsymbol{G}$ respectively [43]:

$$L(\boldsymbol{G}) = \{s \in \Sigma^* \mid \eta(x_0, s) \text{ is defined}\}$$

$$L_m(\boldsymbol{G}) = \{s \in L(\boldsymbol{G}) \mid \eta(x_0, s) \in X_m\}$$

The language $L(\boldsymbol{G})$ represents all the strings that can be generated in automaton $\boldsymbol{G}$ starting at the initial state $x_0$. Similarly, $L_m(\boldsymbol{G})$ is the set of all strings in $L(\boldsymbol{G})$ that end in a marked state. It is clear that $L(\boldsymbol{G})$ is prefix-closed and $L_m(\boldsymbol{G})$ is a subset of $L(\boldsymbol{G})$.

Suppose for a language $L$ and an automaton $\boldsymbol{G}$

$$L_m(\boldsymbol{G}) = L$$

We say $\boldsymbol{G}$ represents $L$. Not all languages can be represented by a finite-state automata. An example of these languages is $L = \{a^n b^n \mid n \geq 0\}$.

**Definition 2.4.** ***Regular Language*** *[19]*

*A language $L$ is called* ***regular*** *if it can be marked by a finite-state automaton.*

The following theorem states that the regularity of languages is preserved under closure, complement, intersection, union and catenation operations.

**Theorem 2.1.** *[19] Let $L_1$ and $L_2$ be regular languages. Then the languages $\overline{L_1}$, $L_1^{co}$, $L_1 \cap L_2$, $L_1 \cup L_2$ and $L_1 L_2$ are regular languages.*

**Definition 2.5.** ***Strict Subautomaton*** *[10]*

*Consider automata $\boldsymbol{G}_1 = (X_1, \Sigma_1, \eta_1, x_{0,1}, X_{m,1})$ and $\boldsymbol{G}_2 = (X_2, \Sigma_2, \eta_2, x_{0,2}, X_{m,2})$. We say $\boldsymbol{G}_1$ is a strict subautomaton of $\boldsymbol{G}_2$ and write $\boldsymbol{G}_1 \sqsubset \boldsymbol{G}_2$ if the following conditions are satisfied:*

1. *$X_1 \subseteq X_2$, $x_{0,1} = x_{0,2}$ and $X_{m,1} \subseteq X_{m,2}$;*
2. *For all $x \in X_1$ and $\sigma \in \Sigma$, if $\eta_1(x, \sigma)$ is defined, then $\eta_2(x, \sigma)$ is defined and $\eta_1(x, \sigma) = \eta_2(x, \sigma)$;*
3. *For all $s \in L(\boldsymbol{G}_2) - L(\boldsymbol{G}_1)$, there exists $s' \in \overline{s}$ such that $\eta_2(x_{0,2}, s') \notin X_1$.*

**Definition 2.6.** ***Refinement*** *[44]*

*Suppose $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ are trim and $L_m(\boldsymbol{G}_1) \subseteq L_m(\boldsymbol{G}_2)$. Then we say $\boldsymbol{G}_1$ refines $\boldsymbol{G}_2$ if for all $s, t \in L(\boldsymbol{G}_1)$, $\eta_1(x_{0,1}, s) = \eta_1(x_{0,1}, t)$ implies $\eta_2(x_{0,2}, s) = \eta_2(x_{0,2}, t)$.*

### 2.1.4 Operations on Automata

In this section we introduce some basic operation on autoamta.

**Definition 2.7.** ***Reachability*** *[43, 7]*
*Let* $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$*. A state* $x \in X$ *is called reachable if there is a path from the initial state* $x_0$ *to* $x$*. Formally,* $x$ *is reachable if there exists* $s \in L(\boldsymbol{G})$ *such that* $\eta(x_0, s) = x$*.*

**Definition 2.8.** ***Reachable Automaton***
*Automaton* $\boldsymbol{G}$ *is called a reachable automaton if every state* $x \in X$ *is reachable.*

**Definition 2.9.** ***Coreachability***
*Let* $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$*. A state* $x \in X$ *is called coreachable if there exists a path from* $x$ *to a marked state, i.e. there exists a string* $s \in \Sigma^*$ *such that* $\eta(x, s) \in X_m$*.*

**Definition 2.10.** ***Coreachable Automaton***
*Automaton* $\boldsymbol{G}$ *is called coreachable if every state* $x \in X$ *is coreachable.*

An automaton is called nonblocking if every reachable state is coreachable. In other words, if there exists a path from every reachable state to a marked state.

**Definition 2.11.** ***Nonblocking***
*Automaton* $\boldsymbol{G}$ *is called nonblocking if and only if* $\overline{L_m(\boldsymbol{G})} = L(\boldsymbol{G})$*.*

**Definition 2.12.** ***Trim***
*Automaton* $\boldsymbol{G}$ *is called a trim automaton if it is both reachable and coreachable. Trim operation is denoted by* $trim(\boldsymbol{G})$ *and defined as the reachable and coreachable subautomaton of* $\boldsymbol{G}$*.*

**Remark 2.1.** *If* $\boldsymbol{G}_1$ *and* $\boldsymbol{G}_2$ *are trim and* $\boldsymbol{G}_1$ *is a strict subautomaton of* $\boldsymbol{G}_2$*, then* $\boldsymbol{G}_1$ *refines* $\boldsymbol{G}_2$*.*

**Definition 2.13.** ***Complement***
*Consider automaton* $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$*.* $\boldsymbol{G}^{co}$ *is an automaton which generates* $\Sigma^*$

*and marks* $\Sigma^* - L_m(\boldsymbol{G}) = [L_m(\boldsymbol{G})]^{co}$.

$$L(\boldsymbol{G}^{co}) = \Sigma^*\,, L_m(\boldsymbol{G}^{co}) = \Sigma^* - L_m(\boldsymbol{G})$$

**Definition 2.14.** ***Product***

*Consider automata* $\boldsymbol{G}_1 = (X_1, \Sigma_1, \eta_1, x_{0,1}, X_{m,1})$ *and* $\boldsymbol{G}_2 = (X_2, \Sigma_2, \eta_2, x_{0,2}, X_{m,2})$. *The product of these two automata, denoted by* $\boldsymbol{G}_1 \times \boldsymbol{G}_2$, *is*

$$\boldsymbol{G}_1 \times \boldsymbol{G}_2 = \textit{The reachable part of } (X_1 \times X_2, \Sigma_1 \cap \Sigma_2, \eta(x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

*with*

$$\eta((x_1, x_2), \sigma) = \begin{cases} (\eta_1(x_1, \sigma), \eta_2(x_2, \sigma)) & if \sigma \in \Sigma_1 \cap \Sigma_2,\ \eta_1(x_1, \sigma)!\ and\ \eta_2(x_2, \sigma)! \\ undefined & otherwise \end{cases}$$

Therefore,

$$L(\boldsymbol{G}_1 \times \boldsymbol{G}_2) = L(\boldsymbol{G}_1) \cap L(\boldsymbol{G}_2)$$

$$L_m(\boldsymbol{G}_1 \times \boldsymbol{G}_2) = L_m(\boldsymbol{G}_1) \cap L_m(\boldsymbol{G}_2)$$

**Definition 2.15.** ***Synchronous Product***

*Consider automata* $\boldsymbol{G}_1 = (X_1, \Sigma_1, \eta_1, x_{0,1}, X_{m,1})$ *and* $\boldsymbol{G}_2 = (X_2, \Sigma_2, \eta_2, x_{0,2}, X_{m,2})$. *The synchronous product of these two automata,* $\boldsymbol{G}_1 \| \boldsymbol{G}_2$, *is defined as*

$$\boldsymbol{G}_1 \| \boldsymbol{G}_2 = \textit{The reachable part of } (X_1 \times X_2, \Sigma_1 \cap \Sigma_2, \eta(x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

*with*

$$\eta((x_1, x_2), \sigma) = \begin{cases} (\eta_1(x_1, \sigma), \eta_2(x_2, \sigma)) & if\ \sigma \in \Sigma_1 \cap \Sigma_2,\ \eta_1(x_1, \sigma)!\ and\ \eta_2(x_2, \sigma)! \\ (\eta_1(x_1, \sigma), x_2) & if\ \sigma \in \Sigma_1 - \Sigma_2\ and\ \eta_1(x_1, \sigma)! \\ (x_1, \eta_2(x_2, \sigma)) & if\ \sigma \in \Sigma_2 - \Sigma_1\ and\ \eta_2(x_2, \sigma)! \\ undefined & otherwise \end{cases}$$

Thus [43, 7]

$$L(\boldsymbol{G}_1 \| \boldsymbol{G}_2) = P_1^{-1}(L(\boldsymbol{G}_1)) \cap P_2^{-1}(L(\boldsymbol{G}_2))$$

$$L_m(\boldsymbol{G}_1 \| \boldsymbol{G}_2) = P_1^{-1}(L_m(\boldsymbol{G}_1)) \cap P_2^{-1}(L_m(\boldsymbol{G}_2))$$

where $P_i$ is a natural projection defined as

$$P_i : (\Sigma_1 \cup \Sigma_2)^* \longmapsto \Sigma_i^* \quad \text{(for i=1,2)}$$

**Definition 2.16.** ***Complete Automaton***
*Automaton* $\boldsymbol{G}$ *is called a complete automaton if the transition function* $\eta$ *is a total function; i.e.* $L(\boldsymbol{G}) = \Sigma^*$.

If an automaton $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$ is not complete, it can be turned into a complete automaton in the following way:

1. Add a new dump state $d$ to the state set.
2. For every $x \in X$, $\sigma \in \Sigma$, if $\eta(x, \sigma)$ is not defined, add a transition $x \xrightarrow{\sigma} d$ to the transition list.
3. For every $\sigma \in \Sigma$, add selfloops $d \xrightarrow{\sigma} d$ to the dump state.

Let $\boldsymbol{G'}$ denote the resulting automaton. Then $L(\boldsymbol{G'}) = \Sigma^*$ and $L_m(\boldsymbol{G'}) = L_m(\boldsymbol{G})$.

## 2.2 Supervisory Control

### 2.2.1 Full Observation

Let automaton $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$ be a model of the plant. Since the plant's behavior is not always admissible (legal or safe), it has to be restricted such that it satisfies certain conditions and characteristics. In the supervisory control theory of DES [32], it is assumed that the event set $\Sigma$ can be partitioned into two disjoint subsets, $\Sigma_c$ and $\Sigma_{uc}$ which are the sets of controllable and uncontrollable events respectively. Suppose that a language $E \subseteq L_m(G)$ represents the legal marked sequences. In the supervisory control theory, a supervisor $S$ is to be designed to restrict the behavior of the plant $\boldsymbol{G}$ to the legal behavior $E$. The configuration of supervisory

control is shown in Figure 2.1.

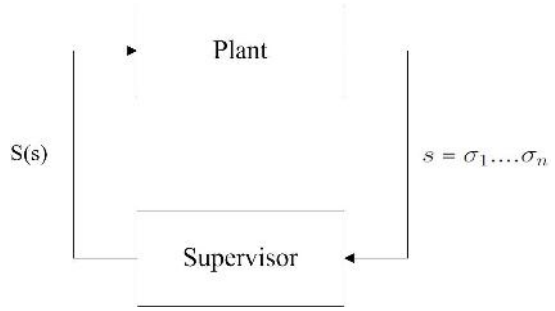



Figure 2.1: Supervisory control configuration.

The supervisor is only allowed to disable controllable events. A supervisory control for DES $\boldsymbol{G}$ can be defined as a map

$$S : \Sigma^* \to \Gamma_\Sigma$$

where

$$\Gamma_\Sigma = \{\gamma \in Pwr(\Sigma) \mid \Sigma_{uc} \subseteq \gamma\}$$

denotes the set of all control patterns in $\Sigma$. In this context, for $s \in \Sigma^*$, $S(s)$ is the set of events enabled by the supervisor. Note that $S$ cannot disable uncontrollable events, i.e. $\Sigma_{uc} \subseteq S(s)$. Let $S/\boldsymbol{G}$ represent the plant $\boldsymbol{G}$ under the supervision of $S$ (or the closed-loop system) and $L(S/\boldsymbol{G}) \subseteq L(\boldsymbol{G})$ the closed behavior of the closed-loop system. The language generated by the system under supervision $L(S/\boldsymbol{G})$ is defined as follows.

1. $\epsilon \in L(S/\boldsymbol{G})$
2. If $s \in L(S/\boldsymbol{G}), \sigma \in S(s)$, and $s\sigma \in L(\boldsymbol{G})$, then $s\sigma \in L(S/\boldsymbol{G})$

The marked behavior of the system under supervision is defined as $L_m(S/\boldsymbol{G}) = L(S/\boldsymbol{G}) \cap L_m(\boldsymbol{G})$.

**Problem 2.1.** ***Nonblocking Supervisory Control Problem (NSCP)*** *[32]*
*Consider an automaton* $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$ *with* $\Sigma = \Sigma_c \cup \Sigma_{uc}$. *Let the legal marked behavior of the plant* $E \subseteq L_m(\boldsymbol{G})$ *be nonempty. Find a supervisor* $S$ *such that*

1. $L_m(S/\boldsymbol{G}) \subseteq E$
2. $L(S/\boldsymbol{G}) = \overline{L_m(S/\boldsymbol{G})}$

The Nonblocking Supervisory Control Problem (NSCP) does not necessarily have a unique solution in general. Consider

$$\mathcal{S} := \{S |\, S \text{ solves the NSCP}\}.$$

A supervisor $S$ is called maximally permissive if and only if

$$\forall S' \in \mathcal{S} : L_m(S'/\boldsymbol{G}) \subseteq L_m(S/\boldsymbol{G}),\ L(S'/\boldsymbol{G}) \subseteq L(S/\boldsymbol{G}).$$

In order to identify the solutions of NSCP we need the following definitions.

**Definition 2.17.** ***Controllability*** *[32]*
*A language* $K \in \Sigma^*$ *is said to be controllable with respect to a closed language* $L(\boldsymbol{G})$ *and* $\Sigma_{uc}$ *if*

$$\overline{K}\Sigma_{uc} \cap L(\boldsymbol{G}) \subseteq \overline{K}$$

It is easy to show that the set of controllable sublanguages is closed with respect to the union operation; it is nonempty since the empty language $\emptyset$ is controllable. Thus, there exists a supremal element for the set of controllable sublanguages of a given language $K$. $SupC(K)$ denotes the supremal controllable sublanguage of $K$ with respect to $L(\boldsymbol{G})$ and $\Sigma_{uc}$.

**Definition 2.18.** $L_m(\boldsymbol{G})$***-closure*** *[32]*
*A language* $K \subseteq L_m(\boldsymbol{G})$ *is* $L_m(\boldsymbol{G})$*-closed if*

$$K = \overline{K} \cap L_m(\boldsymbol{G})$$

The set of $L_m(\boldsymbol{G})$-closed sublanguages of $K$ denoted by $R_{\boldsymbol{G}}(K)$ has also a supremal element since it is closed under arbitrary unions and $\emptyset$ is $L_m(\boldsymbol{G})$-closed. The supremal $L_m(\boldsymbol{G})$-closed sublanguages of $K$ is denoted by $SupR_{\boldsymbol{G}}(K)$ and can be calculated as

$$SupR_{\boldsymbol{G}}(K) = K - (L_m(\boldsymbol{G}) - K)\Sigma^*.$$

**Lemma 2.2.** *[32] If language $K$ is $L_m(\boldsymbol{G})$-closed, then $SupC(K)$ will be $L_m(\boldsymbol{G})$-closed.*

Theorem 2.2 characterizes the solutions to the nonblocking supervisory control problem (NSCP) defined in Problem 2.1.

**Theorem 2.2.** *[32] Consider DES $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$, and let $\Sigma_{uc} \subseteq \Sigma$ be the set of uncontrollable events and $K \subseteq E \subseteq L_m(\boldsymbol{G})$ be a nonempty language. Then NSCP is solvable with $L_m(S/\boldsymbol{G}) = K$ and $L(S/\boldsymbol{G}) = \overline{K}$ if and only if*

1. *$K$ is controllable with respect to $\boldsymbol{G}$ and $\Sigma_{uc}$;*

2. *$K$ is $L_m(\boldsymbol{G})$-closed.*

The supremal $L_m(\boldsymbol{G})$-closed and controllable sublanguage of a given language $K$ provides the maximally permissive solution to NSCP. Since the supremal controllable sublanguage of a $L_m(\boldsymbol{G})$-closed language remains $L_m(\boldsymbol{G})$-closed, the optimal solution of this problem can be computed by obtaining the $L_m(\boldsymbol{G})$-closed sublanguage of $K$ and then computing the supremal controllable sublanguage. In other words:

$$SupR_{\boldsymbol{G}}C(K) = SupC(SupR_{\boldsymbol{G}}(K))$$

### 2.2.2 Control under Partial Event Observation

Now consider the supervisory control problem in case that only a subset of events which the plant can generate can be observed by the supervisor. In this case, we can partition the event set into $\Sigma = \Sigma_o \cup \Sigma_{uo}$ where $\Sigma_o$ and $\Sigma_{uo}$ are the sets of observable and unobservable events respectively.

The events in the plant may become unobservable because of the limitations of the sensors connected to the plant or because some events at some locations cannot be observed at other locations.

Let the natural projection $P$ be defined as $P : \Sigma^* \to \Sigma_o^*$. If two strings $s_1$ and $s_2$ have the same projection, i.e. $P(s_1) = P(s_2)$, the supervisor cannot tell them apart. Therefore, the supervisor has to issue the same control action for lookalike sequences.

**Problem 2.2.** ***Nonblocking Supervisory Control Problem with Partial Observation (NSCP-PO)*** *[11, 25]*
*Consider DES* $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$ *with* $\Sigma = \Sigma_o \cup \Sigma_{uo}$ *and* $\Sigma = \Sigma_c \cup \Sigma_{uc}$. *Also, let* $E \subseteq L_m(\boldsymbol{G})$ *be a nonempty legal marked behavior. Find a supervisor* $S$ *such that*

1. $L_m(S/\boldsymbol{G}) \subseteq E$
2. $L(S/\boldsymbol{G}) = \overline{L_m(S/\boldsymbol{G})}$

In order to characterize the solutions of NSCP-PO, we need to define *observability*.

**Definition 2.19.** ***Observability*** *[11, 25]*
*A language* $K$ *is called* $(L(\boldsymbol{G}), P)$*-observable if and only if*

$$\forall s, s' \in \Sigma^* : [s' \in \overline{K} \text{ and } s'\sigma \in L(\boldsymbol{G}) \text{ and } s\sigma \in \overline{K} \text{ and } P(s) = P(s')] \Rightarrow s'\sigma \in \overline{K}$$

In other words, in case of an observable language, the same control action is used for any two strings in the language that have the same projection.

**Theorem 2.3.** *[11, 25] Let DES* $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$ *with* $\Sigma = \Sigma_o \cup \Sigma_{uo}$ *and* $\Sigma = \Sigma_c \cup \Sigma_{uc}$. *Also, let* $K \subseteq E \subseteq L_m(\boldsymbol{G})$ *be a nonempty legal marked behavior. Then NSCP-PO is solvable with* $L_m(S/\boldsymbol{G}) = K$ *and* $L(S/\boldsymbol{G}) = \overline{K}$ *if and only if*

1. $K$ *is controllable with respect to* $\boldsymbol{G}$ *and* $\Sigma_{uc}$;
2. $K$ *is* $L_m(\boldsymbol{G})$*-closed;*
3. $K$ *is* $(L(\boldsymbol{G}), P)$*-observable.*

Theorem 2.3 characterizes all the solutions of the NSCP-PO problem. Since the set of $(L(\boldsymbol{G}), P)$-observable sublanguages of a given language $K$ is not closed under union operation, a supremal element may not exist for this set. Consequently, the set of controllable, $L_m(\boldsymbol{G})$-closed and $(L(\boldsymbol{G}), P)$-observable sublanguages of a language $K$ does not have a supremal element. Thus unlike the full observation case, an optimal (minimally restrictive) solution may not exist for NSCP-PO in general.
In order to characterize a suboptimal solution for NSCP-PO, we can define a new property which is stronger than observability, and is called *normality*.

**Definition 2.20.** ***Normality*** *[11, 25]*
*A language $K$ is called $(L(\boldsymbol{G}), P)$-normal if and only if*

$$\overline{K} = L(\boldsymbol{G}) \cap P^{-1}(P\overline{K})$$

This property states that the language $K$ is $(L(\boldsymbol{G}), P)$-normal if and only if the legality of its sequences can be determined from its projection $P\overline{K}$ and the closed behavior of the system $L(\boldsymbol{G})$. A main difference between observability and normality is that in a normal language, by watching the projection of a string generated by plant we can determine if the string belongs to the language or not. In general this is not the case for observable languages. Normality always implies observability (but the converse is not always true). If all controllable events are observable, then any controllable and observable language is normal.

Let $E$ be a language such that $E \subseteq L(\boldsymbol{G})$. The class of normal sublanguages of $E$ can be defined as $\overline{N}(E, L(\boldsymbol{G})) = \{K \subseteq E \,|\, \overline{K} = L(\boldsymbol{G}) \cap P^{-1}P(\overline{K})\}$. $\overline{N}(E, L(\boldsymbol{G}))$ is nonempty ($\emptyset \in \overline{N}(E, L(\boldsymbol{G}))$) and closed under union. Therefore it has a supremal element.

The following lemma provides a formula for calculating the supremal normal sublanguage of a closed language.

**Lemma 2.3.** *[4] Let $E$ be a closed language and $E \subseteq L(\boldsymbol{G})$, thus*

$$Sup\overline{N}(E, L(\boldsymbol{G})) = E - P^{-1}P(L(\boldsymbol{G}) - E)\Sigma^*$$

**Lemma 2.4.** *[10] If a language $E \subseteq L_m(\boldsymbol{G})$ is $L_m(\boldsymbol{G})$-closed, then the supremal normal sublanguage of $E$ denoted by $Sup\overline{N}(E, L(\boldsymbol{G}))$ is also $L_m(\boldsymbol{G})$-closed.*

Since the class of normal languages are also observable, the class of controllable, $L_m(\boldsymbol{G})$-closed and $(L(\boldsymbol{G}), P)$-normal sublanguages of a given legal marked behavior $E$ characterizes a subset of solutions to NSCP-PO. Let $Sup\overline{N}R_{\boldsymbol{G}}C(E)$ denotes the supremal element of the aforementioned class of languages. Then it provides the minimally restrictive solution among those based on normal sublanguages.

## 2.3 Robust Supervisory Control

### 2.3.1 Full Observation

Conventional (non-robust) supervisory control assumes that the plant model is known and may not change. On the contrary, this assumption does not hold all the time. For instance, there might be some uncertainties in the system modeling or the system may change over time. Moreover, we can also consider a system that has to perform different tasks so that it may have various configurations. Robust supervisory control is one way to deal with the aforementioned issues.

Among several methods for formulating the robust supervisory control problem, we are interested in the setup in which the system model belongs to a finite set of possible models.

**Problem 2.3.** ***Robust Nonblocking Supervisory Control Problem (RNSCP)*** *[3, 35].*
*Consider $n$ DES models $\boldsymbol{G}_i = (X_i, \Sigma_i, \eta_i, x_{0,i}, X_{m,i})$ where $i \in I = \{1, 2, ..., n\}$, and the corresponding legal behaviors $E_i$. The set of plant models is $\mathcal{G}$= $\{\boldsymbol{G}_1, ..., \boldsymbol{G}_n\}$. Let the event set, and the controllable and uncontrollable sets of events in $\boldsymbol{G}_i$ be denoted by $\Sigma_i$, $\Sigma_{c,i}$ and $\Sigma_{uc,i}$ respectively. It is assumed that all plant models agree on the controllability of events, that is $\Sigma_{c,i} \cap \Sigma_{uc,j} = \emptyset$ for all $i, j \in \{1, ..., n\}$. Furthermore, suppose language $K_i$ denotes the design specification for plant model $\boldsymbol{G}_i$ and hence*

*$E_i = K_i \cap L_m(\boldsymbol{G}_i)$ denotes the corresponding legal marked behavior. In RNSCP, supervisor $S$ should be designed such that $L_m(S/\boldsymbol{G}_i) \subseteq E_i$ and $\overline{L_m(S/\boldsymbol{G}_i)} = L(S/\boldsymbol{G}_i)$ for all $i \in I$.*

The solutions to RNSCP are given in [35]. Before discussing these solutions, we review the definition of $\boldsymbol{G}$-nonblocking languages.

**Definition 2.21.** ***G*-nonblocking** *[35]*
*A language $K \subseteq \Sigma^*$ is called $\boldsymbol{G}$-nonblocking if $\overline{K \cap L_m(\boldsymbol{G})} = \overline{K} \cap L(\boldsymbol{G})$.*

It has been shown in [35] that the class of $\boldsymbol{G}$-nonblocking sublanguages of a given language $E \subseteq \Sigma^*$ is nonempty and has a supremal element. Furthermore, [35] proves that this supremal sublanguage can be obtained from

$$SupNb_{\boldsymbol{G}}(E) = E - (\overline{E} \cap L(\boldsymbol{G}) - \overline{E \cap L_m(\boldsymbol{G})})\Sigma^* \tag{2.1}$$

It follows from (2.1) that for a finite-state automaton $\boldsymbol{G}$, if the legal behavior $E$ is a regular language, then the supremal $\boldsymbol{G}$-nonblocking sublanguage is also regular and can be represented with a finite-state automaton. In Chapter 3, a computational procedure will be introduced to build an automaton marking the supremal $\boldsymbol{G}$-nonblocking sublanguage.

**Theorem 2.4.** *[35] Consider the Robust Nonblocking Supervisory Control Problem (RNSCP). Let $\boldsymbol{G}$ be an automaton such that $L_m(\boldsymbol{G}) = \bigcup_{i\in I} L_m(\boldsymbol{G}_i)$ and $L(\boldsymbol{G}) = \bigcup_{i\in I} L(\boldsymbol{G}_i)$. Furthermore, let $\Sigma = \bigcup_{i\in I} \Sigma_i$ and $\Sigma_{uc} = \bigcup_{i\in I} \Sigma_{uc,i}$ and define $E$ as*

$$E = \bigcap_{i\in I} (E_i \cup (\Sigma^* - L_m(\boldsymbol{G}_i))) \cap L_m(\boldsymbol{G}) \tag{2.2}$$

*If there exists a nonempty sublanguage $K \subseteq E$ that satisfies the following conditions:*

1. *$K$ is controllable with respect to $\boldsymbol{G}$ ($\overline{K}\Sigma_{uc} \cap L(\boldsymbol{G}) \subseteq \overline{K}$);*
2. *$K$ is $L_m(\boldsymbol{G})$-closed ($\overline{K} \cap L_m(\boldsymbol{G}) = K$);*
3. *$K$ is $\boldsymbol{G}_i$-nonblocking ($\overline{K \cap L_m(\boldsymbol{G}_i)} = \overline{K} \cap L(\boldsymbol{G}_i)$ for all $i \in I$).*

*Then RNSCP has a solution $S$ with $K = L_m(S/\boldsymbol{G})$ and $\overline{K} = L(S/\boldsymbol{G})$. Conversely, if $S$ is a solution of RNSCP, $K = L_m(S/\boldsymbol{G})$ satisfies conditions (1), (2) and (3).*

Theorem 2.4 characterizes the set of solutions of RNSCP. Note that the legal language $E$ in (2.2) can be rewritten in terms of design specification languages $K_i$ as

$$E = \bigcap_i (K_i \cup (\Sigma^* - L_m(\boldsymbol{G}_i))) \cap L_m(\boldsymbol{G}) \tag{2.3}$$

If we define $RCNb(E, \mathcal{G})$ as the set of $L_m(\boldsymbol{G})$-closed, controllable and $\boldsymbol{G}_i$-nonblocking sublanguages of $E$ (for all $\boldsymbol{G}_i \in \mathcal{G}$), this set is nonempty and closed under arbitrary unions. Thus, it has a supremal element denoted by $E^*$,

$$E^* = SupRCNb(E, \mathcal{G}) \tag{2.4}$$

Then $E^*$ is the maximally permissive solution of RNSCP.

The robust supervisory control described in RNSCP has applications in various problems. Fault recovery problems can be solved as special cases of robust control [34],[35]. In this case, the plant models $\boldsymbol{G}_N, \boldsymbol{G}_{NF_1} ..., \boldsymbol{G}_{NF_m}$ represent the plant dynamics in normal and $m$ normal-faulty modes, and each mode of the system has its own design specification. The objective is to design a supervisor such that the system under supervision meets the design specifications in each mode while remaining nonblocking in all modes.

Another problem that can be solved as a especial case of robust control is the problem of supervisory control of DES with multiple sets of marked states [14].

**Problem 2.4.** ***Nonblocking Supervisory Control Problem with Multiple sets of Marked states (NSCP-MM)***

*Consider automaton $\boldsymbol{G}$ with multiple sets of marked states $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_{m,1}, X_{m,2}, ..., X_{m,n})$. Let $L_{m,i}(\boldsymbol{G}) = \{s \in L(\boldsymbol{G}) \mid \eta(x_0, s) \in X_{m,i}\}$ denote the marked behavior with respect to $X_{m,i}$ and furthermore $E_i \subseteq L_{m,i}(\boldsymbol{G})$ the corresponding legal language. Find a supervisor $S$ such that $L_{m,i}(S/\boldsymbol{G}) \subseteq E_i$ and $\overline{L_{m,i}(S/\boldsymbol{G})} = L(S/\boldsymbol{G})$.*

In this problem, nonblocking property is desired with respect to multiple sets of marked states.

The problem of supervisory control with multiple sets of marked states can be converted to an equivalent robust nonblocking supervisory control (RNSCP) [9] if we define automata $\boldsymbol{G}_1, ..., \boldsymbol{G}_n$ as $\boldsymbol{G}_i = (X, \Sigma, \eta, x_0, X_{m,i})\,(i = 1, ..., n)$ and design a robust supervisor $S$ such that $L_m(S/\boldsymbol{G}_i) \subseteq E_i$ and $\overline{L_m(S/\boldsymbol{G}_i)} = L(S/\boldsymbol{G}_i)$.

As an example, consider the propulsion system with two engines shown in Figure 2.2. The system has three modes of operation: (1) both engines off, (2) engine E1 on, and (3) engine E2 on. The supervisor (controller) should open or close the valves in such a way that at anytime the propulsion system can move from one mode to another. Each of the three modes corresponds to a set of marked states in the NSCP-MM problem. As an instance for safety specification, both engines should not fire simultaneously. We will revisit a more complete version of this problem which also involves fault recovery later in Chapter 3.

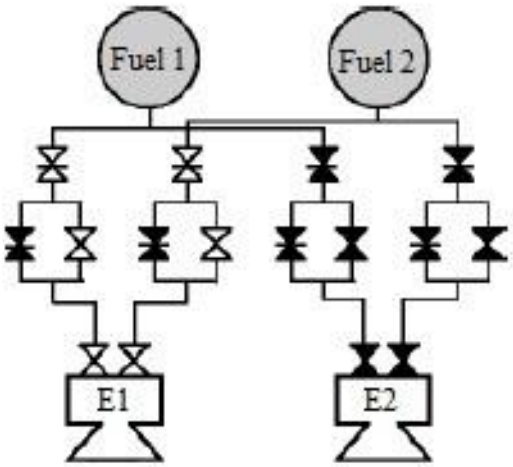



Figure 2.2: Supervisory control with multiple sets of marked states.

### 2.3.2 Partial Observation

As we discussed in Section 2.3.1, the notion of robust supervisory control of discrete event systems has been introduced in order to deal with the situations in which the plant's model and dynamics are unknown. In Section 2.3.1, we studied RNSCP when all the events were assumed to be observable. This problem can be extended to the case of partial event observation. In the following we review the robust nonblocking supervisory control problem in case of partial observation and its solution.

**Problem 2.5.** ***Robust Nonblocking Supervisory Control Problem with Partial Observation (RNSCP-PO)*** *[35].*
*Consider $n$ DES models $\boldsymbol{G}_i = (X_i, \Sigma_i, \eta_i, x_{0,i}, X_{m,i})$ where $i \in I = \{1, 2, ..., n\}$, and the corresponding legal behaviors $E_i$. The set of plant models is $\mathcal{G}= \{\boldsymbol{G}_1, ..., \boldsymbol{G}_n\}$. Let the event set, and the controllable and uncontrollable sets of events in $\boldsymbol{G}_i$ be denoted by $\Sigma_i$, $\Sigma_{c,i}$ and $\Sigma_{uc,i}$ respectively. Similarly, let the observable and unobservable sets of events in $\boldsymbol{G}_i$ be denoted by $\Sigma_{o,i}$ and $\Sigma_{uo,i}$ respectively. It is assumed that all plant models agree on the controllability and observability of events, that is $\Sigma_{c,i} \cap \Sigma_{uc,j} = \emptyset$ and $\Sigma_{o,i} \cap \Sigma_{uo,j} = \emptyset$ for all $i, j \in \{1, ..., n\}$. Furthermore, suppose language $K_i$ denotes the design specification for plant model $\boldsymbol{G}_i$ and hence $E_i = K_i \cap L_m(\boldsymbol{G}_i)$ denotes the corresponding legal marked behavior. In RNSCP-PO, supervisor $S$ should be designed such that $L_m(S/\boldsymbol{G}_i) \subseteq E_i$ and $\overline{L_m(S/\boldsymbol{G}_i)} = L(S/\boldsymbol{G}_i)$ for all $i \in I$.*

The solution to RNSCP with partial observation (RNSCP-PO) is given in [35].

**Theorem 2.5.** *[35] Consider the Robust Nonblocking Supervisory Control Problem with Partial Observation (RNSCP-PO). Let $\boldsymbol{G}$ be an automaton such that $L_m(\boldsymbol{G}) = \bigcup_{i \in I} L_m(\boldsymbol{G}_i)$ and $L(\boldsymbol{G}) = \bigcup_{i \in I} L(\boldsymbol{G}_i)$. Furthermore, let $\Sigma = \bigcup_{i \in I} \Sigma_i$, $\Sigma_c = \bigcup_{i \in I} \Sigma_{c,i}$, and $\Sigma_o = \bigcup_{i \in I} \Sigma_{o,i}$ and define $E$ as*

$$E = \bigcap_i (K_i \cup (\Sigma^* - L_m(\boldsymbol{G}_i))) \cap L_m(\boldsymbol{G}) \tag{2.5}$$

*If there exists a nonempty sublanguage $K \subseteq E$ that satisfies the following conditions:*

1. *$K$ is controllable with respect to $\boldsymbol{G}$: $\overline{K}\Sigma_{uc} \cap L(\boldsymbol{G}) \subseteq \overline{K}$;*

2. *$K$ is $L_m(\boldsymbol{G})$-closed: $\overline{K} \cap L_m(\boldsymbol{G}) = K$;*

3. *$K$ is $(L(\boldsymbol{G}), P)$-observable;*

4. *$K$ is $\boldsymbol{G}_i$-nonblocking: $\overline{K \cap L_m(\boldsymbol{G}_i)} = \overline{K} \cap L(\boldsymbol{G}_i)$ for all $i \in I$,*

*then RNSCP-PO has a solution $S$ with $K = L_m(S/\boldsymbol{G})$ and $\overline{K} = L(S/\boldsymbol{G})$. Conversely, if supervisor $S$ is a solution of RNSCP-PO, then $K = L_m(S/\boldsymbol{G})$ satisfies conditions (1) to (4).*

If we define $RCONb(E, \mathcal{G})$ as the set of $L_m(\boldsymbol{G})$-closed, controllable, observable and $\boldsymbol{G}_i$-nonblocking sublanguages of $E$ for all $\boldsymbol{G}_i \in \mathcal{G}$, this set may not have a supremal element since the class of observable sublanguages of $E$ is not closed under the union operation. Therefore, unlike RNSCP with full observation, RNSCP with partial observation may not have a maximally permissive (optimal) solution.

Next we restate Theorem 2.5 by replacing the normality property instead of observability.

**Theorem 2.6.** *Consider RNSCP-PO in Problem 2.5. Let $\boldsymbol{G}$ be an automaton such that $L_m(\boldsymbol{G}) = \bigcup_{i\in I} L_m(\boldsymbol{G}_i)$ and $L(\boldsymbol{G}) = \bigcup_{i\in I} L(\boldsymbol{G}_i)$. Furthermore, let $\Sigma = \bigcup_{i\in I} \Sigma_i$, $\Sigma_c = \bigcup_{i\in I} \Sigma_{c,i}$, and $\Sigma_o = \bigcup_{i\in I} \Sigma_{o,i}$ and define $E$ as*

$$E = \bigcap_i (K_i \cup (\Sigma^* - L_m(\boldsymbol{G}_i))) \cap L_m(\boldsymbol{G}) \tag{2.6}$$

*If there exists a nonempty sublanguage $K \subseteq E$ that satisfies the following conditions:*

1. *$K$ is controllable with respect to $\boldsymbol{G}$: $\overline{K}\Sigma_{uc} \cap L(\boldsymbol{G}) \subseteq \overline{K}$;*

2. *$K$ is $L_m(\boldsymbol{G})$-closed: $\overline{K} \cap L_m(\boldsymbol{G}) = K$;*

3. *$K$ is $(L(\boldsymbol{G}), P)$-normal: $\overline{K} = L(\boldsymbol{G}) \cap P^{-1}(P\overline{K})$;*

4. *$K$ is $\boldsymbol{G}_i$-nonblocking: $\overline{K \cap L_m(\boldsymbol{G}_i)} = \overline{K} \cap L(\boldsymbol{G}_i)$ for all $i \in I$,*

*Then RNSCP-PO has a solution $S$ with $K = L_m(S/\boldsymbol{G})$ and $\overline{K} = L(S/\boldsymbol{G})$.*

Now, if we define $R\overline{N}CNb(E, \mathcal{G})$ as the set of $L_m(\boldsymbol{G})$-closed, normal, controllable and $\boldsymbol{G}_i$-nonblocking sublanguages of $E$ for all $\boldsymbol{G}_i \in \mathcal{G}$, this set is closed under union operation and also nonempty. Therefore, it has a supremal element denoted by $E^*$, i.e. $E^* = Sup R\overline{N}CNb(E, \mathcal{G})$.

## 2.4 Computational Procedures

In supervisory control, discrete-event systems are modeled by automata. The synthesis of the supervisor is the main challenge in solving supervisory control problems. In general, design specifications and system requirements are not represented as automata since they are mostly given using natural language. So, another key step in design problem is to find an automaton representation for the system specifications. We saw in previous sections that in supervisory control with full event observation, the maximally permissive supervisor is described in terms of a supremal sublanguage of design specification. In the case of partial event observation, where a maximally permissive solution may not exist but suitable solution in terms of a supremal sublanguage (based on the property of normality) can be identified.

In order to develop computational procedures to find these supremal sublanguages we need to review some important definitions from lattice theory.

**Definition 2.22.** ***Poset*** *[27]*
*Let $X$ be a set and $\leq$ be a binary relation on $X$. Then $(X, \leq)$ is called a poset if:*

1. *reflexive:* $(\forall x \in X) \quad x \leq x$
2. *transitive:* $(\forall x_1, x_2, x_3 \in X) \quad x_1 \leq x_2 \ \ \& \ \ x_2 \leq x_3 \Rightarrow x_1 \leq x_3$
3. *antisymmetric:* $(\forall x_1, x_2 \in X) \ \ x_1 \leq x_2 \ \ \& \ \ x_2 \leq x_1 \Rightarrow x_1 = x_2$

**Definition 2.23.** ***Lattice*** *[27]*

*A lattice is a poset $L$ in which the meet and join ($\wedge$ and $\vee$) of any two elements always exist.*

If the least upper bound and greatest lower bound of each subset in the lattice always exist, then the lattice is called a *complete* lattice.

**Definition 2.24.** ***Upper semilattice*** *[27]*

*An upper semilattice is a poset in which the join operation ($\vee$) of any two element always exist.*

Similarly, if an upper semilattice has the least upper bound, it is called a *complete upper semilattice.*

Next, suppose $Pwr(\Sigma^*)$ denotes the set of all sublanguages of $\Sigma^*$. Then, $(Pwr(\Sigma^*), \leq$ ) with $\wedge, \vee$ forms a complete lattice.

Now, let $L$ be a complete lattice and $S \subseteq L$ be a complete upper semilattice under the join operation $\vee$ of $L$. For every $z \in L$, define

$$z^{\uparrow} := sup\{x | x \in S,\ x \leq z\}$$

Before we proceed to introduce the computational algorithms for calculating the supremal element, we need to define some properties of the operators.

**Definition 2.25.** ***Monotone***

*Let $(L, \leq)$ be a poset. An operator $\varphi : L \to L$ is called monotone if and only if*

$$(\forall x_1, x_2 \in L)\ x_1 \leq x_2 \Rightarrow \varphi(x_1) \leq \varphi(x_2)$$

**Definition 2.26.** ***Contractive***

*Let $(L, \leq)$ be a poset. An operator $\varphi : L \to L$ is contractive if and only if*

$$\forall x \in L :\ \varphi(x) \leq x$$

**Definition 2.27.** ***Fixed point***

*Let $(L, \leq)$ be a poset. $x \in L$ is called a fixed point of $\varphi$ if $\varphi(x) = x$.*

**Theorem 2.7.** *[17] Let $L$ be a complete lattice and $S \subseteq L$ be a complete upper semilattice under the join operation $\vee$ of $L$. Suppose operator $\varphi : L \to L$ has the following properties:*

1. *$\varphi(.)$ is monotone,*
2. *$\varphi(.)$ is contractive,*
3. *$S$ is the set of fixed points of $\varphi$.*

*Let $z \in L$ and assume the recursion*

$$z_0 = z,$$
$$z_{k+1} = \varphi(z_k)$$

*terminates in $k^*$ steps where $k^* \geq 0$ is an integer. Then*

$$\forall k \geq k^* : \quad z^{\uparrow} = sup\{x | x \in S, \ \ x \leq z\} = z_k$$

In [44] it is shown that $SupC(L)$, the supremal controllable sublanguage of $L$ with respect to $M$ and $\Sigma_{uc}$, is the largest fixed point of an operator $\Omega : Pwr(\Sigma^*) \to Pwr(\Sigma^*)$ with

$$\Omega(K) = L \cap sup\{T : \overline{T}\Sigma_{uc} \cap M \subseteq \overline{K}\}$$

When $L$ and $M$ are regular languages, the iteration defined by

$$\begin{aligned} K_0 &= L, \\ K_{j+1} &= \Omega(K_j) \end{aligned} \tag{2.7}$$

is known to converge in a finite number of steps to $SupC(L)$.

Many computational algorithms for calculating other supremal sublanguages involve iterative procedures to find fixed points of suitable operators. These procedures are implemented in such a way that in each iteration of the algorithm, some transitions and/or states which violate the desired property are removed from an automaton. In such algorithms, the finite-state automaton becomes smaller (or it remains at the

same size) in each iteration. Hence, the finite convergence follows immediately. The main challenge is the careful choice of the initial automaton which has to have sufficient information about the strings in the languages being examined. As an example, we go over the computational algorithm given in [44] for computing the supremal controllable sublanguage.

Consider an automaton $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$ with $\Sigma = \Sigma_c \cup \Sigma_{uc}$ and assume the nonempty legal marked behavior of the plant $E \subseteq L_m(\boldsymbol{G})$ is regular. Since $E$ is a regular language, we can represent it as a finite-state automaton $\boldsymbol{E}$ where $L_m(\boldsymbol{E}) = E$ and $L(\boldsymbol{E}) = \overline{E}$. In [44] $\boldsymbol{H} = \boldsymbol{G} \times \boldsymbol{E}$ which also marks $E$ is the initial automaton for computing the supremal controllable sublanguage of $E$ (with respect to $L(\boldsymbol{G})$ and $\Sigma_{uc}$). According to Definition 2.6, it is obvious that $\boldsymbol{H}$ refines $\boldsymbol{G}$. The following algorithm uses $\boldsymbol{G}$ and $\boldsymbol{E}$ to build a trim automaton, say $\boldsymbol{K}$, such that $\boldsymbol{K}$ is a strict subautomaton of $\boldsymbol{E}$ and refines $\boldsymbol{E}$, and marks $SupC(E)$. Let us refer to this procedure as $supcon : \boldsymbol{K} = supcon(\boldsymbol{E}, \boldsymbol{G})$.

**Algorithm 2.1.** ***Algorithm for computing*** $supcon(\boldsymbol{E}, \boldsymbol{G})$

1. *$i = 0$*
2. *Construct $\boldsymbol{H} = (X_h, \Sigma, \eta_h, x_{h,0}, X_{h,m})$ as $\boldsymbol{H} = \boldsymbol{G} \times \boldsymbol{E}$. Therefore, $L_m(\boldsymbol{H}) = E$, $L(\boldsymbol{H}) = \overline{E}$ and $\boldsymbol{H}$ refines $\boldsymbol{G}$.*
3. *Compare $\boldsymbol{H}$ and $\boldsymbol{G}$. If $\eta(x, \sigma)$ is defined for $\sigma \in \Sigma_{uc}$ and $\eta_h(x, \sigma)$ is not defined and $x \in X_h$, then simply remove state $x$ and all of its associated transitions from automaton $\boldsymbol{H}$. Then trim $\boldsymbol{H}$.*
4. *Repeat step 3 for all the states of $\boldsymbol{H}$ until there is no state or transition removed in step 3.*
5. *Name the final automaton $\boldsymbol{K}$. $\boldsymbol{K}$ marks $SupC(E)$.*

Algorithm 2.1 implements (2.7) and computes the supremal controllable sublanguage of $E$ with respect to $L(\boldsymbol{G})$ and $\Sigma_{uc}$. In each step of this algorithm the states

and transitions which violate the controllability property are removed and the final automaton marks the supremal controllable sublanguage of $E$.

Next, we present another example of computational procedures which is computing the supremal normal sublanguage. In order to calculate the supremal normal sublanguage, a closed formula is provided in the literature when the legal behavior $E$ is a closed language. However, the language $E$ need not to be closed. In this case, the supremal normal sublanguage of this language can be calculated using the following recursive operator. Let operator $\Psi : Pwr(\Sigma^*) \rightarrow Pwr(\Sigma^*)$ where $\Psi(Z) = Z \cap Sup\overline{N}(\overline{Z}, L(\boldsymbol{G})),\ Z \subseteq \Sigma^*$. Now, consider the following iterative procedure

$$\begin{aligned} Z_0 &:= E \\ Z_i &:= \Psi(Z_{i-1}),\ \ i \geq 1 \end{aligned} \tag{2.8}$$

Since $\overline{Z}$ is closed in operator $\Psi$, we can use the formula provided in Lemma 2.3, thus $\Psi$ can be rewritten as $\Psi(Z) = Z \cap (\overline{Z} - P^{-1}P(L(\boldsymbol{G}) - \overline{Z})\Sigma^*)$.

In dealing with supervisory control problem under partial observation, it is useful to build a projected model $\boldsymbol{G}_p = P(\boldsymbol{G})$ which marks and generates projected languages $L_m(\boldsymbol{G}_p) = P(L_m(\boldsymbol{G}))$ and $L(\boldsymbol{G}_p) = P(L(\boldsymbol{G}))$. The procedure is described in [19] and is omitted here for brevity. It is important to note that $\boldsymbol{G}_p$ can be regarded as an observer for $\boldsymbol{G}$. That is because for every sequence $s \in L(\boldsymbol{G})$, there exists a projected sequence in $P(s) \in L(\boldsymbol{G}_p)$ such that the state in $\boldsymbol{G}_p$ corresponds to a subset of states of $\boldsymbol{G}$ and this subset is the state estimate of $\boldsymbol{G}$ based on observation $P(s)$.

Furthermore, the inverse projection of an automaton $\boldsymbol{G}_p$, denoted by $P^{-1}(\boldsymbol{G}_p)$, can be computed by adding self-loops of all unobservable events $\sigma \in \Sigma_{uo}$ to every state of $\boldsymbol{G}_p$. If we call the resulting automaton $\boldsymbol{G}_s$, it is clear that $L(\boldsymbol{G}_s) = P^{-1}[L(\boldsymbol{G}_p)]$ and $L_m(\boldsymbol{G}_s) = P^{-1}[L_m(\boldsymbol{G}_p)]$.

# Chapter 3

# Robust Nonblocking Supervisory Control Problem (RNSCP)

Robust control has been introduced to supervisory control of discrete-event systems to deal with plant's model and dynamics uncertainties. In Chapter 2, we discussed the RNSCP and reviewed its solution. In this chapter, we develop a computational procedure to obtain a maximally permissive solution for the RNSCP. Moreover, in the process, the finite convergence of the procedure will be proved.

The organization of this chapter is as follows. The main problem is formulated in Section 3.1. Section 3.2 proposes computational procedures for constructing the union and legal behavior automata as well as algorithms for obtaining the supremal $L_m(\boldsymbol{G})$-closed and $\boldsymbol{G}_i$-nonblocking sublanguages. The computational procedure for robust nonblocking supervisory control problem is discussed in Section 3.3. Section 3.4 provides two illustrative examples. We apply the proposed procedures in this chapter to a control and fault recovery problem of a simplified spacecraft propulsion system in Section 3.5. The results of the chapter are summarized in Section 3.6.

## 3.1 Problem Formulation

In this section we revisit the Robust Nonblocking Supervisory Control Problem (RNSCP) (Problem 2.3). In this problem full event observation is assumed. We intend to develop a computational procedure to find the optimal (maximally permissive) solution of RNSCP. We also establish the finite convergence of this procedure in case of regular languages for design specifications. Define the operator $\Omega' : Pwr(\Sigma^*) \to Pwr(\Sigma^*)$ as

$$\Omega'(K) = SupNb_{\boldsymbol{G}_n}(...SupNb_{\boldsymbol{G}_1}(SupC(SupR_{\boldsymbol{G}}(K))))$$

and consider the iterative procedure

$$\begin{aligned} E^{(0)} &= E \\ E^{(k+1)} &= \Omega'(E^{(k)}) \end{aligned} \tag{3.1}$$

**Theorem 3.1.** *Procedure (3.1) computes $E^*$ if it converges after a finite number of steps.*

*Proof.* Operator $\Omega'$ is monotone ($K_1 \subseteq K_2$ implies $\Omega'(K_1) \subseteq \Omega'(K_2)$) and contractive ($\Omega'(K) \subseteq K$). Therefore by Theorem 2.7, if procedure (3.1) terminates after $k^*$ steps, then $E^{(k)} = E^*$ for $k \geq k^*$. □

In [32] it is proved that the supremal controllable sublanguage of an $L_m(\boldsymbol{G})$-closed language is $L_m(\boldsymbol{G})$-closed. We prove that the supremal $\boldsymbol{G}_i$-Nonblocking sublanguage of a given $L_m(\boldsymbol{G})$-closed language will remain $L_m(\boldsymbol{G})$-closed too.

**Proposition 3.1.** *If $E \subseteq L_m(\boldsymbol{G}) \subseteq \Sigma^*$ is $L_m(\boldsymbol{G})$-closed (i.e. $E = \overline{E} \cap L_m(\boldsymbol{G})$), then the supremal $\boldsymbol{G}_i$-nonblocking sublanguage of $E$ is $L_m(\boldsymbol{G})$-closed.*

*Proof.* We need to show $\overline{SupNb_{\boldsymbol{G}_i}(E)} \cap L_m(\boldsymbol{G}) = SupNb_{\boldsymbol{G}_i}(E)$. According to (2.1), with $R = \overline{E} \cap L(\boldsymbol{G}_i) - \overline{E \cap L_m(\boldsymbol{G}_i)}$, $SupNb_{\boldsymbol{G}_i}(E) = E - R\Sigma^*$. Now observe that $\overline{SupNb_{\boldsymbol{G}_i}(E)} = \overline{E - R\Sigma^*} \subseteq \overline{\overline{E} - R\Sigma^*} = \overline{E} - R\Sigma^*$ (The last equality follows from

Lemma 2.1). Therefore,

$$\begin{aligned} SupNb_{\boldsymbol{G}_i}(E) &\subseteq \overline{SupNb_{\boldsymbol{G}_i}(E)} \cap L_m(\boldsymbol{G}) \\ &\subseteq (\overline{E} - R\Sigma^*) \cap L_m(\boldsymbol{G}) \\ &= (\overline{E} \cap L_m(\boldsymbol{G})) - R\Sigma^* \\ &= E - R\Sigma^* \quad \text{(Since } E \text{ is } L_m(\boldsymbol{G})\text{-closed)} \\ &= SupNb_{\boldsymbol{G}_i}(E) \end{aligned}$$

This shows that $\overline{SupNb_{\boldsymbol{G}_i}(E)} \cap L_m(\boldsymbol{G}) = SupNb_{\boldsymbol{G}_i}(E)$. □

Since $SupC$ and $SupNb_{G_i}$ preserve the $L_m(G)$-closure property, we can simplify procedure (3.1) by computing the supremal $L_m(\boldsymbol{G})$-closed sublanguage first before starting the iterative procedure with the operator $\Omega : Pwr(\Sigma^*) \to Pwr(\Sigma^*)$ with $\Omega(K) = SupNb_{\boldsymbol{G}_n}(...SupNb_{\boldsymbol{G}_1}(SupC(K)))$. Therefore:

$$\begin{aligned} E^{(0)} &= SupR_{\boldsymbol{G}}(E) \\ E^{(k+1)} &= \Omega(E^{(k)}) \end{aligned} \tag{3.2}$$

**Theorem 3.2.** *Procedure (3.2) computes $E^*$ if it converges after a finite number of steps.*

The proof is similar to the proof of Theorem 3.1 and is omitted for brevity.
This fixed point can be computed recursively by applying the supremal controllable and supremal $\boldsymbol{G}_i$-nonblocking operators. In Section 3.3 we show that considering $n$ finite-state automata $\boldsymbol{G}_i$ $(i = 1, ..., n)$, and assuming the design specifications $K_i$ are regular languages, the iterative procedure (3.2) converges in a finite number of steps. We will also present a computational procedure for implementing procedure (3.2).

## 3.2 Basic Computational Algorithms

In this section we develop some computational algorithms which can be regarded as building blocks of the main result for obtaining the maximally permissive solution of RNSCP. We start by presenting a lemma which helps us in this chapter.

**Lemma 3.1.** *Consider a deterministic automaton* $\boldsymbol{G} = (X, \Sigma, \eta, x_0, X_m)$. *Let* $X' \subseteq X$ *be a subset of states and* $L_{X'}(\boldsymbol{G}) = \{s \in L(\boldsymbol{G}) \mid \eta(x_0, s) \in X'\}$ *the set of strings leading to states in* $X'$. *If* $\boldsymbol{G}^*$ *is the finite-state automaton remaining from* $\boldsymbol{G}$ *after removing the states in* $X'$, *then* $L(\boldsymbol{G}^*) = L(\boldsymbol{G}) - L_{X'}(\boldsymbol{G})\Sigma^*$ *and* $L_m(\boldsymbol{G}^*) = L_m(\boldsymbol{G}) - L_{X'}(\boldsymbol{G})\Sigma^*$.

*Proof.* Removing states in $X'$ results in the removal of trajectories that include a state (or states) from $X'$. These trajectories correspond exactly to sequences $L_{X'}(\boldsymbol{G})\Sigma^*$ (since $\boldsymbol{G}$ is deterministic by assumption). □

### 3.2.1 Automata for union model and legal behavior

The following algorithm obtains a "union" automaton $\boldsymbol{G}$ that generates $L(\boldsymbol{G}) = \bigcup_{i=1}^{n} L(\boldsymbol{G}_i)$ and marks $L_m(\boldsymbol{G}) = \bigcup_{i=1}^{n} L_m(\boldsymbol{G}_i)$ (Theorem 2.4).

**Algorithm 3.1.** ***Algorithm for constructing automaton G***

$\boldsymbol{G} = Gu(\boldsymbol{G}_1, ..., \boldsymbol{G}_n)$

1. *Change automata* $\boldsymbol{G}_1$, $\boldsymbol{G}_2$,..., *and* $\boldsymbol{G}_n$ *to complete automata (with respect to* $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$*) as discussed in Sec. 2.1.4 to obtain* $\boldsymbol{G}'_1$, $\boldsymbol{G}'_2$,..., *and* $\boldsymbol{G}'_n$.

2. *Define a new automaton* $\boldsymbol{G} := \boldsymbol{G}'_1 \times \boldsymbol{G}'_2 \times ... \times \boldsymbol{G}'_n$. *Each state in* $\boldsymbol{G}$ *is an* $n$*-tuple* $x = (x_1, ..., x_n)$ *where* $x_i$ *is the corresponding state of* $\boldsymbol{G}'_i$.

3. *Remove the dump state* $(d_1, d_2, ..., d_n)$ *(and all of the transitions leading to this state) where* $d_i$ *is the dump state in* $\boldsymbol{G}'_i$.

4. *For every state* $x = (x_1, ..., x_n)$, *mark* $x$ *if and only if* $x_1 \in X_{m,1}$ *or* $x_2 \in X_{m,2}$ *or ... or* $x_n \in X_{m,n}$.

Automata $\boldsymbol{G}'_1$,..., $\boldsymbol{G}'_n$ generate $\Sigma^*$ and so does $\boldsymbol{G}$. Step 3 removes sequences that are not in $\bigcup_{i=1}^{n} L(\boldsymbol{G}_i)$. Thus $L(\boldsymbol{G}) = \bigcup_{i=1}^{n} L(\boldsymbol{G}_i)$. Step 4 marks the sequences that belong to $L_m(\boldsymbol{G}_1)$ or $L_m(\boldsymbol{G}_2)$ or, ..., or $L_m(\boldsymbol{G}_n)$, i.e. $L_m(\boldsymbol{G}) = \bigcup_{i=1}^{n} L_m(\boldsymbol{G}_i)$.

The following algorithm builds a trim automaton which marks the overall legal behavior $E$ in (2.2). In addition to automata $\boldsymbol{G}_1, ..., \boldsymbol{G}_n$, the inputs to the algorithm include trim automata $\boldsymbol{K}_1$, $\boldsymbol{K}_2$,..., and $\boldsymbol{K}_n$ that mark specification languages $K_1$, $K_2$,..., and $K_n$.

**Algorithm 3.2.** ***Algorithm for constructing automaton marking the legal behavior*** $E$

$\boldsymbol{E} = legal(\boldsymbol{G}_1, ..., \boldsymbol{G}_n, \boldsymbol{K}_1, ..., \boldsymbol{K}_n)$

1. *Complete automata* $\boldsymbol{G}_1$,..., $\boldsymbol{G}_n$ *and* $\boldsymbol{K}_1$,..., $\boldsymbol{K}_n$ *(with respect to* $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$*) to obtain* $\boldsymbol{G}'_1$,..., $\boldsymbol{G}'_n$ *and* $\boldsymbol{K}'_1$,..., $\boldsymbol{K}'_n$.

2. *Define a new automaton* $\boldsymbol{H}_0 := \boldsymbol{G}'_1 \times \boldsymbol{G}'_2 \times ... \times \boldsymbol{G}'_n \times \boldsymbol{K}'_1 \times \boldsymbol{K}'_2 \times ... \times \boldsymbol{K}'_n$. *It is immediate from the definition of automaton* $\boldsymbol{H}_0$ *that each state in* $\boldsymbol{H}_0$ *is a* $2n$*-tuple* $x = (x_1, ..., x_n, x_{n+1}, ..., x_{2n})$ *in which* $x_1, ..., x_n$ *are the states of* $\boldsymbol{G}'_1$,..., $\boldsymbol{G}'_n$ *and* $x_{n+1}, ..., x_{2n}$ *are the states of* $\boldsymbol{K}'_1$,..., $\boldsymbol{K}'_n$. *Mark all states of* $\boldsymbol{H}_0$.

3. *For* $i = 1, 2, ..., n$

   *For every state* $x = (x_1, ..., x_n, x_{n+1}, ..., x_{2n}) \in X$,

   *Unmark* $x$ *if* $x_i \in X_{m,i}$ *and* $x_{n+i} \notin X_m(K_i)$

   *end*

   *end*

4. *Unmark any state* $x = (x_1, ..., x_n, x_{n+1}, ..., x_{2n})$ *in which* $x_i \notin X_{m,i}$ *for all* $1 \leqslant i \leqslant n$.

5. *Trim* $\boldsymbol{H}_0$.

6. *Rename every state* $x = (x_1, ..., x_n, x_{n+1}, ..., x_{2n})$ *of* $\boldsymbol{H}_0$ *as* $x = (x_1, ..., x_n, y)$ *where* $y = (x_{n+1}, ..., x_{2n})$ *and call the resulting automaton* $\boldsymbol{E}$.

**Theorem 3.3.** *Upon exiting Algorithm 3.2, the trim automaton* $\boldsymbol{E}$ *marks* $E$ *in (2.2).*

*Proof.* After step 2, $\boldsymbol{H}_0$ will generate and mark $\Sigma^*$ since $L(\boldsymbol{H}_0) = [\bigcap_{i=1}^{n} L(\boldsymbol{G}_i')] \cap [\bigcap_{i=1}^{n} L(\boldsymbol{K}_i')] = \Sigma^*$ and $L_m(\boldsymbol{H}_0) = \Sigma^*$. In step 3, after the first iteration ($i = 1$), $\boldsymbol{H}_0$ will mark $\Sigma^* - (K_1^{co} \cap L_m(G_1)) = \Sigma^* - (K_1 \cup L_m^{co}(G_1))^{co}$. After iteration $i$, $L_m(\boldsymbol{H}_0) = \Sigma^* - \bigcup_{j=1}^{i}((K_j \cup L_m^{co}(\boldsymbol{G}_j)))^{co}$. In step 4, the states corresponding to the sequences in $L_m(\boldsymbol{G})^{co}$ will also be unmarked. Thus, $L_m(\boldsymbol{H}_0) = \Sigma^* - (\bigcup_{j=1}^{n}(K_j \cup L_m^{co}(\boldsymbol{G}_j))^{co} \cup L_m(\boldsymbol{G})^{co})$ which is exactly the same as (2.2). □

**Remark 3.1.** *Automaton $\boldsymbol{E}$ refines $\boldsymbol{G}$.*

**Remark 3.2.** *The computation of automaton $\boldsymbol{E}$ using Algorithm 3.2 has a time complexity of $O(nm^nk^n)$ where $m$ and $k$ are the number of states of $\boldsymbol{G}_i$ and $\boldsymbol{K}_i$ respectively and $n$ is the number of plants. This can be explained as follows. Adding a dump state in step 1 and completing the automaton is linear with respect to the number of states in each automaton. Constructing the product of these plants and their specifications has a time complexity of $O(m^nk^n)$. The number of states of $\boldsymbol{H}_0$ is $O(m^nk^n)$. Step 3 involves $n$ changes of marking for each state of $\boldsymbol{H}_0$. The final trim operation has a time complexity of $O(m^nk^n)$. Therefore, the total complexity of this algorithm is $O(nm^nk^n)$.*

### 3.2.2 $L_m(\boldsymbol{G})$-closed sublanguage

Now, we provide an algorithm to build the supremal $L_m(\boldsymbol{G})$-closed sublanguage of a given language $E \subseteq L_m(\boldsymbol{G})$ in the case that $E$ is regular and $\boldsymbol{G}$ is a finite-state automaton. This algorithm is different from the one in [10] and fits our overall procedure better.

Let $\boldsymbol{E}$ be a finite-state automaton such that $L(\boldsymbol{E}) = \overline{E}$ and $L_m(\boldsymbol{E}) = E$, and $\boldsymbol{E}$ refines $\boldsymbol{G}$. This algorithm takes $\boldsymbol{G}$ and $\boldsymbol{E}$ (obtained from Algorithms 3.1 and 3.2) as inputs and constructs a strict subautomaton of $\boldsymbol{E}$, denoted by $\boldsymbol{F}^*$, that marks the supremal $L_m(\boldsymbol{G})$-closed (closed relative to $L_m(\boldsymbol{G})$) sublanguage.

**Algorithm 3.3.** ***Algorithm for computing the supremal $L_m(\boldsymbol{G})$-closed sublanguage***

$\boldsymbol{F}^* = suprel(\boldsymbol{E}, \boldsymbol{G})$

1. *Remove any state* $x = (x_1, ..., x_n, y)$ *of* $\boldsymbol{E}$ *if* $(x_1, ..., x_n, y) \in X_m(\boldsymbol{G})$ *and* $x \notin X_m(\boldsymbol{E})$ *to obtain* $\boldsymbol{F}$.

2. *Trim* $\boldsymbol{F}$ *to obtain* $\boldsymbol{F}^*$.

**Theorem 3.4.** *Automaton* $\boldsymbol{F}^*$ *obtained in Algorithm 3.3 is a strict subautomaton of* $\boldsymbol{E}$ *and marks the supremal* $L_m(\boldsymbol{G})$*-closed sublanguage of* $L_m(\boldsymbol{E})$.

*Proof.* The strings leading to the states that are removed in step 1 are:

$$\begin{aligned}\{s \in \Sigma^* | s \in L(\boldsymbol{E}) \wedge s \in L_m(\boldsymbol{G}) \wedge s \notin L(\boldsymbol{E}) - L_m(\boldsymbol{E})\} &= \overline{E} \cap L_m(\boldsymbol{G}) \cap (L(\boldsymbol{E}) - L_m(\boldsymbol{E})) \\ &= \overline{E} \cap L_m(\boldsymbol{G}) \cap (\overline{E} - E) \\ &= \overline{E} \cap (L_m(\boldsymbol{G}) - E)\end{aligned} \tag{3.3}$$

Therefore by Lemma 3.1,

$$\begin{aligned}L_m(\boldsymbol{F}) &= L_m(\boldsymbol{E}) - (\overline{E} \cap (L_m(\boldsymbol{G}) - E))\Sigma^* \\ &= E - (\overline{E} \cap (L_m(\boldsymbol{G}) - E))\Sigma^*\end{aligned} \tag{3.4}$$

Next rewrite $L_m(\boldsymbol{G}) - E$ as $L_m(\boldsymbol{G}) - E = [(L_m(\boldsymbol{G}) - E) \cap \overline{E}] \cup [(L_m(\boldsymbol{G}) - E) - \overline{E}] = [(L_m(\boldsymbol{G}) - E) \cap \overline{E}] \cup [L_m(\boldsymbol{G}) - \overline{E}]$. Therefore,

$$\begin{aligned}E - (L_m(\boldsymbol{G}) - E)\Sigma^* &= E - ([(L_m(\boldsymbol{G}) - E) \cap \overline{E}] \cup [L_m(\boldsymbol{G}) - \overline{E}])\Sigma^* \\ &= E - ([(L_m(\boldsymbol{G}) - E) \cap \overline{E}]\Sigma^* \cup [L_m(\boldsymbol{G}) - \overline{E}]\Sigma^*) \\ &= (E - [(L_m(\boldsymbol{G}) - E) \cap \overline{E}]\Sigma^*) \cap (E - [L_m(\boldsymbol{G}) - \overline{E}]\Sigma^*) \\ &\qquad \text{(Since } A - (B \cup C) = (A - B) \cap (A - C))\end{aligned}$$

Since $E \cap [(L_m(\boldsymbol{G}) - \overline{E})\Sigma^*] = \emptyset$, $E - (L_m(\boldsymbol{G}) - \overline{E})\Sigma^* = E$, and therefore

$$E - (L_m(\boldsymbol{G}) - E)\Sigma^* = E - [(L_m(\boldsymbol{G}) - E) \cap \overline{E}]\Sigma^*$$

Substituting in (3.4), we obtain

$$\begin{aligned}L_m(\boldsymbol{F}) &= E - (L_m(\boldsymbol{G}) - E)\Sigma^* \\ &= SupR_{\boldsymbol{G}}(E)\end{aligned}$$

Thus, after step 2, automaton $\boldsymbol{F}^*$ will mark the supremal $L_m(\boldsymbol{G})$-closed sublanguage of $E$. □

**Remark 3.3.** *Algorithm 3.3 has a time complexity of $O(|X_{\boldsymbol{E}}|)$ where $|X_{\boldsymbol{E}}|$ is the number of states in the finite-state automaton $\boldsymbol{E}$.*

**Remark 3.4.** *Algorithm 3.3 can be used even if $\boldsymbol{E}$ and $\boldsymbol{G}$ are not obtained from Algorithms 3.1 and 3.2. In such case, if $\boldsymbol{E}$ does not refine $\boldsymbol{G}$, this automaton should be substituted with the product of $\boldsymbol{G}$ and $\boldsymbol{E}$, i.e. $\boldsymbol{E} := \boldsymbol{G} \times \boldsymbol{E}$. Then the time complexity of computing the supremal $L_m(\boldsymbol{G})$-closed sublanguage using Algorithm 3.3 will be $O(|X_{\boldsymbol{G}}|.|X_{\boldsymbol{E}}|)$ where $|X_{\boldsymbol{G}}|$ and $|X_{\boldsymbol{E}}|$ are the sizes of the state sets of $\boldsymbol{G}$ and $\boldsymbol{E}$.*

### 3.2.3 $\boldsymbol{G}_i$-nonblocking sublanguage

The notion of $\boldsymbol{G}_i$-nonblocking sublanguage has been introduced in Chapter 2. Next, we will present an algorithm which takes the automaton $\boldsymbol{E}$ marking the legal behavior, and the union plant $\boldsymbol{G}$ as constructed in Algorithms 3.1 and 3.2, to build a trim automaton $\boldsymbol{E}_{ni}$ that marks the supremal $\boldsymbol{G}_i$-nonblocking sublanguage of $E$. In other words, $L_m(\boldsymbol{E}_{ni}) = SupNb_{G_i}(E)$.

Before studying the algorithm, recall that the states of $\boldsymbol{E}$ are $(n+1)$-tuples belonging to $(X_1 \cup \{d_1\}) \times (X_2 \cup \{d_2\}) \times ... \times (X_n \cup \{d_n\}) \times Y$.

**Algorithm 3.4.** ***Algorithm for obtaining the supremal $\boldsymbol{G}_i$-nonblocking sublanguage***

$\boldsymbol{E}_{ni} = supnblki(\boldsymbol{E}, \boldsymbol{G})$

1. *Let $\boldsymbol{H}_1$ be a copy of automaton $\boldsymbol{E}$ but with marked state set $X_m(\boldsymbol{H}_1) = \{x \,|\, x = (x_1, ..., x_n, y) \in X_m(\boldsymbol{E})$ and $x_i \in X_{m_i}\}$.*

2. *Identify all of the uncoreachable states of $\boldsymbol{H}_1$ whose i-th element, $x_i$, is not a dump state and remove them from $\boldsymbol{E}$ and call the automaton $\boldsymbol{E}_{ni}$.*

3. *Trim $\boldsymbol{E}_{ni}$.*

**Theorem 3.5.** *The trim automaton $\boldsymbol{E}_{ni}$ obtained from Algorithm 3.4 is a strict subautomaton of $\boldsymbol{E}$ and marks the supremal $\boldsymbol{G}_i$-nonblocking sublanguage of $E$.*

*Proof.* It is straight-forward that after step 1, automaton $\boldsymbol{H}_1$ marks

$$L_m(\boldsymbol{H}_1) = L_m(\boldsymbol{G}_i) \cap L_m(\boldsymbol{E}) = L_m(\boldsymbol{G}_i) \cap E.$$

The set of sequences leading to states $x = (x_1, ..., x_n, y)$ in $\boldsymbol{H}_1$ that are coreachable or $x_i = d_i$ is $\overline{L_m(\boldsymbol{H}_1)} \cup L(\boldsymbol{G}_i)^{co}$. Therefore the sequences leading to states in $\boldsymbol{H}_1$ that are identified in step 2 are

$$\begin{aligned} L(\boldsymbol{H}_1) - (L(\boldsymbol{G}_i)^{co} \cup \overline{L_m(\boldsymbol{H}_1)}) &= \overline{E} \cap (L(\boldsymbol{G}_i)^{co} \cup \overline{L_m(\boldsymbol{H}_1)})^{co} \\ &= (\overline{E} \cap L(\boldsymbol{G}_i)) - \overline{L_m(\boldsymbol{H}_1)} \\ &= (\overline{E} \cap L(\boldsymbol{G}_i)) - \overline{E \cap L_m(\boldsymbol{G}_i)} \end{aligned}$$

Using Lemma 3.1, after step 2,

$$\begin{aligned} L_m(\boldsymbol{E}_{ni}) &= L_m(\boldsymbol{E}) - (\overline{E} \cap L(\boldsymbol{G}_i) - \overline{E \cap L_m(\boldsymbol{G}_i)})\Sigma^* \\ &= E - (\overline{E} \cap L(\boldsymbol{G}_i) - \overline{E \cap L_m(\boldsymbol{G}_i)})\Sigma^* \end{aligned}$$

Therefore, we can conclude that after step 3 the resulting trim automaton $\boldsymbol{E}_{ni}$ marks the supremal $\boldsymbol{G}_i$-nonblocking sublanguage of $E$. □

**Remark 3.5.** *The computational complexity of the above algorithm is $O(m^n k^n)$ since $(m+1)^n(k+1)^n$ is the maximum number of states of $\boldsymbol{E}$.*

## 3.3 Computational Algorithm for calculating the solution of RNSCP

In this section we present the main result which is an algorithm for solving RNSCP. The inputs are the plant models $\boldsymbol{G}_1, ..., \boldsymbol{G}_n$, and automata $\boldsymbol{K}_1, ..., \boldsymbol{K}_n$ which mark the specification languages. The algorithm, which constructs a trim automaton $\boldsymbol{E}^*$ marking the supremal element $\boldsymbol{E}^* = SupRCNb(E, \mathcal{G})$, implements the iterative procedure (3.2).

**Algorithm 3.5.** ***Algorithm for obtaining the solution of RNSCP***

$\boldsymbol{E}^* = suprcn(\boldsymbol{G}_1, \boldsymbol{K}_1, ..., \boldsymbol{G}_n, \boldsymbol{K}_n)$

1. *Build* $\boldsymbol{G} := Gu(\boldsymbol{G}_1, ..., \boldsymbol{G}_n)$ *using Algorithm 3.1.*

2. *Build* $\boldsymbol{E} := legal(\boldsymbol{G}_1, ..., \boldsymbol{G}_n, \boldsymbol{K}_1, ..., \boldsymbol{K}_n)$ *using Algorithm 3.2.*

3. $\boldsymbol{R}_0^1 := suprel(\boldsymbol{E}, \boldsymbol{G})$

4. $j = 1$

   *While* $\boldsymbol{R}_0^j$ *is nonempty*

   $\boldsymbol{R}_1^j = supcon(\boldsymbol{R}_0^j, \boldsymbol{G})$

   *For* $i = 1, ..., n$

   $\boldsymbol{R}_{i+1}^j = supnblki(\boldsymbol{R}_i^j, \boldsymbol{G})$

   *End*

   *If* $\boldsymbol{R}_{n+1}^j \neq \boldsymbol{R}_0^j$

   $\boldsymbol{R}_0^{j+1} := \boldsymbol{R}_{n+1}^j$

   $j = j + 1$

   *Else*

   $\boldsymbol{E}^* = \boldsymbol{R}_{n+1}^j$,

   *stop*

   *End(If)*

   *End(While)*

   $\boldsymbol{E}^*$ = *empty automaton,*

   *stop.*

**Theorem 3.6.** *Algorithm 3.5 terminates in at most p steps where p is the number of states of* $\boldsymbol{E}$ *calculated in Algorithm 3.2. The resulting automaton* $\boldsymbol{E}^*$ *marks* $SupRCNb(E, \mathcal{G})$.

*Proof.* In step 3, $\boldsymbol{R}_0^1$ is a strict subautomaton of $\boldsymbol{E}$ and marks the supremal $L_m(\boldsymbol{G})$-closed sublanguage of $E$. This corresponds to the first step in procedure (3.2).

Next $\boldsymbol{R}_1^1 \sqsubset \boldsymbol{R}_0^1 \sqsubset \boldsymbol{E}$ is obtained that marks $SupC(L_m(\boldsymbol{R}_0^1))$. Then the sequence $\boldsymbol{R}_{n+1}^1 \sqsubset \boldsymbol{R}_n^1 \sqsubset ... \sqsubset \boldsymbol{R}_1^1$ is obtained in which $\boldsymbol{R}_{i+1}^1$ marks the supremal $\boldsymbol{G}_i$-nonblocking sublanguage of $L_m(\boldsymbol{R}_i^1)$. This completes the first iteration of procedure (3.2). The algorithm proceeds with the next iterations. Since in each step of each iteration, a strict subautomaton of an automaton of the previous step (and that of $\boldsymbol{E}$) is obtained, and all automata, including $\boldsymbol{E}$, are finite-state, then the algorithm terminates in at most $p$ steps (where $p$ is the number of states of $\boldsymbol{E}$). □

**Remark 3.6.** *Since $\boldsymbol{E}$ refines $\boldsymbol{G}$ and has $O(m^n k^n)$ states, the complexity of supcon algorithm in step 1 is $O(m^n k^n)$. Similarly the complexity of computing supnblki is $O(m^n k^n)$. Therefore, the computational complexity of each iteration of step 4 would be $O(nm^n k^n)$. This algorithm will iterate $O(m^n k^n)$ times since automaton $\boldsymbol{E}$ has at most $(m+1)^n(k+1)^n$ states. Thus, Algorithm 3.5 has a time complexity of $O(nm^{2n}k^{2n})$.*

## 3.4 Examples

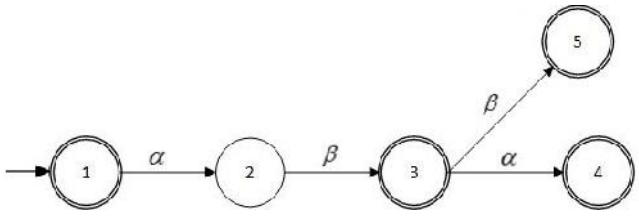
In this section we present two examples to illustrate the algorithms. In the first example, given two plant models and their specifications we find the overall legal behavior $E$ as well as the supremal $\boldsymbol{G}_1$-nonblocking sublanguage of $E$. In the second example, we seek the solution of the RNSCP with two plant models.

**Example 3.1.** Consider plants $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ and the automata representing their design specifications $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ in Fig. 3.1. In these representations, the doubly circled states denote the marked states of each automaton.

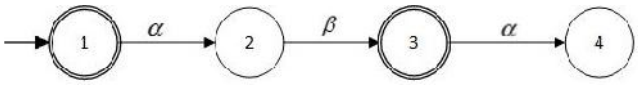
The overall legal marked behavior $\boldsymbol{E}$ for RNSCP can be built following the steps of Algorithm 3.2 as described below.

1. This step is straight-forward and its results are omitted for brevity.

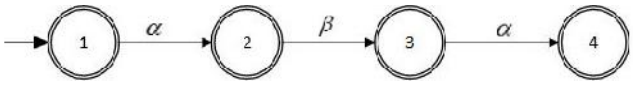
2. Automaton $\boldsymbol{H}_0$ from step 2 where all the states are marked is shown in Fig. 3.2(a).



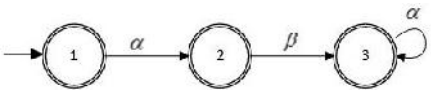

(a) $\boldsymbol{G}_1$




(b) $\boldsymbol{G}_2$

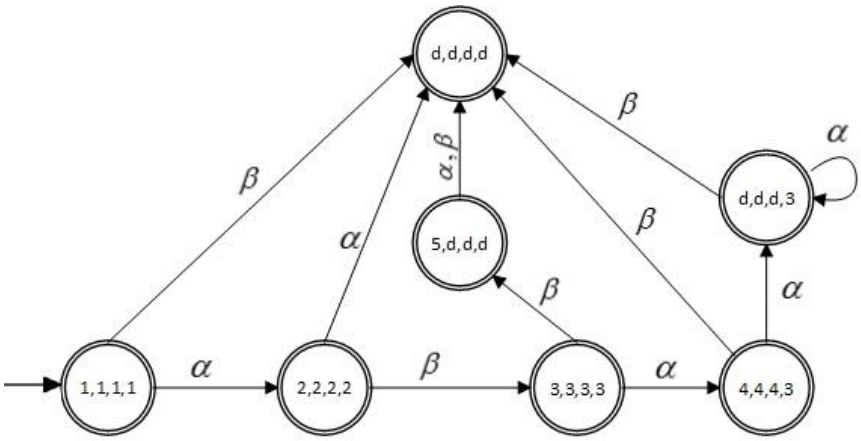



(c) $\boldsymbol{K}_1$

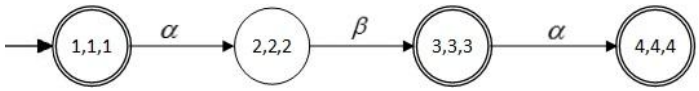



(d) $\boldsymbol{K}_2$

Figure 3.1: Example 3.1. Plants and specifications automata.




(a) $\boldsymbol{H}_0$ after step 2.




(b) $\boldsymbol{E}$

Figure 3.2: Example 3.1. Using Algorithm 3.2 to find $\boldsymbol{E}$.

3. In this step, state $(5, d, d, d)$ will be unmarked since $x_1 = 5 \in X_{m,1}$ and $x_3 = d \notin X_m(\boldsymbol{K}1)$.

4. State $(2, 2, 2, 2)$ becomes unmarked in this stage since $x_1 = 2 \notin X_{m,1}$ and $x_2 = 2 \notin X_{m,2}$. Using the same argument, states $(d, d, d, 3)$ and $(d, d, d, d)$ will be unmarked too. The sequences leading to these states do not belong to the marked behavior of the union plant $\boldsymbol{G}$.

5. Finally $\boldsymbol{H}_0$ is trimmed and after renaming states, automaton $\boldsymbol{E}$ is obtained (Fig. 3.2(b)).

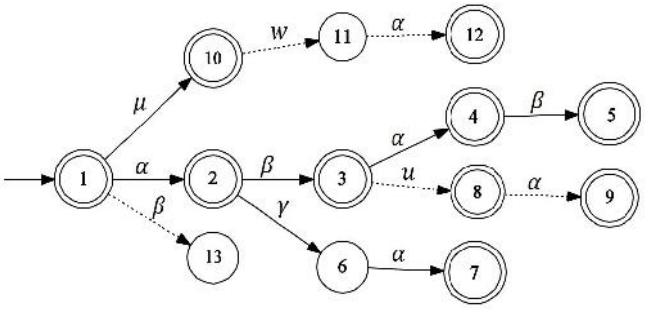
**Example 3.2.** In this example the problem of robust nonblocking supervisory control is being discussed. Consider plants $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ and their specifications $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ in Fig. 3.3. The illegal transitions in $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ are shown with dotted lines. Let the set of uncontrollable events be $\Sigma_{uc} = \{u, v, w\}$. Algorithm 3.5 is used to solve the problem.

1. Automaton $\boldsymbol{G}$ constructed using Algorithm 3.1 is shown in Fig. 3.4.

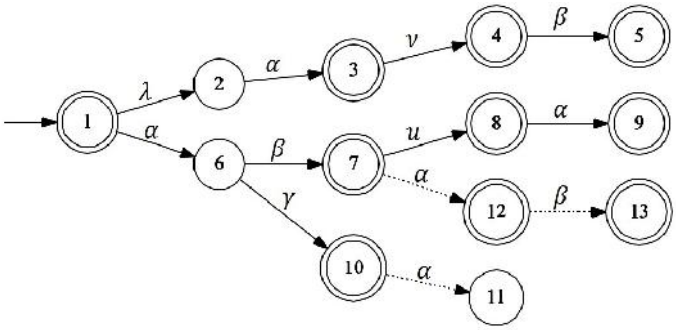
2. The overall legal marked behavior can be obtained using Algorithm 3.2 (Fig. 3.5).

3. The legal behavior $E$ is $L_m(\boldsymbol{G})$-closed, and $suprel(\boldsymbol{E}, \boldsymbol{G})$ returns $\boldsymbol{E}$.

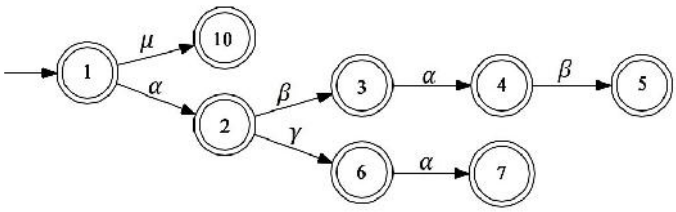
4. Now we can go through the iterative procedure in step 4 of Algorithm 3.5. The first step in iteration $j = 1$ is to find an automaton marking the supremal controllable sublanguage. It is easy to see that states $(3, 7, 9)$ and $(10, d, 10)$ violate controllability and are removed from automaton $\boldsymbol{R}_0^1$ which results in $\boldsymbol{R}_1^1$ in Fig. 3.6(a). Since $\boldsymbol{R}_1^1$ is $\boldsymbol{G}_1$-nonblocking, $\boldsymbol{R}_2^1$ remains the same as $\boldsymbol{R}_1^1$ with no state or transition elimination.



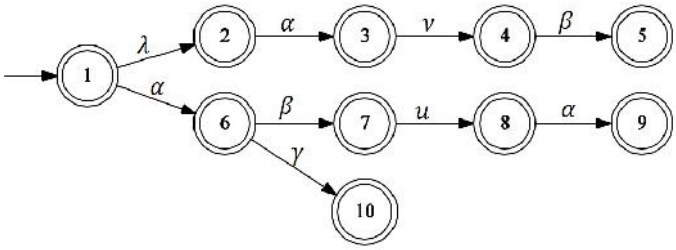

(a) $\boldsymbol{G}_1$




(b) $\boldsymbol{G}_2$




(c) $\boldsymbol{K}_1$




(d) $\boldsymbol{K}_2$

Figure 3.3: Example 3.2. Plants and specifications automata.

Next observe that the sequence $\alpha\gamma\alpha$ violates $\boldsymbol{G}_2$-nonblocking property. Following Algorithm 3.4 to obtain the supremal $\boldsymbol{G}_2$-nonblocking sublanguage of $L_m(\boldsymbol{R}_2^1)$, in step 1, states $(2,6,6)$ and $(7,11,8)$ are unmarked and since $(7,11,8)$ is not coreachable and $x_2 = 11$ is not a dump state, $(7,11,8)$ will be removed from $\boldsymbol{R}_2^1$ to obtain $\boldsymbol{R}_3^1$.

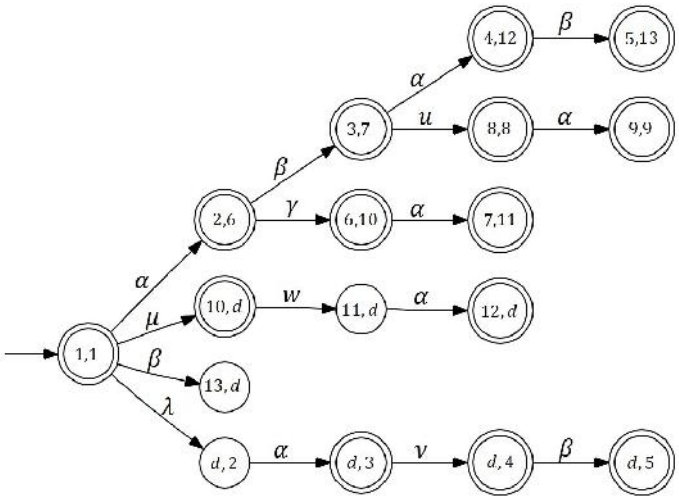



Figure 3.4: Example 3.2. Automaton $\boldsymbol{G}$.

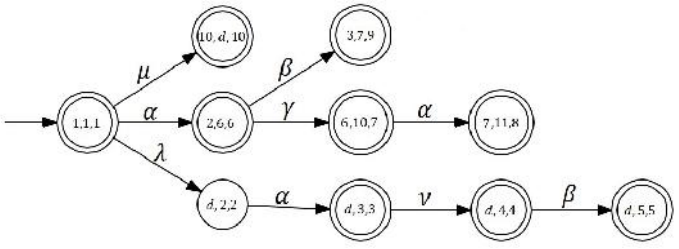
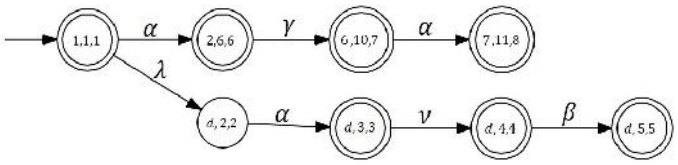
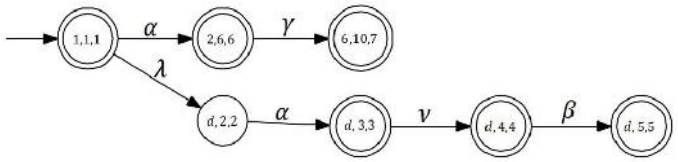



Figure 3.5: Example 3.2. Automaton $\boldsymbol{E}$ and $suprel(\boldsymbol{E}, \boldsymbol{G})$.

Since $\boldsymbol{R}_3^1$ is not the same as $\boldsymbol{R}_0^1$, we should proceed to the second iteration. Thus, $\boldsymbol{R}_0^2 := \boldsymbol{R}_3^1$ (Fig. 3.7(a)). $\boldsymbol{R}_0^2$ satisfies the controllability condition and therefore $\boldsymbol{R}_1^2 = \boldsymbol{R}_0^2$. State $(6,10,7)$ is removed in this step since $\alpha\gamma$ violates $\boldsymbol{G}_1$-nonblocking. The result is $\boldsymbol{R}_2^2$. In the next step, $\boldsymbol{R}_3^2$ is obtained by removing state $(2,6,6)$ to achieve $\boldsymbol{G}_2$-nonblocking property.

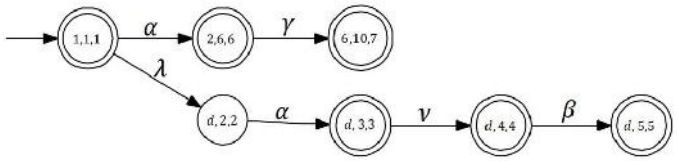



(a) $\boldsymbol{R}_1^1 = \boldsymbol{R}_2^1$

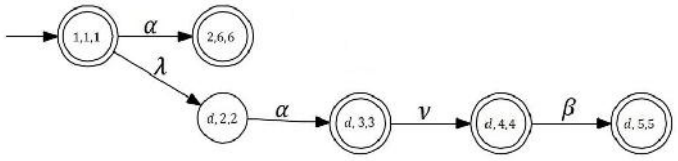



(b) $\boldsymbol{R}_3^1$

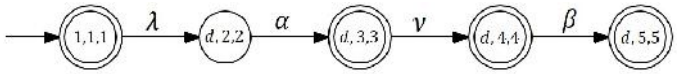
Figure 3.6: Example 3.2. Automata in iteration $j = 1$.




(a) $\boldsymbol{R}_0^2 = \boldsymbol{R}_1^2$




(b) $\boldsymbol{R}_2^2$




(c) $\boldsymbol{R}_3^2 = \boldsymbol{R}_0^3 = \boldsymbol{R}_3^3$

Figure 3.7: Example 3.2. Automata in iteration $j = 2$.

As $\boldsymbol{R}_3^2$ is not the same as $\boldsymbol{R}_0^2$, we should go through the third iteration by substituting $\boldsymbol{R}_0^3$ with $\boldsymbol{R}_3^2$. It can be easily checked that this automaton is controllable, $\boldsymbol{G}_1$-nonblocking and $\boldsymbol{G}_2$-nonblocking. Thus, no new state and

transition will be removed in this iteration and $\boldsymbol{R}_3^3 = \boldsymbol{R}_2^3 = \boldsymbol{R}_1^3 = \boldsymbol{R}_0^3$. Hence, automaton $\boldsymbol{E}^* = \boldsymbol{R}_3^3$ marks the supremal $L_m(\boldsymbol{G})$-closed, controllable and $\boldsymbol{G}_1$-nonblocking and $\boldsymbol{G}_2$-nonblocking sublanguage of the legal behavior $\boldsymbol{E}$.

## 3.5 Application Example: Spacecraft Propulsion System

In this section we study supervisory control of a simplified version of the Propulsion Module Subsystem (PMS) of the Cassini spacecraft [23, 28]. Fig. A.1 shows the full schematic of Cassini propulsion module subsystem. Fig. 3.8 shows the simplified propulsion system which consists of two propellant tanks and two engines E1 and E2. The valve assembly for E1 includes valves $V_1$, $V_2$, normally-open pyro valves $PV_1$ and $PV_2$, and normally-closed pyro valves $PV_3$ and $PV_4$. Two pressure sensors measure pressures $P_1$ and $P_2$, and a temperature sensor $T1$ monitors chemical reactions and thrust generation in E1.

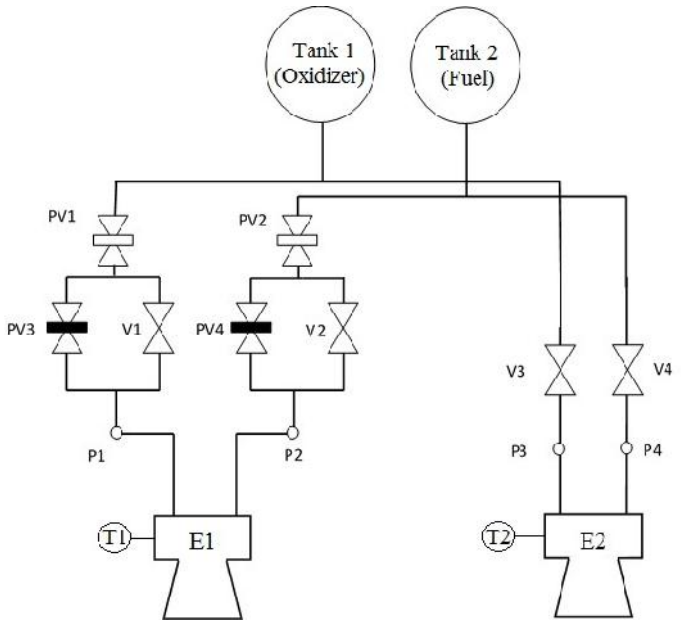



Figure 3.8: Spacecraft propulsion module subsystem.

The valve assembly for E2 is simpler consisting of valves $V_3$, $V_4$ and pressure sensors $P_3$, $P_4$ and temperature sensor $T2$. When the fuel paths between the propellant tanks and engines are open, propellants can combine, ignite and produce thrust. In this example, we intend to design a supervisor that in response to a high-level Master Controller fires the engine in such a way that the design specifications are satisfied. In our problem, for simplicity, only valve $V_1$ is assumed prone to "stuck-open" failure and valves $V_2$, $V_3$ and $V_4$ are assumed fault-free. Fig. 3.9 and 3.10 show the DES models of valve $V_1$ and valves $V_2$, $V_3$ and $V_4$ respectively. We assume that the valves are initially closed. The "stuck-open" failure mode of $V_1$ is permanent and the valve never returns to normal mode. Hence, any open or close command for valve $V_1$ after a failure has no effect.

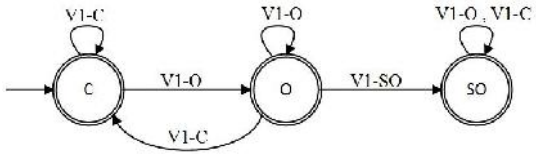



Figure 3.9: DES model of valve $V_1$.

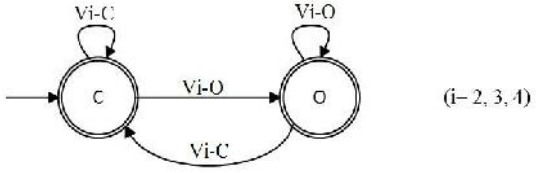
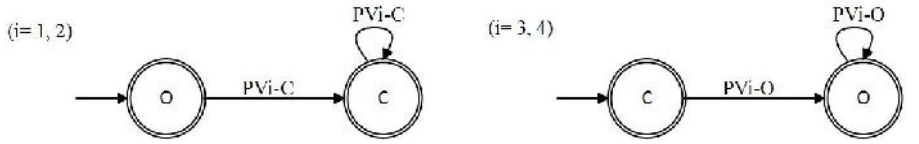



Figure 3.10: DES model of valves $V_2$,$V_3$ and $V_4$.

The models of the normally-open and normally-closed pyro valves are shown in Fig. 3.11.

The models of the four pressure sensors and two sensors measuring temperature as well as the Master Controller are shown in Fig. 3.12. Start and stop commands can be generated at any time by the Master Controller (Fig. 3.12(c)). Table 3.1 provides the list of events and their controllability status.



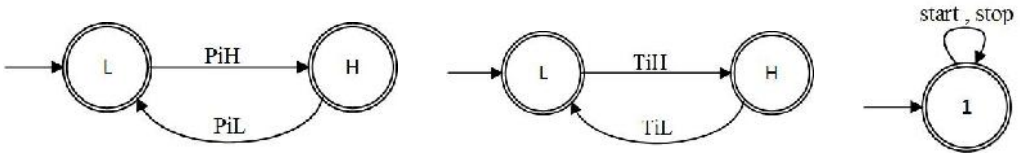

(a) Normally-open pyro valves $PV_1$ and $PV_2$.

(b) Normally-closed pyro valves $PV_3$ and $PV_4$.

Figure 3.11: DES model of pyro valves.

Table 3.1: List of Events and descriptions

| Event | Description | Controllable |
|---|---|---|
| $Vi - C$ | Valve $Vi$ closes (i=1,...,4) | Yes |
| $Vi - O$ | Valve $Vi$ opens (i=1,...,4) | Yes |
| $Vi - SO$ | Valve $Vi$ fails stuck open | No |
| $PVi - C$ | Pyro valve $Vi$ closes (i=1,2) | Yes |
| $PVi - O$ | Pyro valve $Vi$ opens (i=3,4) | Yes |
| $PiH$ | Pressure sensor $i$ becomes high (i=1,...,4) | No |
| $PiL$ | Pressure sensor $i$ becomes low (i=1,...,4) | No |
| $TiH$ | Thrust of engine $i$ becomes high (i=1,2) | No |
| $TiL$ | Thrust of engine $i$ becomes low (i=1,2) | No |
| *start* | Master Controller issues start command | No |
| *stop* | Master Controller issues stop command | No |




(a) Pressure sensors $P_i$ $(i = 1, 2, 3, 4)$.

(b) Temperature sensors $T_1$,$T_2$.

(c) Master Controller

Figure 3.12: Sensors and Master Controller models

To complete the model, we build DES models INT1 to INT4 to describe the effect of valve positions on the pressure sensors, and INT5 and INT6 to model the reading of the temperature (thrust) sensors as a function of the pressures measured by $P_1$, $P_2$, $P_3$ and $P_4$. The reading of pressure sensor $P_1$ depends on the state of $V_1$, $PV_1$ and $PV_3$. Specifically the pressure goes high only when $PV_1$ is open, and either $V_1$ is open (or stuck-open) or $PV_3$ is open. This dependency is represented by INT1 in Fig. 3.13. INT1 is obtained by adding selfloops of pressure sensor readings to the $sync(PV_1, PV_3, V_1)$ (The transitions of *sync* are not displayed to avoid cluttering the figure). The state names in Fig. 3.13 indicate the current state of $PV_1$, $PV_3$ and $V_1$ respectively. INT2, INT3 and INT4 are constructed similarly for pressure sensors $P_2$, $P_3$ and $P_4$.

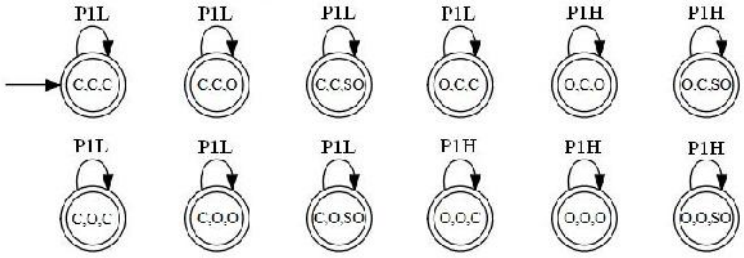



Figure 3.13: INT1: Interaction between $PV_1$, $PV_3$, $V_1$ and $P_1$.

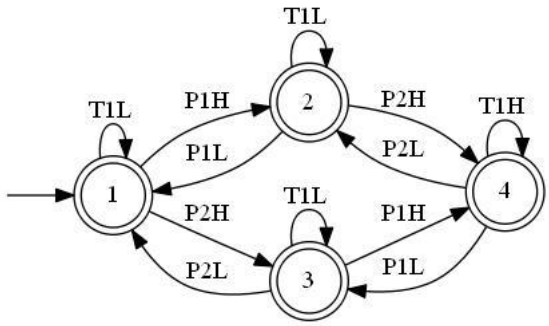
Now we proceed to model the interactions between pressure sensor readings and the thrust of engines. The thrust of an engine goes high only when the corresponding pressure sensors show high pressure reading. Fig. 3.14 shows these interactions. The plant model $\boldsymbol{G}$ is obtained by the synchronous product of all component and interaction models.

Broadly speaking, the design specifications require the use of E1 for normal mode and E2 in case of valve $V_1$ failure and prohibit simultaneous firing of E1 and E2. In normal (resp. faulty) mode, the supervisor must be able to turn on E1 (resp. E2) and turn off E1 (resp. E2). (Other requirements that deal with other issues such as fuel waste are not considered for brevity.) The detailed design specifications are as

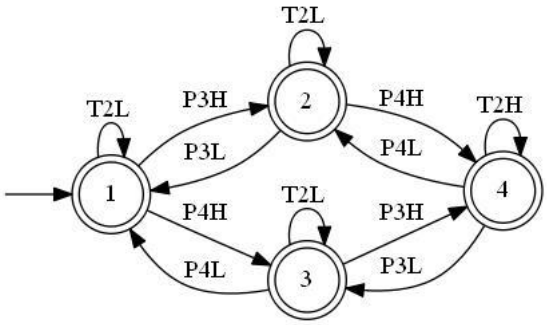
follows.

In normal mode:

- The system should wait for the start command to start firing engine E1.
- After the start command is issued by the master controller, any start and stop command should be ignored during the start-up procedure until thrust is generated.




(a) INT5




(b) INT6

Figure 3.14: Interactions between pressure sensors and temperature sensors.

- When the thrust is high, once the master controller issues the stop command, the shutdown procedure starts.
- During the shutdown procedure, any start and stop command must be ignored.

The automaton for start-up and shutdown in normal mode is shown in Fig. 3.15.

Engine 2 is a backup engine and must be used in case of engine 1 failure. When fault

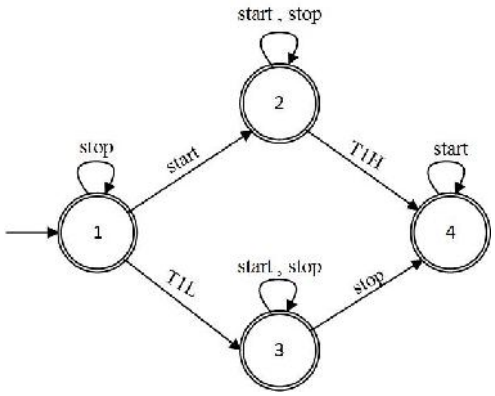



Figure 3.15: DES model of start up and shutdown procedure in normal mode.

(i.e. valve $V_1$ stuck-open) occurs, engine 1 shall no longer be used and the system must switch over to engine 2 based on the current state of the system. The design specifications for the faulty mode of the system are given below.

- If the fault occurs before a start command, the system should switch completely to engine 2 and wait for the start command. In other words, the procedure for starting and shutting down engine 2 should be followed.

- If the fault occurs after the master controller issues a start command, engine 1 should continue firing until the master controller issues the stop command and engine E1 is turned off. Then the system must switch to engine E2 for future maneuvers.

- If the fault occurs after the master controller issues a stop command but before engine E1 is turned off, the engine must be turned off and then the system should switch over to engine E2.

The DES model of these design specifications is shown in Fig. 3.16.

This supervisory control problem is a problem of fault recovery with a normal and a faulty mode. In each mode there are two sets of marked states, one for thrust low

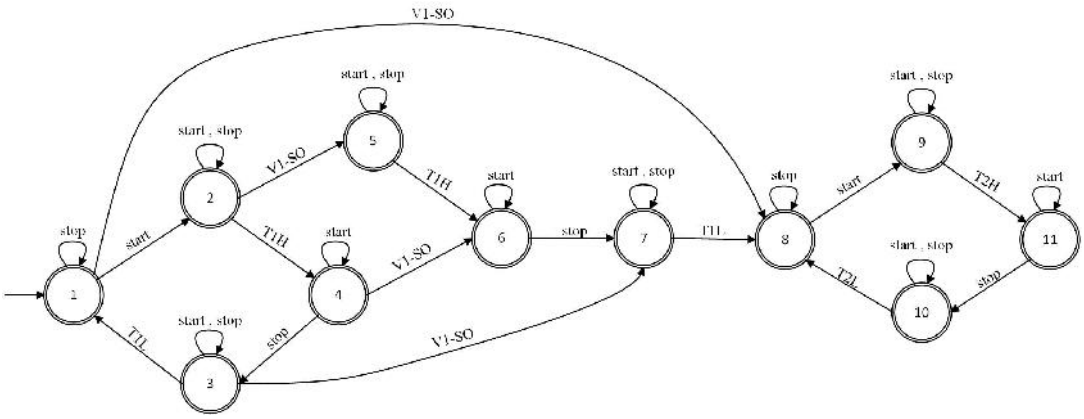



Figure 3.16: DES model of start-up and shutdown procedure in faulty mode.

(engine off) and another for thrust high (engine on). The problem can be solved as a robust control problem with four plant models $\boldsymbol{G}_{N,off}$, $\boldsymbol{G}_{N,on}$, $\boldsymbol{G}_{NF,off}$ and $\boldsymbol{G}_{NF,on}$. $\boldsymbol{G}_{N,off}$ (resp. $\boldsymbol{G}_{N,on}$) are the subautomaton of plant model $\boldsymbol{G}$ with states in normal mode $\boldsymbol{G}_N$ with thrust low (resp. high) marked. This marking can be done by the synchronous product of $\boldsymbol{G}_N$ with automaton Marker N in Fig. 3.17 with state 1 (rep. state 2) of Marker N marked. Similarly $\boldsymbol{G}_{NF,off}$ (resp. $\boldsymbol{G}_{NF,on}$) are obtained by the synchronous product of $\boldsymbol{G}$ with Marker F (Fig. 3.18) with state 3 (resp. state 4) marked.

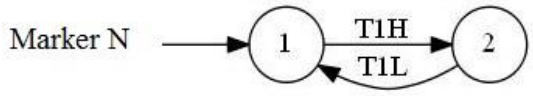



Figure 3.17: Mark state 1 for OFF and state 2 for ON.

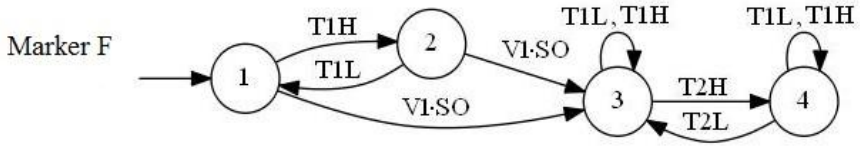



Figure 3.18: Mark state 3 for OFF and state 4 for ON.

All algorithms presented in this chapter have been implemented using DECK [46],

[47]. The resulting plant model $\boldsymbol{G}$ has 16384 states and 278528 transitions. The solution of the RNSCP has 944 states and 11438 transitions.

To illustrate what the robust supervisor does, consider a sample sequence from the system under supervision automaton depicted in Fig. 3.19. Master Controller issues a start command. After valves $V_1$ and $V_2$ become open, pressures $P_1$ and $P_2$ become high. At this stage the thrust of E1 becomes high and the first engine fires. Next valve $V_1$ becomes stuck-open (It is assumed the failure is diagnosed quickly and hence is treated an observable event.) Next when a stop command is issued, engine E1 is switched off. This is done (in the sample sequence shown in Fig. 3.19) by firing $PV_1$ to shut the path between the oxidizer tank and engine E1 and by closing valve $V_2$ which shuts off supply from the fuel tank. Following another start command from Master Controller, the system switches to engine E2 and the start-up procedure for E2 opens valves $V_3$ and $V_4$, the pressure sensors $P_3$ and $P_4$ show high pressure and E2 fires. Later, when a stop command comes from Master Controller, engine E2 is turned off by closing the corresponding valves $V_3$ and $V_4$. When Master Controller issues another start command to generate thrust, engine E2 is being used and the same sequence happens to complete a thrust on and thrust off sequence.

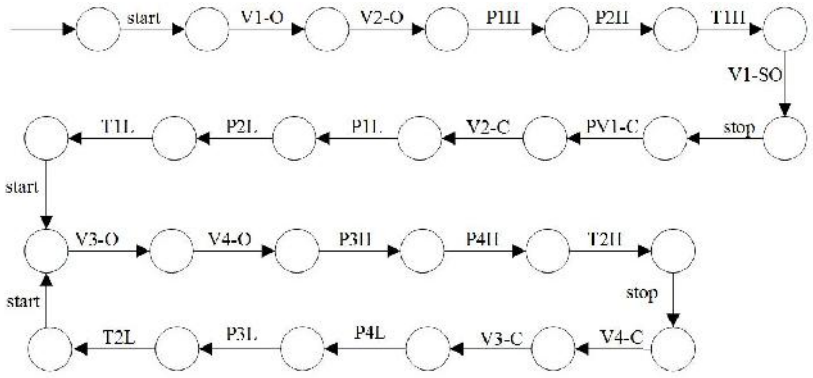



Figure 3.19: Sample start-up and shutdown sequences.

It should be noted that it is necessary to ensure nonblocking with respect to

multiple sets of marked states (and not just one set). Suppose we choose the E1 off states as the only marked state set for normal mode. Assume E1 is "on" in normal mode and Master Controller issues thrust shutoff command. The conventional (non robust) supervisory control solution allows the system to close $PV_1$ to turn off the thrust of engine 1. But the supervisor will no longer be able to turn on E1 since the pyro valves cannot be opened again. The robust supervisor designed based on RNSCP however does not fire $PV_1$ and $PV_2$ unless a fault occurs.

In the configuration shown in Fig. 3.8, pyro valves $PV_3$ and $PV_4$ are redundant. In fact, it can be validated that the RNSCP still has a solution event if $PV_3$ and $PV_4$ are removed from the propulsion system. This shows that the RNSCP and its maximally permissive solution can be used as a design tool to validate different valve configurations. Finally, it should be noted that the purpose of having $PV_3$ and $PV_4$ in Fig. 3.8 is to have the ability for handling stuck-closed failures of valves $V_1$ and $V_2$ (which have not been studied in this example).

## 3.6 Conclusions

This section presents a computational algorithm for the supremal controllable, $L_m(\boldsymbol{G})$-closed and $\boldsymbol{G}_i$-nonblocking sublanguage. This sublanguage provides the maximally permissive solution for the robust nonblocking supervisory control problem. The algorithm is used to solve a control and fault recovery problem in a propulsion system. In the next chapter, the algorithm will be extended to the case of partial event observation.

# Chapter 4

# Robust Nonblocking Supervisory Control Problem with Partial Observation

The notion of robust supervisory control of discrete event systems has been introduced in order to handle the situations in which the plant's models and dynamics are not known. In Chapter 3, we have examined RNSCP when all the events are assumed to be observable. This problem can be extended so that it can cover partial observation of the events. In this chapter we will consider the robust nonblocking supervisory control problem in case of partial observation (RNSCP-PO) and will present a computational algorithm for it.

This chapter is organized as follows. Section 4.1 describes the problem formulation. In Section 4.2, the class of supremal normal sublanguage are introduced. Section 4.2 proposes a computational algorithm for refining the automaton marking the legal behavior to have sufficient information on the states and transitions as well as projection mapping. This algorithm is used later in this section to form the initial automaton of the computational procedure for finding the supremal normal

sublanguage. In Section 4.3 the computational procedure for the RNSCP-PO is provided and its finite convergence is proved. Section 4.4 demonstrates the proposed algorithms by providing illustrative examples. This chapter is concluded in Section 4.5.

## 4.1 Problem Formulation

As mentioned earlier in Chapter 2, the class of observable sublanguages of $E$ need not have a supremal element in general. Hence, the RNSCP-PO may not have an optimal (maximally permissive) solution. However, if we substitute observability with normality (restricting the set of solutions with an stronger property), we can find the supremal element of the new set of solutions since the set of normal sublanguages has a supremal element.

Now recall the robust nonblocking supervisory control problem with partial observation discussed in Chapter 2 (Problem 2.5).

Define the operator $\Omega' : Pwr(\Sigma^*) \to Pwr(\Sigma^*)$ as

$$\Omega'(K) = SupNb_{\boldsymbol{G}_n}(...SupNb_{\boldsymbol{G}_1}(SupC(Sup\overline{N}(SupR_{\boldsymbol{G}}(K)))))$$

and consider the iterative procedure

$$\begin{aligned} E^{(0)} &= E \\ E^{(k+1)} &= \Omega'(E^{(k)}) \end{aligned} \tag{4.1}$$

**Theorem 4.1.** *Procedure (4.1) computes $E^*$ if it converges after a finite number of steps.*

*Proof.* Operator $\Omega'$ is monotone ($K_1 \subseteq K_2$ implies $\Omega'(K_1) \subseteq \Omega'(K_2)$) and contractive ($\Omega'(K) \subseteq K$). Therefore by Theorem 1 of [17], if procedure (4.1) terminates after $k^*$ steps, then $E^{(k)} = E^*$ for $k \geq k^*$. □

It has been shown in [32] and Chapter 2 that $SupC$ and $SupNb_{G_i}$ preserve the

$L_m(\boldsymbol{G})$-closure property. Since $Sup\overline{N}(E)$ also remains $L_m(\boldsymbol{G})$-closed if $E$ is $L_m(\boldsymbol{G})$-closed, then we can simplify procedure (4.1) by computing the supremal $L_m(\boldsymbol{G})$-closed sublanguage first before starting the iterative procedure and use the operator $\Omega : Pwr(\Sigma^*) \to Pwr(\Sigma^*)$ with

$$\Omega(K) = SupNb_{\boldsymbol{G}_n}(...SupNb_{\boldsymbol{G}_1}(SupC(Sup\overline{N}(K))))$$

Therefore:

$$\begin{aligned} E^{(0)} &= SupR_{\boldsymbol{G}}(E) \\ E^{(k+1)} &= \Omega(E^{(k)}) \end{aligned} \tag{4.2}$$

**Theorem 4.2.** *Procedure (4.2) computes $E^*$ if it converges after a finite number of steps.*

The proof is similar to the proof of Theorem 4.1 and is omitted for brevity.
This fixed point can be computed recursively by applying the supremal normal, supremal controllable and supremal $\boldsymbol{G}_i$-nonblocking operators. In the next section we show that considering $n$ finite-state automata $\boldsymbol{G}_i$ $(i = 1, ..., n)$, and assuming the design specifications $K_i$ are regular languages, the iterative procedure (4.2) converges in a finite number of steps. We will also present a computational procedure for implementing procedure (4.2).

In order to be able to find a computational procedure for obtaining the solution of RNSCP with partial observation formulated above, we introduce a procedure for calculating $Sup\overline{N}(E, L(\boldsymbol{G}))$.

## 4.2 Computation of Supremal Normal Sublanguage

A computational algorithm for calculating the supremal normal sublanguage is presented in [10]. In this work, we aim to come up with an initial automaton and make all the changes to this single automaton until we reach to the solution. The difference in our work and [10] is that the latter does some preprocessing on the input

automata to make their state name match each other. Then, in each iteration it uses two intermediate automata. We will present an alternative algorithm which uses a single automaton and appears to be simpler.

We start by presenting an algorithm for obtaining automaton $\boldsymbol{H}$ which has enough information to find the supremal normal sublanguage by removing transitions and states using the plant $\boldsymbol{G}$, legal behavior automaton $\boldsymbol{E}$ and the projection map $P$. In this approach, $\boldsymbol{G}$ and $\boldsymbol{E}$ are obtained using Algorithm 3.1 and 3.2 respectively.

Automaton $\boldsymbol{H}$ can be constructed by defining it as $\boldsymbol{H} = P^{-1}P(\boldsymbol{G})$ with $P(.)$ and $P^{-1}$ are defined in Chapter 2. It is clear that $L_m(\boldsymbol{H}) = E \cap P^{-1}P(\boldsymbol{G}) = E$ since $E \subseteq L_m(\boldsymbol{G}) \subseteq P^{-1}P(L_m(\boldsymbol{H}))$. Similarly $L(\boldsymbol{H}) = \overline{E}$.

This new automaton can be used as the initial automaton for going through the procedure of solving the Robust Nonblocking Supervisory Control problem with Partial Observation stated in (4.2).

**Remark 4.1.** *Automaton $\boldsymbol{H}$ refines $\boldsymbol{G}$.*

Recall that each state of automaton $\boldsymbol{G}$ is a $n$-tuple $x_{\boldsymbol{G}} = (x_1, ..., x_n)$ and the states of $\boldsymbol{E}$ are $(n+1)$-tuples $x_{\boldsymbol{E}} = (x_1, ..., x_n, y)$ according to Algorithms 3.1, 3.2. Now, we simplify the names of state of $\boldsymbol{G}$ for further use in this work. Let $x_{\boldsymbol{G}}$ denote the state names of $\boldsymbol{G}$ where $x_{\boldsymbol{G}} = (x_1, ..., x_n)$. With this definition, we can rename the states of $\boldsymbol{E}$ as pairs $x_{\boldsymbol{E}} = (x_{\boldsymbol{G}}, y)$.
With the aforementioned comment on state renaming, it is easy to see that the states of automaton $\boldsymbol{H}$ obtained from $\boldsymbol{H} = \boldsymbol{E} \times P^{-1}P(\boldsymbol{G})$ can be written as $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$ with $z_{\boldsymbol{G}} \in Pwr(x_{\boldsymbol{G}})$. Before introducing some useful properties of this new automaton, we need to provide new definitions.

**Definition 4.1.** *Two states of $\boldsymbol{H}$, $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$ and $(x'_{\boldsymbol{G}}, y', z'_{\boldsymbol{G}})$, are called a matching pair if $z_{\boldsymbol{G}} = z'_{\boldsymbol{G}}$ but $x_{\boldsymbol{G}} \neq x'_{\boldsymbol{G}}$.*

This definition resembles matching pairs in [20].

**Definition 4.2.** ***Pair Deficiency***

*A state* $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$ *is called pair-deficient if it has less than* $|z_{\boldsymbol{G}}|$ *matching pairs in automaton* $\boldsymbol{H}$*.*

It is clear from the above definition that if a state $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$ with $|z_{\boldsymbol{G}}| = 1$, cannot be pair-deficient.

**Lemma 4.1.** *If* $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$ *is pair deficient, then there exist* $x'_{\boldsymbol{G}} \in z_{\boldsymbol{G}}$ *and* $s \in L(\boldsymbol{G})$ *such that* $s$ *leads to* $x'_{\boldsymbol{G}}$ *(in* $\boldsymbol{G}$*) and* $P(s)$ *leads to* $z_{\boldsymbol{G}}$ *(in* $P(\boldsymbol{G})$*), but* $s \notin \overline{E}$*.*

**Lemma 4.2.** *Suppose* $(x_{\boldsymbol{G}_1}, y_1, z_{\boldsymbol{G}})$,...,$(x_{\boldsymbol{G}_i}, y_i, z_{\boldsymbol{G}})$ *be matching pairs. Then, there exists* $i$ *different strings* $s_1$,...,$s_i$ *leading to* $(x_{\boldsymbol{G}_1}, y_1, z_{\boldsymbol{G}})$,...,$(x_{\boldsymbol{G}_i}, y_i, z_{\boldsymbol{G}})$ *respectively such that* $P(s_1) = ... = P(s_i)$*.*

*Proof.* States $(x_{\boldsymbol{G}_1}, y_1, z_{\boldsymbol{G}})$,...,$(x_{\boldsymbol{G}_i}, y_i, z_{\boldsymbol{G}})$ are reachable states in automaton $\boldsymbol{H}$. Therefore, if string $s_1$ leads to state $(x_{\boldsymbol{G}_1}, y_1, z_{\boldsymbol{G}})$ in $\boldsymbol{H}$, then $s_1 \in L(\boldsymbol{E}) \cap L(\boldsymbol{G}_s)$. Thus, string $s_1$ leads to state $(x_{\boldsymbol{G}_1}, y_1)$ in $\boldsymbol{E}$ and from the structure of this automaton we can deduce that execution of $s_1$ takes us to the state $x_{\boldsymbol{G}_1} \in z_{\boldsymbol{G}}$ in automaton $\boldsymbol{G}$. Similarly $x_{\boldsymbol{G}_j} \in z_{\boldsymbol{G}}$ $(j = 2, ..., i)$. Now, since $s_1$ leads to $z_{\boldsymbol{G}}$ in $P^{-1}P(\boldsymbol{G})$ and $x_{\boldsymbol{G}_1}, ..., x_{\boldsymbol{G}_i} \in z_{\boldsymbol{G}}$, there exist strings $s_2$,...,$s_i$ such that $P(s_1) = ... = P(s_i)$. □

Next, we will present an algorithm which takes the automaton marking the legal behavior, $\boldsymbol{E}$, and the union plant $\boldsymbol{G}$ as constructed in Algorithms 3.2 and 3.1 ($legal(.)$ and $Gu(.)$ respectively), to build a trim automaton $\boldsymbol{E}_{norm}$ that marks the supremal normal sublanguage of $E$ with respect to $P$ and $L(\boldsymbol{G})$. In other words, $L_m(\boldsymbol{H}_n) = Sup\overline{N}(E, L(\boldsymbol{G}))$.

**Algorithm 4.1.** ***Algorithm for obtaining the supremal normal sublanguage*** $\boldsymbol{E}_{norm} = supnorm(\boldsymbol{E}, \boldsymbol{G})$

1. *Build automaton* $\boldsymbol{H} = \boldsymbol{E} \times P^{-1}P(\boldsymbol{G})$*.*

2. *Repeat the steps (a) and (b) until there is no state removal:*

(a) *Identify states* $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$ *where* $|z_{\boldsymbol{G}}| \geq 2$. *If the state is pair-deficient, remove it along with all of its matching pairs. Name the new automaton* $\boldsymbol{H}$.

(b) *Trim* $\boldsymbol{H}$.

*End(repeat)*

3. *Let* $\boldsymbol{E}_{norm}$ *be the final automaton.*

**Theorem 4.3.** *The trim automaton* $\boldsymbol{E}_{norm}$ *obtained from Algorithm 4.1 is a subautomaton of* $\boldsymbol{E}$ *and marks the supremal normal sublanguage of* $E$.

*Proof.* At the start of algorithm, $L_m(\boldsymbol{H}) = E$ and $L(\boldsymbol{H}) = \overline{E}$. In step 2, those states which are pair deficient will be removed. Suppose $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$ is a pair-deficient state. In this case, there exist strings $s_1$ and $s_2$ such that $s_1 \in \overline{E}$, $s_2 \in L(\boldsymbol{G}) - \overline{E}$ and $P(s_1) = P(s_2)$. It is clear that the existence of these strings contradicts the normality property. Therefore, by removing this pair deficient state and all of its remaining matching pairs, all strings $s_i \in P^{-1}(P(s_2))$ would be deleted from automaton $\boldsymbol{H}$. Thus, after step 2(a)

$$L(\boldsymbol{H}) = \overline{E} - P^{-1}P(L(\boldsymbol{G}) - \overline{E})\Sigma^*$$

$$\begin{aligned} L_m(\boldsymbol{H}) &= E \cap L(\boldsymbol{H}) \\ &= E \cap (\overline{E} - P^{-1}P(L(\boldsymbol{G}) - \overline{E})\Sigma^*) \end{aligned}$$

Next, in step 2(b) the uncoreachable states which correspond to unmarked states in $\boldsymbol{E}$ are removed. The above operations correspond to the first iteration of procedure (2.8). Finally after step 3 the resulting trim automaton $\boldsymbol{E}_{norm}$ marks the supremal normal sublanguage of $E$, $Sup\overline{N}(E, L(\boldsymbol{G}))$. □

## 4.3 Computational Procedure for RNSCP-PO

In this section, we present an algorithm for solving RNSCP-PO. The inputs are the plant models $\boldsymbol{G}_1, ..., \boldsymbol{G}_n$, and $\boldsymbol{K}_1, ..., \boldsymbol{K}_n$ which mark the specification languages. Algorithm 4.2, which constructs a trim automaton $\boldsymbol{E}^*$ that marks the supremal element $\boldsymbol{E}^* = Sup\overline{RN}CNb(E, \mathcal{G})$, implements the iterative procedure (4.2).

**Algorithm 4.2.** ***Algorithm for obtaining the solution of RNSCP-PO***

$\boldsymbol{E}^* = suprncn(\boldsymbol{G}_1, \boldsymbol{K}_1, ..., \boldsymbol{G}_n, \boldsymbol{K}_n)$

1. *Build* $\boldsymbol{G} := Gu(\boldsymbol{G}_1, ..., \boldsymbol{G}_n)$.
2. *Build* $\boldsymbol{E} := legal(\boldsymbol{G}_1, ..., \boldsymbol{G}_n, \boldsymbol{K}_1, ..., \boldsymbol{K}_n)$.
3. *Build* $\boldsymbol{H} = \boldsymbol{E} \times P^{-1}P(\boldsymbol{G})$.
4. $\boldsymbol{R}_0^1 := suprel(\boldsymbol{H}, \boldsymbol{G})$
5. $j = 1$

   *While* $\boldsymbol{R}_0^j$ *is nonempty*

   $\boldsymbol{R}_1^j = supnorm(\boldsymbol{R}_0^j, \boldsymbol{G})$

   $\boldsymbol{R}_2^j = supcon(\boldsymbol{R}_1^j, \boldsymbol{G})$

   *For* $i = 1, ..., n$

   $\boldsymbol{R}_{i+2}^j = supnblki(\boldsymbol{R}_{i+1}^j, \boldsymbol{G})$

   *End*

   *If* $\boldsymbol{R}_{n+2}^j \neq \boldsymbol{R}_0^j$

   $\boldsymbol{R}_0^{j+1} := \boldsymbol{R}_{n+2}^j$

   $j = j + 1$

   *Else*

   $\boldsymbol{E}^* = \boldsymbol{R}_{n+2}^j$,

   *stop*

   *End(If)*

   *End(While)*

$\boldsymbol{E}^* = \textit{empty automaton,}$

*stop.*

**Theorem 4.4.** *Algorithm 4.2 terminates in at most $p$ steps where $p$ is the number of states of $\boldsymbol{H}$. The resulting automaton $\boldsymbol{E}^*$ marks $Sup\overline{RN}CNb(E, \mathcal{G})$.*

*Proof.* In step 4, $\boldsymbol{R}_0^1$ is constructed that is a strict subautomaton of $\boldsymbol{H}$ and marks the supremal $L_m(\boldsymbol{G})$-closed sublanguage of $\boldsymbol{E}$. This corresponds to the first step in procedure (4.2).
In the first step of the iteration, $\boldsymbol{R}_1^1 \sqsubset \boldsymbol{R}_0^1 \sqsubset \boldsymbol{E}$ is built which marks the supremal normal sublanguage of $E$, $Sup\overline{N}(L_m(\boldsymbol{R}_0^1), L(\boldsymbol{G}))$. Next $\boldsymbol{R}_2^1 \sqsubset \boldsymbol{R}_1^1$ is obtained that marks $SupC(L_m(\boldsymbol{R}_1^1))$. Then the sequence $\boldsymbol{R}_{n+2}^1 \sqsubset \boldsymbol{R}_n^1 \sqsubset ... \sqsubset \boldsymbol{R}_2^1$ is obtained in which $\boldsymbol{R}_{i+2}^1$ marks the supremal $\boldsymbol{G}_i$-nonblocking sublanguages of $L_m(\boldsymbol{R}_i^1)$. This completes the first iteration of procedure (4.2). The algorithm proceeds with the next iterations. Since in each step of each iteration a strict subautomaton of an automaton of the previous step (and that of $\boldsymbol{H}$) is obtained, and all automata, including $\boldsymbol{H}$, are finite-state, then the algorithm terminates in at most $p$ steps (where $p$ is the number of states of $\boldsymbol{H}$). □

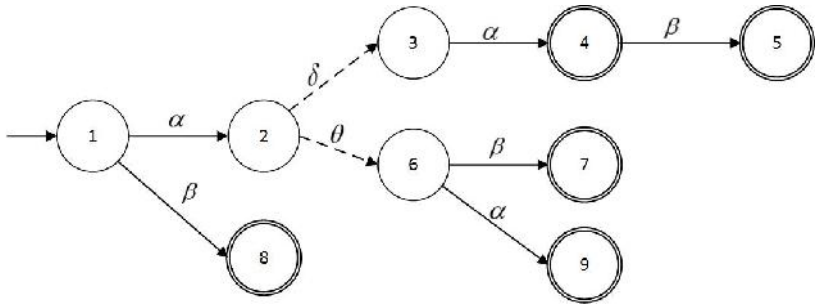
**Remark 4.2.** *The states of $\boldsymbol{H}$ are of the form $(x_{\boldsymbol{G}}, y, z_{\boldsymbol{G}})$. In supcon and supnblki procedures the component $z_{\boldsymbol{G}}$ is not used.*

## 4.4 Examples

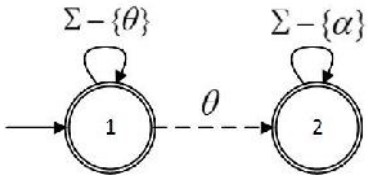
In this section we provide two illustrative examples for the computational procedures developed in this chapter. In the first example, we show the computation of supremal normal sublanguage. In the second example, we provide the solution of a robust nonblocking supervisory control problem with partial observation using Algorithm 4.2.

### 4.4.1 Example 1

Consider the plant $\boldsymbol{G}$ and design specification $\boldsymbol{K}$ in Fig. 4.1. It is assumed that events $\delta$ and $\theta$ are not observable, i.e. $\Sigma_{uo} = \{\delta, \theta\}$. In these representations, the doubly circled states denote the marked states of each automaton.




(a) $\boldsymbol{G}$




(b) $\boldsymbol{K}$

Figure 4.1: Example 4.4.1. Plant and legal behavior.

The legal marked behavior $\boldsymbol{E}$ can be built following the steps of Algorithm 3.2. This legal behavior is shown in Fig. 4.2.

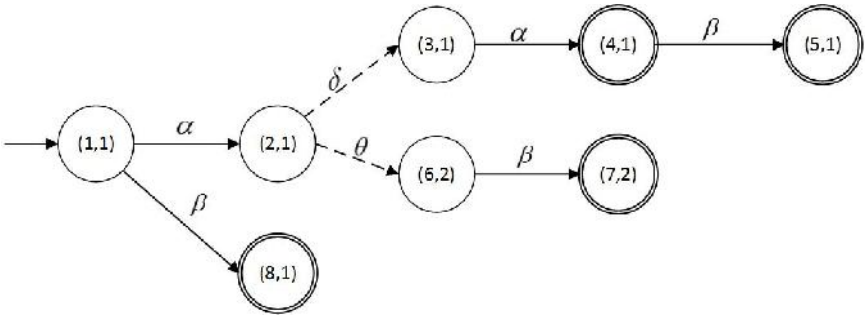
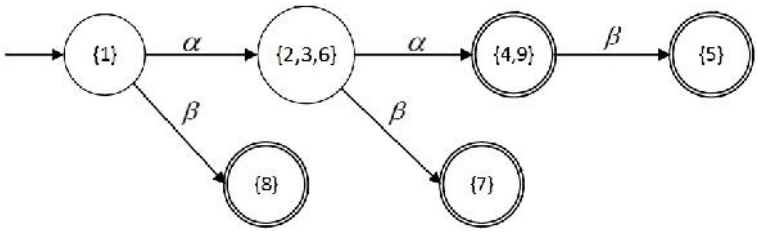


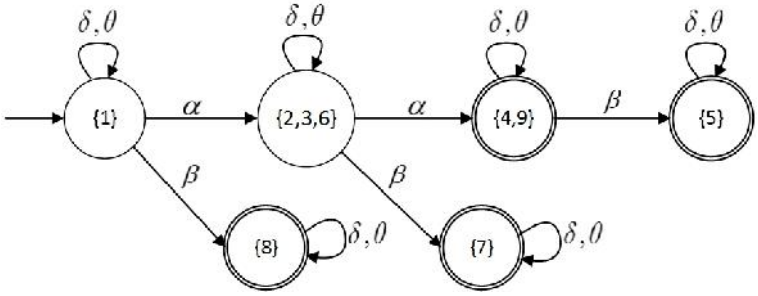

Figure 4.2: Example 4.4.1. Automaton $\boldsymbol{E}$ obtained from Algorithm 3.2.

Next automaton $\boldsymbol{H}$ is built.

1. In the first step, the plant automaton $\boldsymbol{G}$ is projected and is denoted by $\boldsymbol{G}_p$, Fig. 4.3(a).




(a) $\boldsymbol{G}_p$ after step 1




(b) $\boldsymbol{G}_s$ after step 2

Figure 4.3: Example 4.4.1. Procedures for obtaining automaton $\boldsymbol{H}$.

2. Next, self-loops of $\delta, \theta$ are added to the states. The resulting automaton $\boldsymbol{G}_s$ is shown in Fig. 4.3(b).

3. In the final step of this algorithm, automaton $\boldsymbol{H}$ is obtained by building the product of automata $\boldsymbol{E}$ and $\boldsymbol{G}_s$, Fig. 4.4.

Next, the iterative step of Algorithm 4.1 is executed.

- In the first iteration, by checking the pair-deficiency property in all states, we can see that state $(4, 1, \{4, 9\})$ is pair-deficient since the $z_{\boldsymbol{G}}$ part of the state

name is $\{4, 9\}$, therefore $|z_{\boldsymbol{G}}| = 2$. Next, we remove $(4, 1, \{4, 9\})$ and all of its matching pairs (it is clear that the matching pair for this state does not exist). The resulting automaton in this step, $\boldsymbol{H}_0$, is shown in Fig. 4.5(a).

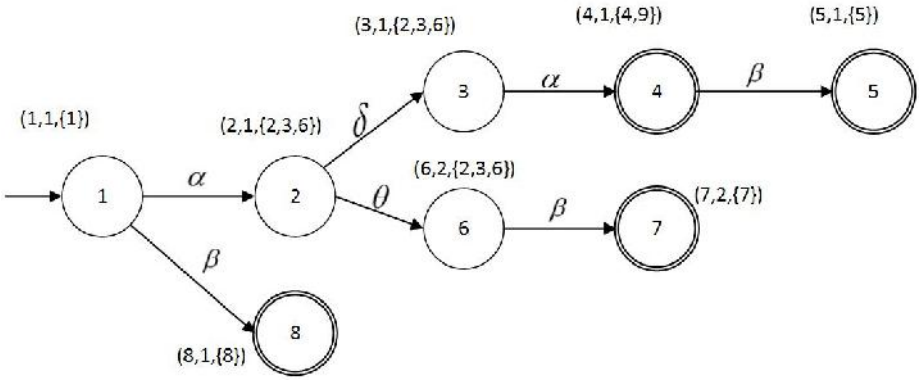
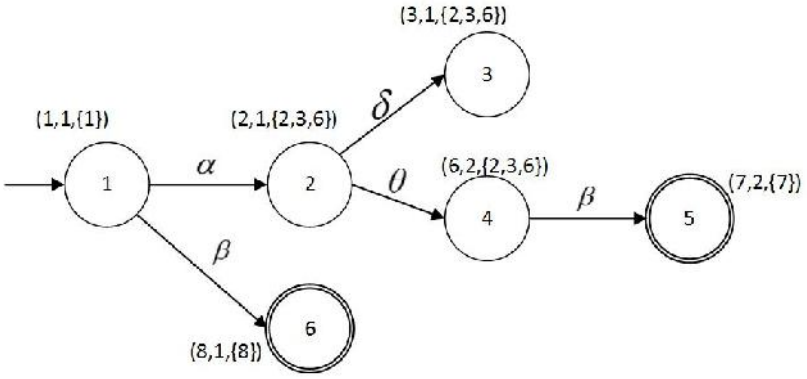
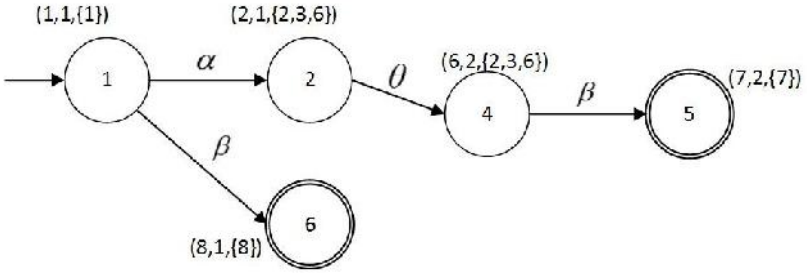



Figure 4.4: Example 4.4.1. Automaton $\boldsymbol{H}$ obtained from $\boldsymbol{E} \times P^{-1}P(\boldsymbol{G})$.

- Next automaton $\boldsymbol{H}_0$ is trimmed, Fig. 4.5(b).
- Since some states were removed in the first iteration we have to go through another iteration.
- In the first step of the second iteration we can see that state $(6, 2, \{2, 3, 6\})$ is pair-deficient since $|z_{\boldsymbol{G}}| = 3$ and there are only two matching pair states. Hence, state $(6, 2, \{2, 3, 6\})$ and its matching pair $(2, 1, \{2, 3, 6\})$ must be removed from $\boldsymbol{H}_0$. The resulting automaton is shown in Fig. 4.6.
- In this step, the automaton $\boldsymbol{H}_0$ is trimmed Fig. 4.6.
- No state removal is required in the third iteration and we can conclude that the resulting automaton represents the supremal normal sublanguage of $E$. Automaton $\boldsymbol{E}_{norm}$ is shown in Fig. 4.7.




(a) $\boldsymbol{H}_0$ after step 2.a.




(b) $\boldsymbol{H}_0$ after trimming in step 2.b

Figure 4.5: Example 4.4.1. Using Algorithm 4.1 in the first iteration.

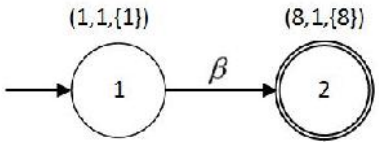



Figure 4.6: Example 4.4.1. Using Algorithm 4.1 in the second iteration.

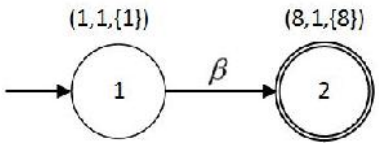



Figure 4.7: Example 4.4.1. Automaton $\boldsymbol{E}_{norm}$ marking the supremal normal sublanguage.

### 4.4.2 Example 2

In this example, a problem of robust nonblocking supervisory control with partial observation is solved. Consider plants $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ and their legal behaviors $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ in Fig. 4.8. Let the set of uncontrollable and unobservable events be $\Sigma_{uc} = \{u, v, w\}$ and $\Sigma_{uo} = \{e, c, w\}$ respectively. Algorithm 4.2 is used to solve the problem.

1. Automaton $\boldsymbol{G}$ is constructed using Algorithm 3.1 and is shown in Fig. 4.9.

2. The overall legal marked behavior can be obtained using Algorithm 3.2 (Fig. 4.10).

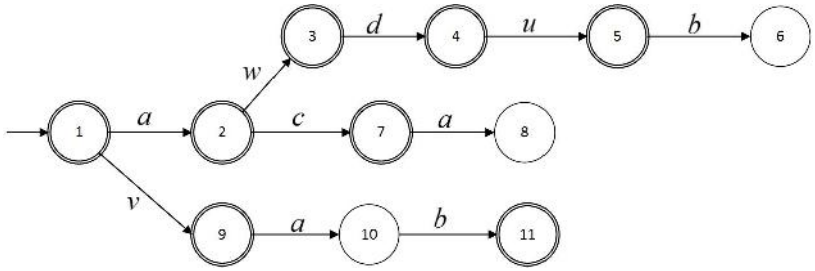
3. Automaton $\boldsymbol{H} = \boldsymbol{E} \times P^{-1}P(\boldsymbol{G})$ is depicted in Fig. 4.11.

4. The legal behavior $H$ is $L_m(\boldsymbol{G})$-closed, and $suprel(\boldsymbol{H}, \boldsymbol{G})$ returns $\boldsymbol{H}$ in Fig. 4.11.

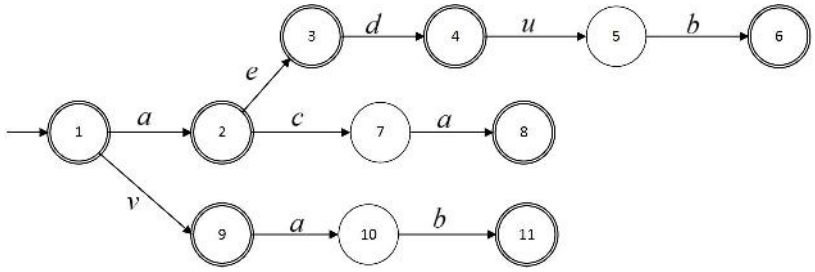
5. Now we can go through the iterative procedure in step 5 of Algorithm 4.2.

   - The first step in the first iteration $j = 1$ is to obtain an automaton marking the supremal normal sublanguage. Using Algorithm 4.1, it is easy to see that state $(d, 6, 6, \{(d, 6), (6, d)\})$ is pair deficient and has to be removed. Next, state $(d, 5, 5, \{(d, 5), (5, d)\})$ is removed after taking the trim operation since this state is not coreachable. In the next recursion of Algorithm 4.1, state $(5, d, 11, \{(d, 5), (5, d)\})$ is pair deficient now because its matching pair has been removed in the previous iteration. Therefore, this state has to be deleted from automaton $\boldsymbol{H}$. The remaining automaton is trim and after performing one more iteration, we can see that no new state will be removed and the recursion ends. Automaton $\boldsymbol{R}_1^1$ representing the supremal normal sublanguage of $\boldsymbol{H}$ is shown in Fig. 4.12(a).

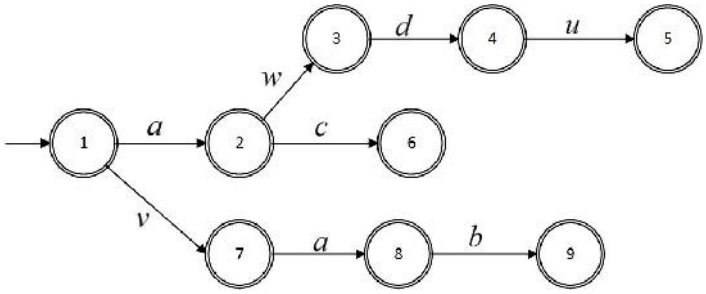
   - In the next step of this iterative procedure, it can be seen that states $(d, 4, 4, \{(d, 4), (4, d)\})$ and $(4, d, 10, \{(d, 4), (4, d)\})$ violate controllability



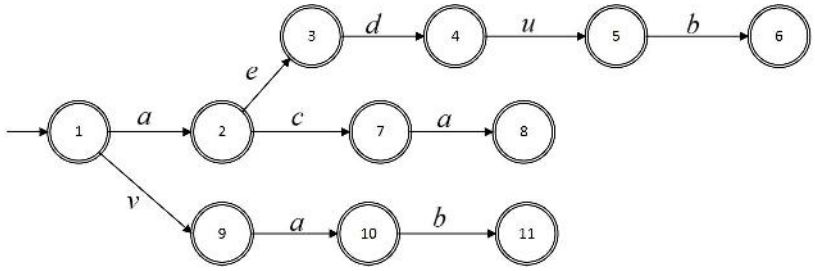

(a) $\boldsymbol{G}_1$




(b) $\boldsymbol{G}_2$




(c) $\boldsymbol{K}_1$




(d) $\boldsymbol{K}_2$

Figure 4.8: Example 4.4.2. Plants and legal behaviors automata.

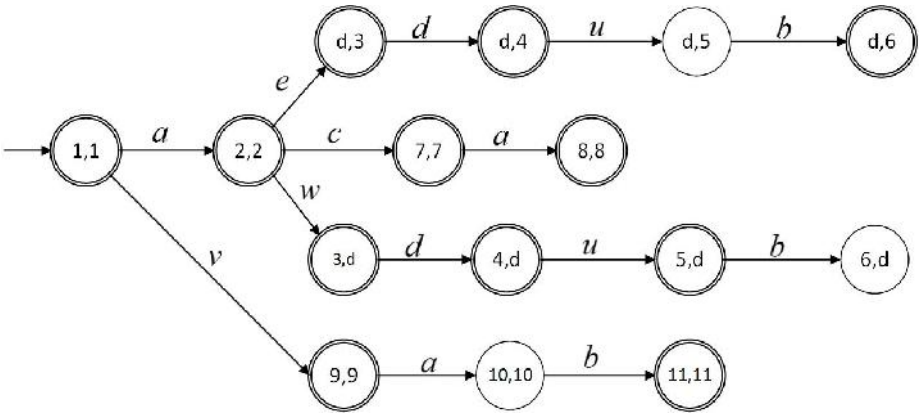



Figure 4.9: Example 4.4.2. Automaton $\boldsymbol{G}$.

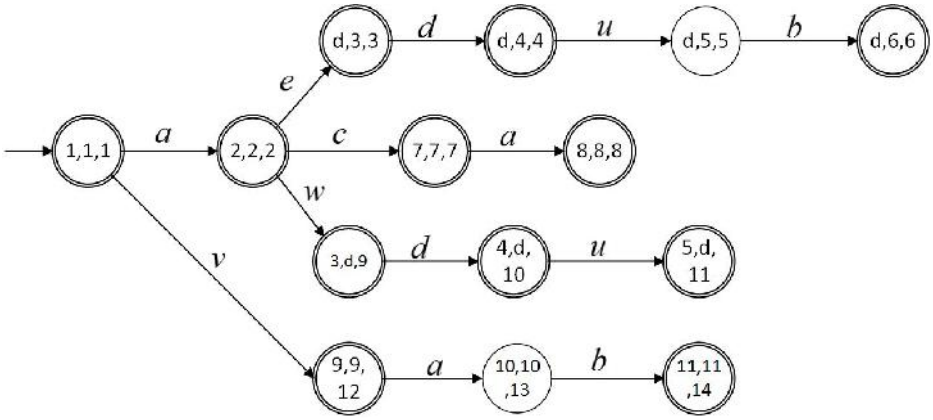



Figure 4.10: Example 4.4.2. Automaton $\boldsymbol{E}$.

(since event $u$ is uncontrollable and has been disabled) and would be removed from automaton $\boldsymbol{R}_1^1$ and results in $\boldsymbol{R}_2^1$ in Fig. 4.12(b).

- Next observe that the sequence $aca$ violates $\boldsymbol{G}_1$-nonblocking property. Following Algorithm 3.4 to obtain supremal $\boldsymbol{G}_1$-nonblocking sublanguage of $L_m(\boldsymbol{R}_2^1)$, state $(8,8,8,\{8,8\})$ is unmarked and since it is not coreachable and $x_1 = 8$ is not a dump state, $(8,8,8,\{8,8\})$ will be removed from $\boldsymbol{R}_2^1$ to obtain $\boldsymbol{R}_3^1$.

- Similarly, sequence $ac$ violates $\boldsymbol{G}_2$-nonblocking property and using Algorithm 3.4, state $(7,7,7,\{(2,2),(d,3),(3,d),(7,7)\})$ will be removed from $\boldsymbol{R}_3^1$ to obtain $\boldsymbol{R}_4^1$ in Fig. 4.13.

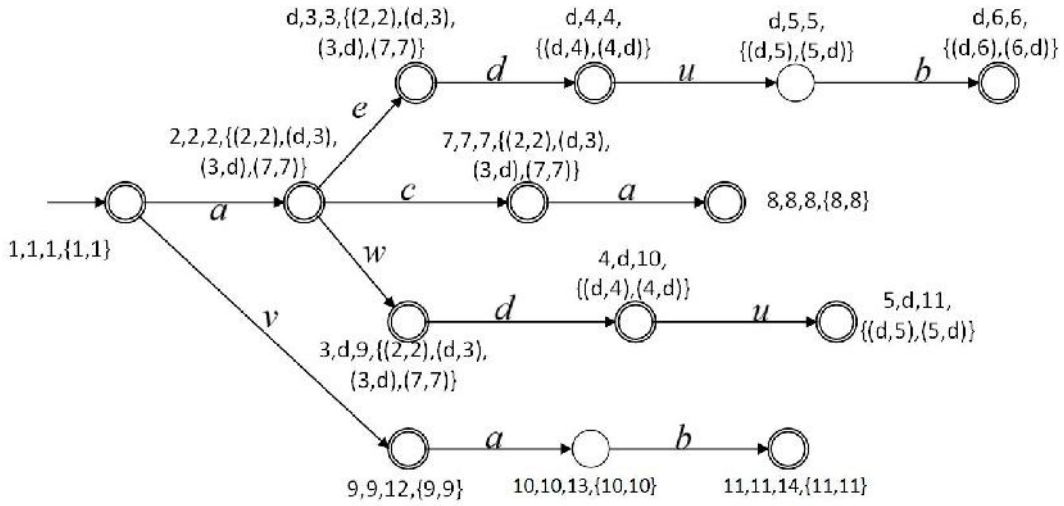
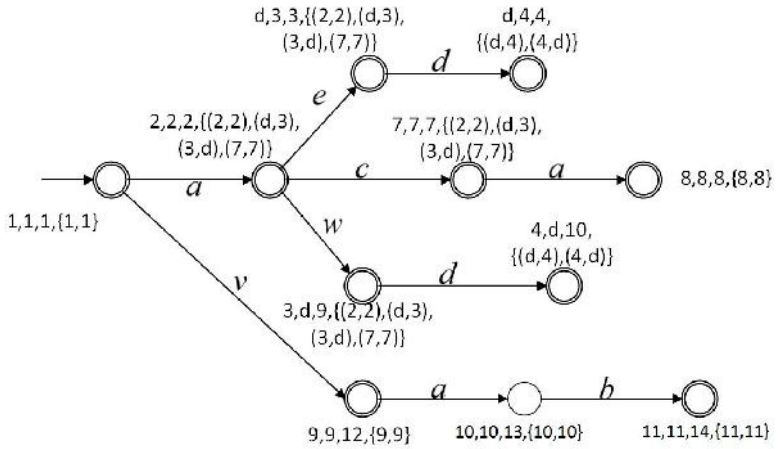
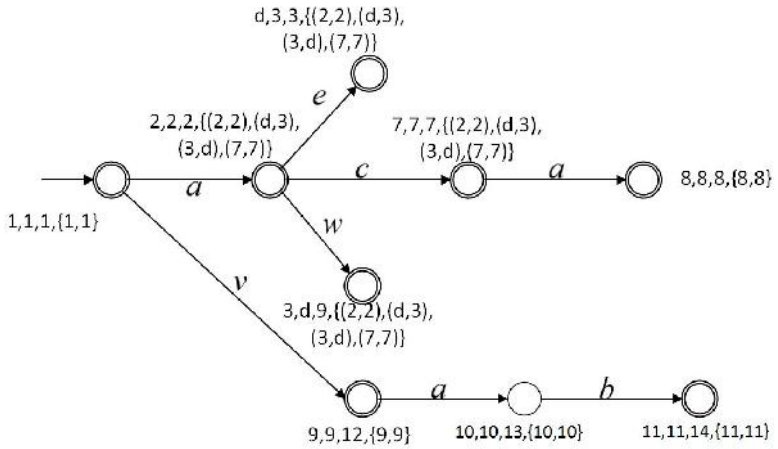



Figure 4.11: Example 4.4.2. Automaton $\boldsymbol{H}$ and $suprel(\boldsymbol{H}, \boldsymbol{G})$.

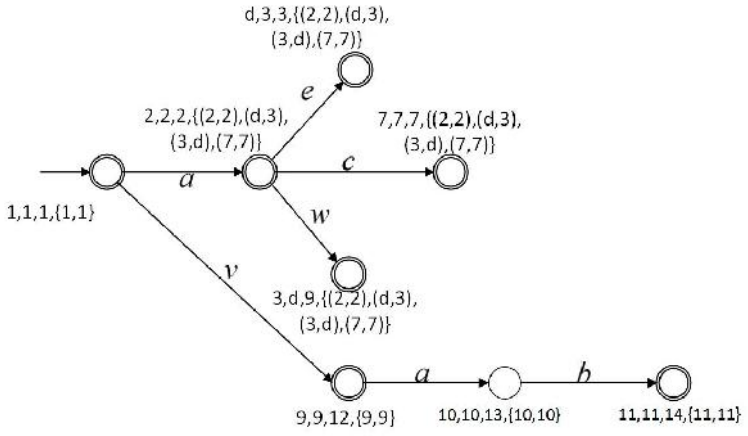
- Since $\boldsymbol{R}_4^1$ is not the same as $\boldsymbol{R}_0^1$, we proceed to the second iteration. Thus, $\boldsymbol{R}_0^2 := \boldsymbol{R}_4^1$ (Fig. 4.13). In the first step, the supremal normal sublanguage of $\boldsymbol{R}_0^2$ is computed. It is not difficult to see that state $(d, 3, 3, \{(2, 2), (d, 3), (3, d), (7, 7)\})$ is pair-deficient since $|\{(2, 2), (d, 3), (3, d), (7, 7)\}| = 4$ and there are only 3 matching pairs. Thus, state $(d, 3, 3, \{(2, 2), (d, 3), (3, d), (7, 7)\})$ and all of its matching pairs $(2, 2, 2, \{(2, 2), (d, 3), (3, d), (7, 7)\})$ and $(3, d, 9, \{(2, 2), (d, 3), (3, d), (7, 7)\})$ are removed from automaton $\boldsymbol{R}_0^2$. The remaining automaton is trim and after one more iteration we can obtain automaton $\boldsymbol{R}_1^2$ in Fig. 4.14(b) which represents the supremal normal sublanguage of $\boldsymbol{R}_0^2$.
- In the next step, no new state or transition is removed from the automaton $\boldsymbol{R}_1^2$ since it is controllable. Therefore, $\boldsymbol{R}_2^2$ is the same as $\boldsymbol{R}_1^2$.
- Furthermore, since $\boldsymbol{R}_2^2$ is $\boldsymbol{G}_1$-nonblocking and $\boldsymbol{G}_2$-nonblocking, automata $\boldsymbol{R}_3^2$ and $\boldsymbol{R}_4^2$ are the same as $\boldsymbol{R}_2^2$.




(a) $\boldsymbol{R}_1^1$




(b) $\boldsymbol{R}_2^1$




(c) $\boldsymbol{R}_3^1$

Figure 4.12: Example 4.4.2. Automata in iteration $j = 1$.

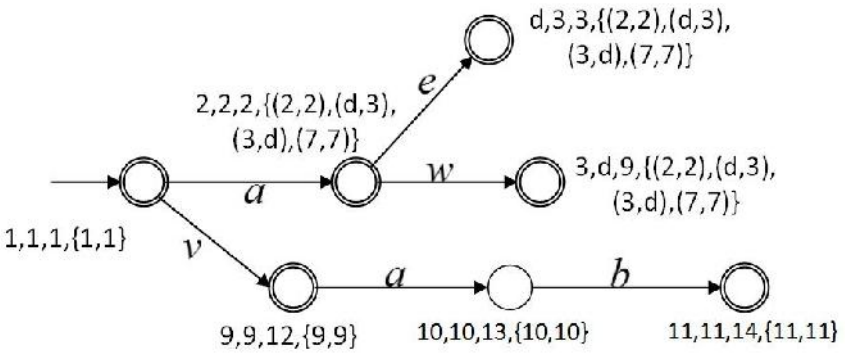
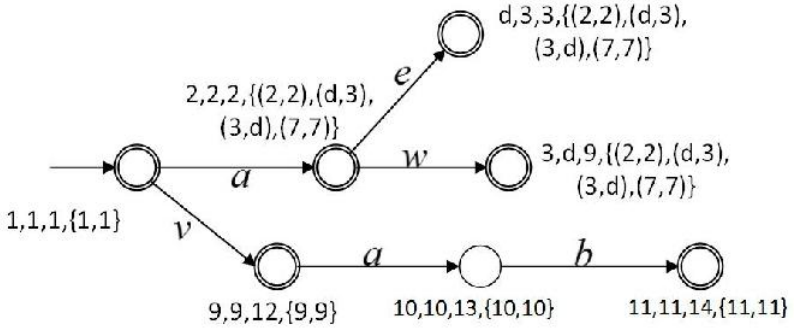
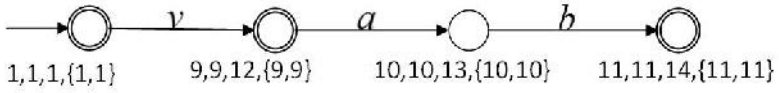



Figure 4.13: Example 4.4.2. Automaton in $\boldsymbol{R}_4^1$ iteration $j = 1$.




(a) $\boldsymbol{R}_0^2$




(b) $\boldsymbol{R}_1^2 = \boldsymbol{R}_2^2 = \boldsymbol{R}_3^2 = \boldsymbol{R}_4^2$

Figure 4.14: Example 4.4.2. Automata in iteration $j = 2$.

- As $\boldsymbol{R}_4^2$ is not the same as $\boldsymbol{R}_0^2$, we should go through a third iteration by substituting $\boldsymbol{R}_0^3$ with $\boldsymbol{R}_3^2$. It can be easily checked that this automaton is normal, controllable, $\boldsymbol{G}_1$-nonblocking and $\boldsymbol{G}_2$-nonblocking. Thus, no new state and transition will be removed in this iteration and $\boldsymbol{R}_4^3 = \boldsymbol{R}_3^3 = \boldsymbol{R}_2^3 = \boldsymbol{R}_1^3 = \boldsymbol{R}_0^3$. Hence, automaton $\boldsymbol{E}^* = \boldsymbol{R}_4^3$ in Fig. 4.15 marks the

supremal $L_m(\boldsymbol{G})$-closed, normal, controllable and $\boldsymbol{G}_1$-nonblocking and $\boldsymbol{G}_2$-nonblocking sublanguage of the legal behavior $\boldsymbol{E}$.

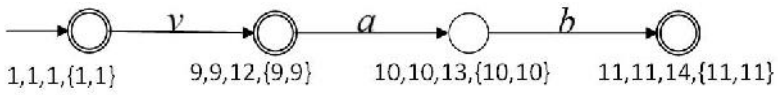



Figure 4.15: Example 4.4.2. Automaton $\boldsymbol{R}_4^3$ in iteration $j = 3$.

## 4.5 Conclusion

This chapter presents a computational procedure for the supremal $L_m(\boldsymbol{G})$-closed, controllable, normal and $\boldsymbol{G}_i$-nonblocking sublanguage. This sublanguage provides a solution for the robust nonblocking supervisory control problem under partial event observation.

# Chapter 5

# Conclusions

## 5.1 Summary

In this thesis, we present computational procedures for providing solutions to the robust nonblocking supervisory control problem (RNSCP) in discrete-event systems. First we consider the case of control with full event observation and focus on computing the maximally permissive (optimal) supervisor such that the system under supervision satisfies certain desired closed-loop properties. In the supervisory control theory, maximally permissive supervisors can be characterized in terms of supremal sublanguages. In this study, we assume that the plant can be modeled as finite-state automata and the specifications are regular languages. Having these assumptions, we prove the finite convergence of our proposed iterative algorithm. Each step of the algorithm involves removing states and transitions from an initial automaton until a final automaton which marks the desired supremal sublanguage is reached. The main challenge of developing algorithms for computing supremal sublanguages is to form the initial automaton in a way that adequate information about the important sequences are embedded in the states and transition structures.

The solution of robust nonblocking supervisory control problem has been addressed in the literature. In order to obtain the optimal solution to this problem, we

develop a computational algorithm for calculating the supremal $\boldsymbol{G}$-nonblocking sublanguage. Next, an algorithm for computing supremal $L_m(\boldsymbol{G})$-closed sublanguage is developed. Combining these procedures with the algorithm for computation of supremal controllable sublanguage provided in literature, leads us to the optimal solution to the robust nonblocking supervisory control problem with full event observation. We implement the proposed computational procedures in MATLAB environment using Discrete Event Control Kit (DECK). We also apply our procedures to a fault recovery problem in a simplified spacecraft propulsion system.

To tackle the RNSCP with partial event observation, we extend the computational algorithms for finding the solution of the robust problem with full observation by developing a computational procedure for obtaining the supremal normal sublanguage. In order to find the supremal normal sublanguage, first we provide an algorithm to form a new initial automaton which has enough information about the projected model as well as the states and transition structures. Then we develop an iterative procedure to compute supremal normal sublanguage. Using this procedure along with the computational algorithm provided for the case of full observation we present the procedure for the solution of robust nonblocking supervisory control problem with partial observation. The resulting solution is the supremal solution based on the normality property. Furthermore we show that the algorithm converges in a bounded number of steps.

## 5.2 Future Work

The directions for future research may include the following:

- The computer code developed in this thesis implements the algorithms for control with full event observation. The extension of the code to the case of partial observation would be a very useful step.

- In this thesis we developed iterative algorithms for computing the supremal

sublanguages that lead us to the optimal solutions for robust nonblocking supervisory control problem. The downside of the iterative algorithms is their high computational complexity. Hence, finding certain conditions on the plants, specifications and projection map under which a non-iterative algorithm can be developed is one of the future areas of research. Non-iterative algorithms have been found for subsets of non-robust supervisory control problems (e.g. [18]).

- Another possible area for the research is the development of computational procedures using symbolic calculations. This approach has provided far more efficient algorithms in verification problems (e.g., [5, 6]) and in (non-robust) supervisory control (e.g., [1], [16] and [26]).

- The implementation of the algorithms can be further improved using compilers to speed up the calculation processing time. The code can be also optimized by means of parallel computing and efficient memory management.

# Bibliography


[1] S. Balemi, G. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control*, 38(7):1040–1059, Jul 1993.

[2] F. Boroomand and S. Hashtrudi Zad. A limited lookahead policy in robust nonblocking supervisory control of discrete event systems. Proc. *American Control Conference (ACC), 2013*, pages 935–939, Washington, DC, June 2013.

[3] S. Bourdon, M. Lawford, and W.M. Wonham. Robust nonblocking supervisory control of discrete-event systems. *IEEE Transactions on Automatic Control*, 50(12):2015–2021, Dec. 2005.

[4] R. Brandt, V. Garg, R. Kumar, F. Lin, S. Marcus, and W.M. Wonham. Formulas for calculating supremal controllable and normal sublanguages. *Systems & Control Letters*, 15(2):111 – 117, 1990.

[5] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug 1986.

[6] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10 states and beyond. *Inf. Comput.*, 98(2):142–170, June 1992.

[7] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2008.

[8] E. Chen and S. Lafortune. On nonconflicting languages that arise in supervisory control of discrete event systems. *Systems & Control Letters*, 17(2):105 – 113, 1991.

[9] X. Y. Chen and S. Hashtrudi Zad. A direct approach to robust supervisory control of discrete-event systems. *Proc. Canadian Conference on Electrical and Computer Engineering*, pages 957–962, Niagara Falls, ON, Canada, May 2008.

[10] H. Cho and S. Marcus. On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. *Mathematics of Control, Signals and Systems*, 2(1):47–69, 1989.

[11] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260, Mar 1988.

[12] J. Cury and B. Krogh. Design of robust supervisors for discrete event systems with infinite behaviors. In *Proceedings of the 35th IEEE Conference on Decision and Control, 1996*, volume 2, pages 2219–2224 vol.2, Dec 1996.

[13] J. Cury and B. Krogh. Robustness of supervisors for discrete-event systems. *IEEE Transactions on Automatic Control*, 44(2):376–379, Feb 1999.

[14] M. de Queiroz, J. Cury, and W.M. Wonham. Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 15(4):375–395, 2005.

[15] C. Economakos and F. Koumboulis. Modular implementation of robust supervisory controllers for discrete event systems. *IEEE Transactions on Automatic Control*, 53(6):1559–1563, July 2008.

[16] J. Gunnarsson. *Symbolic methods and tools for discrete event dynamic systems.* Ph.D. dissertation, Linköping Studies in Science and Technology, Linköping, Sweden, 1997.

[17] S. Hashtrudi Zad, R. Kwong, and W.M. Wonham. Supremum operators and computation of supremal elements in system theory. *SIAM Journal on Control and Optimization*, 37(3):695–709, 1999.

[18] S. Hashtrudi Zad, M. Moosaei, and W.M. Wonham. On computation of supremal controllable, normal sublanguages. *Systems & Control Letters*, 54(9):871 – 876, 2005.

[19] J. Hopcroft, R. Motwani, and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Pearson/Addison Wesley, 2007.

[20] R. Kumar and V. Garg. Optimal supervisory control of discrete event dynamical systems. *SIAM Journal on Control and Optimization*, 33(2):419–439, 1995.

[21] R. Kumar, V. Garg, and S. I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems & Control Letters*, 17(3):157 – 168, 1991.

[22] R. Kumar and S. Takai. A framework for control-reconfiguration following fault-detection in discrete event systems. In *International symposium on fault detection, supervision and safety of technical processes*, pages 848–853, 2012.

[23] M. Leeds, R. Eberhardt, and R. Berry. Development of the Cassini spacecraft propulsion subsystem. In *32nd Joint Propulsion Conference and Exhibit*, AIAA 96-2864, Lake Buena Vista, FL, 1996.

[24] F. Lin. Robust and adaptive supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 38(12):1848–1852, Dec 1993.

[25] F. Lin and W.M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173 – 198, 1988.

[26] C. Ma and W.M. Wonham. Nonblocking supervisory control of state tree structures. *IEEE Transactions on Automatic Control*, 51(5):782–793, May 2006.

[27] S. McLane and G. Birkhoff. *Algebra*. Chelsea Publishing Series. American Mathematical Society, 1999.

[28] P. Morgan. Cassini spacecraft's in-flight fault protection redesign for unexpected regulator malfunction. *Proc. 2010 IEEE Aerospace Conference*, pages 1–14, Big Sky, MT, March 2010.

[29] A. Paoli, M. Sartini, and S. Lafortune. Active fault tolerant control of discrete event systems using online diagnostics. *Automatica*, 47(4):639 – 649, 2011.

[30] S.-J. Park. Robust and nonblocking supervisory control of nondeterministic discrete event systems with communication delay and partial observation. *International Journal of Control*, 85(1):58–68, 2012.

[31] S.-J. Park and J.-T. Lim. Non-blocking supervision for uncertain discrete event systems with internal unobservable transitions. *IEE Proceedings : Control Theory and Applications,*, 152(2):165–170, March 2005.

[32] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[33] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, Jan 1989.

[34] A. Saboori and S. Hashtrudi Zad. Fault recovery in discrete event systems. In *Proc. ICSC Congress on Computational Intelligence Methods and Applications*, Istanbul, Turkey, 6 pages, Dec. 2005.

[35] A. Saboori and S. Hashtrudi Zad. Robust nonblocking supervisory control of discrete-event systems under partial observation. *Systems & Control Letters*, 55(10):839 – 848, 2006.

[36] S. Takai. Robust supervisory control of a class of timed discrete event systems under partial observation. *Systems & Control Letters*, 39(4):267 – 273, 2000.

[37] S. Takai. Verification of robust diagnosability for partially observed discrete event systems. *Automatica*, 48(8):1913 – 1919, 2012.

[38] S. Takai. Robust prognosability for a set of partially observed discrete event systems. *Automatica*, 51(0):123 – 130, 2015.

[39] S. Takai and T. Ushio. Effective computation of an lm(g)-closed, controllable, and observable sublanguage arising in supervisory control. *Systems & Control Letters*, 49(3):191 – 200, 2003.

[40] F. Wang, S. Shu, and F. Lin. Robust supervisory control of networked discrete event systems. In *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 981–988, Monticello, IL, Oct 2013.

[41] Q. Wen, R. Kumar, and J. Huang. Framework for optimal fault-tolerant control synthesis: Maximize prefault while minimize post-fault behaviors. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 44(8):1056–1066, Aug 2014.

[42] Q. Wen, R. Kumar, J. Huang, and H. Liu. A framework for fault-tolerant control of discrete event systems. *IEEE Transactions on Automatic Control*, 53(8):1839–1849, Sept 2008.

[43] W.M. Wonham. *Supervisory Control of Discrete Event Systems*. Systems Control Group, Dept. of Electrical and Computer Engineering, University of Toronto, Canada,, available at http://www.control.utoronto.ca/DES, 2014.

[44] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

[45] T.-S. Yoo, S. Lafortune, and F. Lin. A uniform approach for computing supremal sublanguages arising in supervisory control theory. *Technical Report, Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor*, 2001.

[46] Discrete event control kit (deck 1.2013.11). `http://www.ece.concordia.ca/~shz/deck`. [Online].

[47] Computer code for robust control. `http://www.ece.concordia.ca/~shz/deck/robust`. [Online].

# Appendix A

# Cassini Propulsion System

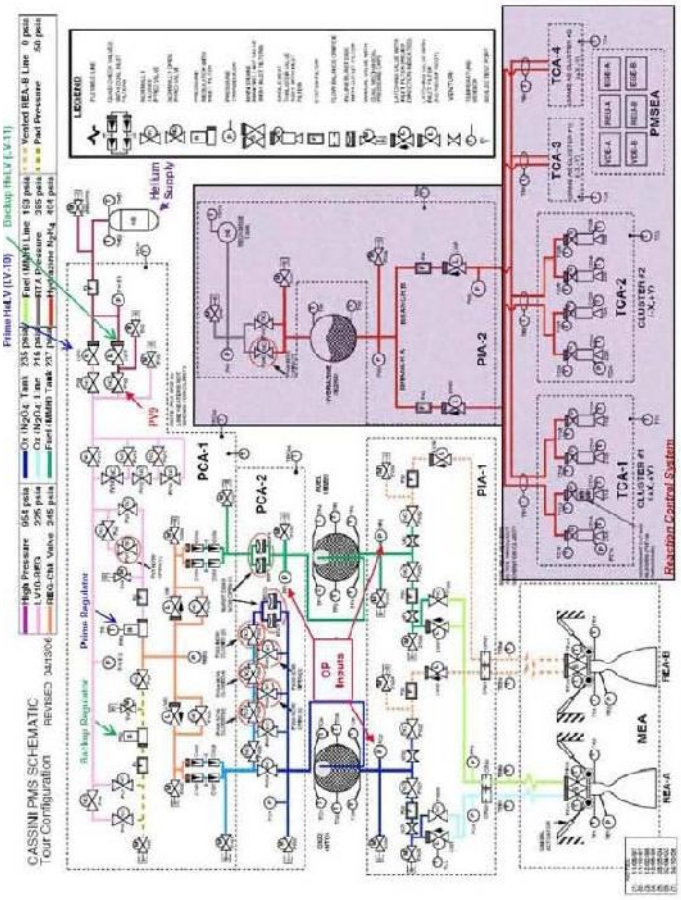



Figure A.1: Cassini propulsion module subsystem schematic [28].

# Appendix B

# Discrete Event Control Kit (DECK)

DECK is a toolbox written in MATLAB for the analysis and synthesis of supervisory control problem of discrete-event systems. In this appendix we review the functions of this toolbox [46].

1. Automaton: Creates an automaton model for use by the toolbox (DECK). Number of states, the matrix of transition list and a vector of marked states are the inputs to this function.

2. Automatonchk: This function verifies the validity of the automaton model. It gets the automaton model as input.

3. Complement: Returns the complement of the input deterministic automaton.

4. Controllable: This function determines if the input automaton is controllable with respect to the second input automaton and the input vector of uncontrollable events.

5. Deterministic: Converts the nondeterministic input automaton to a deterministic automaton.

6. Isnondet: This function determines if the input automaton is nondeterministic.

7. Product: Computes the product of a finite number of input automata.

8. Project: Finds a deterministic automaton to represent the projected model of the input automaton and the input vector of unobservable events.

9. Reach: Computes the reachable states of a transition list matrix and the vector of source states.

10. Reachable: This function computes the reachable subautomaton of an input automaton.

11. Selfloop: Adds selfloop to the input automaton.

12. Supcon: Finds the supremal controllable sublanguage of the first input automaton with respect to the second automaton and the input vector of uncontrollable events.

13. Sync: This function computes the synchronous product of a finite number of input automata.

14. Trim: Finds the trim (reachable and coreachable) subautomaton of the input automaton.

# Appendix C

# Computer Code for Robust Control

This appendix presents the functions written in DECK to implement the computational procedures proposed in the thesis for solving nonblocking robust supervisory control (with full event observation)

1. $Gco = completeE(G, Ea)$

   This function converts an automaton to a complete automaton (i.e., an automaton with a total transition function).

   **function** Gco=completeE(G,Ea)
   COMPLETEE make the transition **function** of a deterministic automaton complete
   SYNTAX: Gco=completeE(G)
   Gco=completeE(G,Ea)

   INPUTS: G Input deterministic automaton
   Ea List of events (vector)

   OUTPUTS: Gco Output deterministic automaton

2. $[Gt, States] = Gu(Gi)$

   $Gu$ builds the union automaton of the input automata and also returns the state labels for each new state of union automaton.

```
function [Gt, States]=Gu(Gi)
Gu          Union of two automata
SYNTAX:     G=Gu(Gi)
INPUTS:     Gi      The input automaton cell (Plants)
OUTPUTS:    Gt      The union automaton
            States      State numbers list in the G union
            automaton
```

3. $[E, EStates] = legal(Gi, Ki)$

   The legal behavior and its state labels are built in this function from input plants and specifications.

```
function [E, EStates]= legal(Gi,Ki)
Legal     The Legal behavior
SYNTAX:     [E, EStates]= legal(Gi,Ki)
INPUTS:     Gi      cell of input plants automaton
            Ki      cell of input specs automaton
OUTPUTS:    E       The overall legal behavior
            EStates     List of the current state numbers
            of each plant
```

4. $[ER, ERStates] = suprel(Gt, GtSt, E, ESt)$

   This function computes the supremal $L_m(\boldsymbol{G})$-closed sublanguage of $L_m(\boldsymbol{E})$ with respect to the union plant model Gt.

```
function [ER, ERStates]=suprel(Gt,GtSt,E,ESt)
SUPR  Supremal Relative-Closed Sublanguage
```

SYNTAX: [ER, ERStates]=suprel(Gt, GtSt, E, ESt)

INPUTS: G Plant (deterministic) automaton
GSt List of State numbers in automaton G
E Specification (deterministic) automaton
ESt List of State numbers in automaton E

OUTPUTS: ER Trim (deterministic) automaton marking supremal Lm(G)−closed sublangage.
ERStates List of State numbers in automaton ER

5. $[EN, ENStates] = supnblki(Gi, i, E, ESt, XmSp)$

This function computes the supremal $\boldsymbol{G}_i$-Nonblocking sublanguage of $L_m(\boldsymbol{E})$.

**function** [EN, ENStates]=supnblki(Gi, i, E, ESt, XmSp)

SUPNBLKI Supremal Gi−nonblocking Sublanguage

SYNTAX: [EN, ENStates]=supnblki(Gi, i, E, ESt, XmSp)

INPUTS: Gi Plant (deterministic) automaton
i The number of the specific automaton Gi in the model cell
E Specification (deterministic) automaton
ESt Specification state list
XmSp The list of marked states in the legal behavior

OUTPUTS: EN Trim (deterministic) automaton marking supremal Gi−nonblocking sublangage.
ENStates SupNbGi state list

6. $[K, KSt] = supC(H, HS, G, GS, Euc, ESt)$

This function computes the supremal controllable sublanguage of $L_m(\boldsymbol{H})$ with respect to the union plant G and uncontrollable events $Euc$.

**function** [K, KSt]=supC(H, HS, G, GS, Euc, ESt)

SUPC Supremal Controllable Sublanguage

SYNTAX: [K KSt]=supC(H,HS,G,GS,Euc)

INPUTS: H Specification (deterministic) automaton

HS Spec state list

G Plant (deterministic) automaton

GS Plant State list

Euc Uncontrollable events (vector)

OUTPUTS: K Trim (deterministic) automaton marking supremal controllable sublangage

KSt SupC state list

7. $[H, HSt] = SupRCN(Euc, varargin)$

This is the main function for the Robust Nonblocking Supervisory Control Problem (RNSCP) which uses the other functions in DECK and the developed functions here to find the maximally permissive solution of RNSCP.

**function** [H,HSt]=SupRCN(Euc,varargin)

SUPRCN Supremal Relative−Closed, Controllable, Gi−nonblocking Sublanguage

SYNTAX: K=supRCN(Euc,G1,E1,...,Gi,Ei,...,Gn,En)

INPUTS: Euc List of uncontrllable events (vector)

Gi Plant models automata

Ei Spec models automata

OUTPUTS: H Trim (deterministic) automaton marking supremal Lm(G)−closed, controllable, Gi−nonblokcing sublangage.

HSt States numbers representing the current state in each plant