Limited Lookahead Policies for Robust Supervisory Control of Discrete Event Systems

Farzam Boroomand

A thesis
in
The Department
of
Electrical & Computer Engineering

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

August 2013

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Farzam Boroomand**

Entitled: **Limited Lookahead Policies for Robust Supervisory Control of Discrete Event Systems**

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

Master of Applied Science (Electrical and Computer Engineering)

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

Signed by the final examining commitee:

______________________________ Chair
Dr. S. Williamson

______________________________ External Examiner
Dr. W. F. Xie

______________________________ Examiner
Dr. K. Khorasani

______________________________ Supervisor
Dr. S. Hashtrudi Zad

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

__________ 2013 ______________________________
Dr. C. W. Trueman
Interim Dean, Faculty of Engineering and Computer Science

Abstract

Limited Lookahead Policies for Robust Supervisory Control of Discrete Event Systems

Farzam Boroomand

In this thesis, Limited Lookahead Policies (LLP) have been developed for Robust Nonblocking Supervisory Control Problem (RNSCP) of discrete event systems. In the robust control problem considered here, the plant model is assumed to belong to a given finite set of DES models.

The introduced supervisor computes the control action in online fashion and it is named Robust Limited Lookahead (RLL) supervisor. In comparison with offline supervisory control, RLL supervisor can reduce the complexity associated with the computation of control law as it looks at the behavior of system at the current state and of a limited depth in future.

Since a conservative policy is adopted here, the behavior of the system under supervision of the RLL supervisor is generally more restrictive than the optimal offline supervisor. A sufficient condition is presented under which a limited lookahead window can guarantee the optimality (maximal permissiveness) of the RLL supervisor.

In some problems, the required window length for maximally permissive RLL supervisor may become unbounded. To overcome this limitation RNSCP with State information (RNSCP-S) is studied and solved resulting in a state-based RLL (RLL-S) supervisor.

The results of this thesis can be regarded as an extension of previous work in the literature on limited lookahead policies for (non-robust) supervisory control to the case of nonblocking robust supervisory control.

The robust limited lookahead design procedures are implemented in MATLAB environment and applied to two examples involving spacecraft propulsion systems.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Shahin Hashtrudi Zad, whose guidance, patience and support from the initial to the final stage enabled me to write this thesis. Completion of my masters would not have been possible without his continuous help and support. I appreciate all his contributions of time and ideas.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis studies robust nonblocking supervisor control problem of discrete event systems where the exact model of the system is unknown but it belongs to a finite set of possible models. The solution to the robust nonblocking control problem in literature is computed off-line before the system starts execution. Finding the off-line supervisory control map can be computationally infeasible due to the size of the system. Even storing the resulting supervisor may not be possible because of limited available computer memory. Further, the complete description of the system might be unavailable in the design stage in the case of time-varying systems. To tackle these issues, we present an online solution for robust nonblocking supervisory control problem based on limited lookahead policy. The online supervisor computes the control action for the current state based on some knowledge of the future behavior of the plant.

In the remainder of this chapter, we first introduce conventional (non-robust) supervisory control problem in Section 1.1. In Section 1.2 robust supervisory control with model uncertainty is introduced. Section 1.3 introduces online supervisory control with lookahead policies. After a literature review on robust supervisory control and lookahead policies in Section 1.4, the objectives and contributions of this thesis are presented in Section 1.5. The organization of this thesis is finally presented in Section 1.6.

1.1 Supervisory Control of Discrete Event Systems

Discrete-event systems (DES) constitute a class of systems with a discrete state-space whose dynamics can be characterized by sequences of discrete events. A transition within the state-space of a DES happens upon the occurrence of a discrete event.

Example 1.1. As an example of DES consider pressure isolation assembly of a simplified version of the Cassini Main Propulsion System (CMPS) [1] presented in Figure 1.1. The system consists of four pyro-valves ($PV_i,\ i = 1, ..., 4$), two regular valves ($V_1,\ V_2$) and two pressure sensors. The valves have two states which are open or closed. The pressure sensors measure the pressure in the upstream of each engine which can be either low or high. The overall state of the system can be expressed by considering combination of the components' states. ■

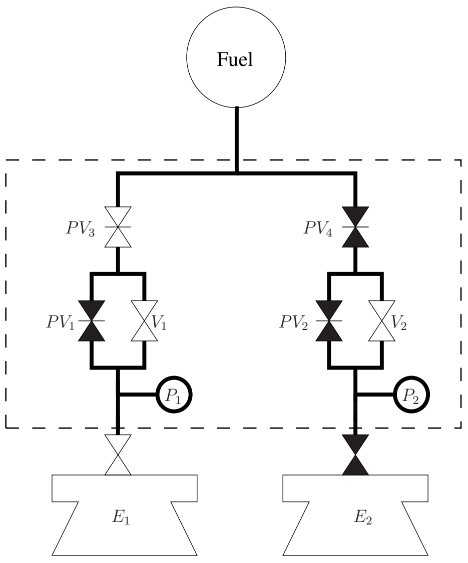


Figure 1.1: Simplified CMPS.

In general, some behaviors of the system may be undesirable. In the preceding example, we would like to avoid the states where both engines are fired simultaneously. We must assure that there exists a fuel path between the tank and only one of the engines at each time. The goal of supervisory control is to determine how the behavior of a system should be altered in order to achieve the desired behavior and avoid the undesirable behavior. This can be achieved through the Ramadge-Wonham supervisory control framework presented in [2], [3], [4].

In this framework, a supervisor positioned in a feedback loop as shown in Figure 1.2, alters the behavior of the plant by disabling or enabling controllable events so that the behavior of system is restricted to the desirable behavior. The supervisor also prevents the system from being blocked. In Example 1.1 *opening* and *closing* of the valves are *controllable* events. The changes in pressure sensor readings are *uncontrollable* as they are dependent to some other events in the system.

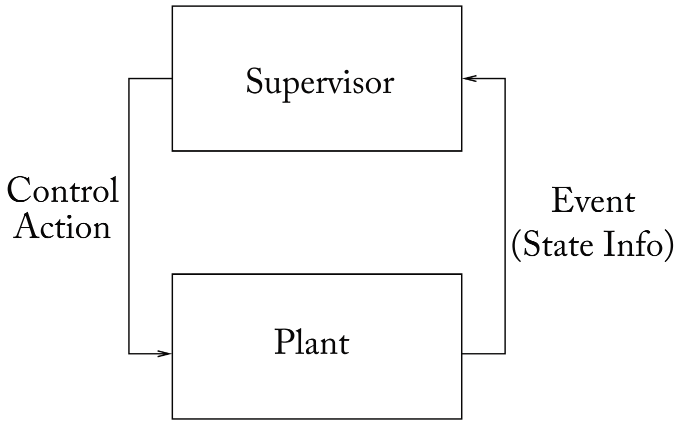


Figure 1.2: Supervisory control framework.

In many practical cases, the supervisor can be regarded as a finite-state automaton. Each state of this automaton corresponds to a unique control action. Specifically, the presence of an event in a state indicates that the event is enabled. If an event is absent in a state of the supervisor automaton, then it means that the event will be disabled by the supervisor (should the plant attempt to execute the event). The control action can also be based on the plant states (instead of executed event sequences). In this case, the supervisor is a state feedback law (a map from plant state to control action).

1.2 Robust Supervisory Control

Conventional supervisory control assumes that the system model is known and it is certain at the design stage. This assumption is not always true, the system model may change during the operation due to different reasons (e.g. failure in a component). Also, the design objectives may vary over the course of operation. Robust control is developed in order to deal with this uncertainty in the systems. Robust control approaches have been studied in both continuous and discrete systems.

For discrete event systems different approaches exist to study modeling uncertainty. In this thesis, we assume that the exact model of the DES is unknown but it belongs to a finite set of possible models. The specification for each of the possible models might be different from the others. The goal is to design a robust supervisor which works properly for all plant models. This means that the plants should remain nonblocking under supervision, while their behavior remain within the legal limits. The main problem of this thesis is to design a maximally permissive online supervisor which ensures safety and nonblocking requirements of all the possible models.

Two problems that can be regarded as special cases of the robust control problem are fault recovery [22] and supervisory control with multiple marked state sets [5]. In fault recovery problem, the true model of a system can be either the normal model or any of the normal-failure models. In case of a component failure, a robust supervisor should guarantee that the system fulfills safety requirement while remaining nonblocking.

Consider the propulsion system in Example 1.1. Assume that each of the regular valves V_1 or V_2 has a failure mode, *stuck-closed*. Before the system starts operation, we do not know if any of the failures will happen or not. Considering a single failure scenario, we will have three different possible models of the system. A normal model, where there is no failure in the system, and two normal-failure models, where either V_1 or V_2 fails and become stuck-closed.

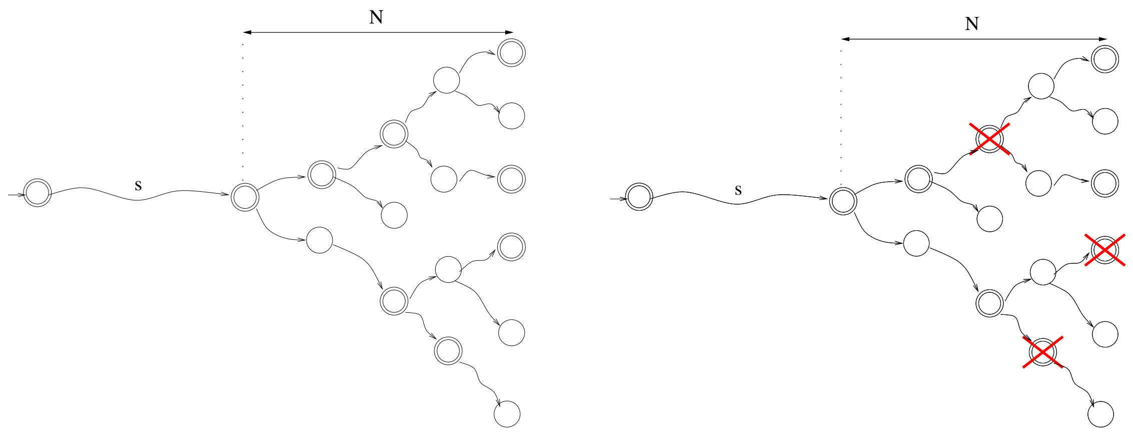
In addition to uncertainty in components' models, the mission goals might be different before and after a failure. By mission goal, we mean the states in a system which corresponds to completion of a task. In our example, the goal before a failure is: to fire one of the engines and possibly switch from one engine to another. The objective after a failure is usually less strict. In this example, we want to make sure that we can always generate thrust in case of a single failure. This multi-objective problem, which is robust supervisory control with multiple marking, can be formulated as a special case of robust problem.

1.3 Lookahead Policies for Supervisory Control

In off-line supervisory control, the control action (off-line supervisor) for all possible plant behavior is computed at the design stage before the plant starts operation. This off-line computation is a one-time investment, which guarantees satisfaction of the design specification. However, when the size of the plant state space becomes very large, it might be impossible to accommodate this enormous computation. In addition, the size of an off-line supervisor may become too large to be stored in computer memory. Consider a propulsion system with thousands of failure models. Storing an off-line supervisor for such a system requires memory which may not be available. On the other hand, in online computation of the control action there is no need for storing a design.

In light of the above mentioned considerations, [7] presented supervisory control with Limited Lookahead Policy (LLP) as an alternative to off-line computation.

In LLP supervisory control, the control action is computed online after execution of each event. This computation is based on an *N-step* prediction of the future behavior of the system. Assuming that an event sequence "*s*" has been generated, the control action



(a) All possible behavior after s up to N steps (b) All legal behavior after s up to N steps

Figure 1.3: Tree expansion in the LLP scheme.

following "*s*" will be a list of enabled events. This procedure must be repeated after the execution of every event. This means that the off-line design problem (which may be computationally expensive) is replaced with repetitive computation of some similar but smaller problems.

In LLP scheme, the set of event sequences (language) that the system can generate in the next *N-steps* is initially generated (Fig. 1.3 (a)). Next, a subset of this set which includes legal sequences according to a specification is constructed. The two sets (languages) are finite and are represented by their tree generators. The latter language is represented by a sub-tree of the former one (Fig. 1.3 (b)). These trees change dynamically from step to step and must be updated after the execution of every event.

The control action is then computed by finding supremal controllable sublanguage of the *predicted* legal behavior with respect to the *predicted* closed behavior. The control loop for LLP supervision is shown in Figure 1.4 [8].

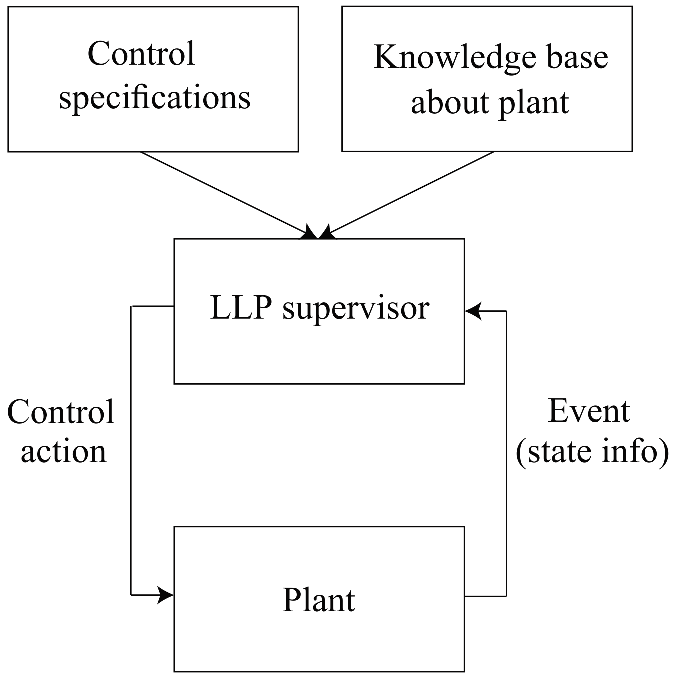


Figure 1.4: Supervisory Control with Lookahead Policies.

In the LLP framework, the behavior of the system beyond a trace (event sequence) with the length of "*N*" is completely unknown. These traces are called *pending traces*. Two different policies can be adopted toward pending traces. A pending trace may lead to an illegal string by executing a set of uncontrollable events. To avoid the risk, we may adopt a conservative policy toward the pending traces which treats all pending traces to be illegal. In this sense, conservative policy assumes the worst situation and thus a conservative supervisor is more restrictive than an optimal off-line supervisor. On the other hand, we may assume that all the pending traces lead to a legal marked string which is an optimistic attitude. In this situation, the system is given maximum freedom to proceed. However, some of the pending traces may not lead to a legal marked state through a controllable path. This means that an optimistic supervisor is more permissive than an optimal off-line supervisor.

Variable Lookahead Policies (VLP) are later presented as a modification of LLP in order to compute the control action more efficiently. In this scheme, the tree expansion ends whenever the control decision can be made unambiguously or whenever the boundary of N-level tree is reached, whichever comes first [8].

In this thesis, we extend LLP with conservative policy to solve the robust nonblocking supervisory control problem. We also obtain a set of sufficient conditions for optimality of the robust LLP supervisor.

1.4 Literature Review

1.4.1 Robust Control

Robust supervisory control of DES was first studied by Lin in [12]. In the proposed framework the author assumes that the exact model of the DES is unknown while it belongs to a finite set of possible models $\mathcal{G} = \{G_1, ..., G_n\}$. It also assumes a common specification, *K*, which is a sublanguage of marked behaviors of the possible models ($K \subseteq \bigcap_i L_m(G_i)$). The goal within the framework is to find a supervisor which works with all the possible DES models.

Takai [18] relaxes the constraint $K \subseteq \bigcap_i L_m(G_i)$ in [12]; however, it only studies the case of prefix-closed specifications. Thus, it does not consider the nonblocking property. Later Takai [14], generalized the framework presented by [12] to the case of timed DES. Park-Lim [19], [20] generalizes the framework presented by [12] to work with nondeterministic DES.

In another attempt, [21] generalizes the framework of [12] by considering non-prefix closed specifications that are not necessarily the same for all the models. The nonblocking property of the models is ensured by a sufficient condition referred to as nonconflicting property. Later [23] extended the results of [21] by (i) replacing nonconflicting property with *G-nonblocking* property (which results in a necessary and sufficient condition set the solution) and (ii) considering control under partial observation.

The authors in [28], extend robust supervisory control in [21] to the hierarchical framework presented by [32].

In another framework of robust supervisory control, Curry-Krogh [29] and Takai [15], [17] model the uncertainty in modeling using ω-*languages*. Having a supervisor which works for a nominal plant, the framework aims to maximize the (infinite) set of plants for which the supervisor is suitable.

In this thesis we are interested in the problem of robust nonblocking supervisory control studied in [21] and [23]. We modify conventional LLP supervisor to solve the Robust Nonblocking Supervisory Control Problem (RNSCP) in an online fashion.

The presented approach is referred to as *Robust Limited Lookahead* (RLL) Supervisor. It considers the behavior of the possible DES models by finding tree-expansions N-step beyond the currently executed trace.

1.4.2 Lookahead Policies

Limited Lookahead Policy

In conventional (non-robust) DES framework of Ramadge-Wonham the supervisors are computed entirely before the system begins operation (*off-line* design). Limited lookahead supervision is referred to as online control because the control action is computed after the occurrence of each event, while the system is operational. Limited Lookahead Policies (LLP) are proposed in [7] to address the following difficulties of traditional supervisory control:

- Finding off-line supervisor could be computationally infeasible due to the complexity of DES or its legal behavior;
- Complete description of the DES or its legal behavior might be unavailable due to time-varying property of the system.

In view of these observations, [7] proposed LLP supervisor for online supervisory control of DES, where control action is computed based on *N-step* truncation of DES and its legal behavior. The estimation of the plant model is in form of a tree-expansion. A tree-expansion distinguishes between states based on their event histories. Once the plant and its specification are estimated, the rest is very similar to off-line supervisory control. In order to compute the control action, [7] has taken two different attitudes toward traces of the length *N* in truncated model of DES (which are called *pending traces*): 1- conservative, and 2- optimistic. The conservative policy assumes the worst case, where all pending traces lead to an illegal string with execution of a set of uncontrollable events. Generally speaking, the LLP supervisor with conservative attitude is less permissive compared with the off-line supervisor. On the other hand the optimistic policy assumes that every pending trace lead to a legal marked state through a controllable path. Thus, the LLP supervisor with optimistic attitude is more permissive compared with the optimal supervisor. In summary, with an insufficient length of lookahead window the LLP supervisor can be overly-accepting with optimistic policy or overly-restrictive following the conservative policy.

The main challenge in the scheme of LLP supervisory control is to determine the minimum required length of lookahead window which guarantees optimality of the online supervision. [7] presented a set of sufficient conditions under which the LLP supervisor performs as well as optimal supervisor with complete information about the plant and its specification.

Variable Lookahead Policy

The authors of [7] later observed that length of lookahead window can be modified for some of the strings. Thus, in [8] they proposed a method for efficient calculation of online supervisory control decision. Like LLP supervisor, the new approach works based on *N-step* tree-expansion of the systems behavior, while tree expansion terminates whenever the control action can be made unambiguously or whenever the boundary of the N-step projection reaches. This new approach is called *Variable Lookahead Policy* (VLP). VLP is an efficient way of implementation for LLP.

Both LLP and VLP supervisors take linguistic approach for online computation of control action. This causes the required window size for the optimal supervisor to be infinite in some cases. [9] presented a variable lookahead policy with state information named VLP-S. The window size required for optimal VLP-S is always bounded for finite state plants. VLP-S requires the specification to be a sub-automaton of the plant. If this is not the case, [9] provides an approach for modifying the automata appropriately. Finally, [6] presents a VLP for robust supervisory control of DES.

Other works in the area of online supervisory control of DES are as follows. Extension-based Limited Lookahead (ELL) Policy is presented [10], [11]. The ELL supervisor avoids the notion of pending traces by extending the behavior of the plant with any arbitrary behavior beyond the N-step lookahead window. [13] presents an estimate-based Limited Lookahead Policy which studies a special case where the specification is a closed language. [24] generalizes the LLP framework to work with probabilistic DES. [25] studies the methods for estimation of the size of state-space for lookahead tree-expansions.

In this thesis we study the robust nonblocking supervisory control with a conservative limited lookahead policy. We also study the problem using state information in order to guarantee the optimality of the Robust Limited Lookahead (RLL) supervisor.

1.5 Thesis Contributions

We can summarize the major contributions of this thesis as follows:

- We generalize the conservative limited lookahead policy presented in [7] to the case of robust nonblocking control. We call the new supervisor *Robust Limited Lookahead*

(RLL). We show the monotonicity property of the RLL supervisor in terms of the length of the lookahead window. We determine the closed-form expression of the language generated by the DES under supervision of the RLL supervisor. We also present the lower bound of N which guarantees the optimal performance of the RLL supervisor (under certain conditions).

- We study the case where the required length of window size for optimality becomes infinite. To deal with this situation, we extend the linguistic approach of the RLL supervisor to the case of supervisory control with state information. We refer to the new supervisor as RLL-S. We present algorithms for online computation of RLL-S based control actions.

- Furthermore, the procedures required for our lookahead policies are implemented in MATLAB environment using the toolbox Discrete Event Control Kit (DECK) [44].

- We apply our proposed method to two different problems in propulsion systems of Viking and Cassini spacecrafts. We study the effectiveness of the approach in each of the two cases.

1.6 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 reviews the supervisory control of DES, as well as robust supervisory control, and LLP supervision. Chapter 3 formulates the problem studied in the thesis and provides the solution to online robust non-blocking supervisory control with limited lookahead policy. It also discusses the properties of the RLL supervisor. Chapter 4 transforms the RNSCP to a state-based problem and then solves the online problem with state information. Chapter 5 discusses two application examples of the proposed method. The thesis is concluded at the end with a discussion and presenting possible future research.

1.7 Summary

This chapter provides an introduction to this thesis by introducing and motivating the Robust Limited Lookahead (RLL) supervision problem. The relevant literature has been reviewed, and the contributions of the thesis have been explained.

Chapter 2

Background

This chapter contains two parts. The first part presents a quick review on formal languages, conventional and robust supervisory control, and online supervisory control scheme. The second part presents preliminaries on the properties of post languages and other results which are used to prove main results of this thesis. This chapter is mainly based on [42], [41].

2.1 Discrete Event Models

A DES is formally defined as a dynamic system with a discrete state space, performing asynchronous event driven transitions within the state-space. Discrete event systems are typically employed to model man-made system which are governed by man-made rules rather than physical laws. Examples of this type of systems are manufacturing systems, telecommunication protocols and database systems. Different models are proposed in literature to model a DES each of which focuses on a different aspect of a DES. For example, finite-state automaton takes a logical deterministic approach while Markov chain models probabilistic aspect of a DES. In this research we model a DES using a finite-state automaton.

2.1.1 Languages

The transitions of a DES in state-space follows the occurrences of discrete events. We assume that each event happens abruptly and that only one event happens at a time. Let Σ be the event set of a DES also called an **alphabet**. A sequence of events over the alphabet Σ is called a **string** or a **trace**. The empty string, denoted by ϵ, is defined as a string with no event. A set of strings over an alphabet Σ is called a **language**. The language that includes all the strings except ϵ is defined as:

$$\Sigma^+ = \{\alpha_1 ... \alpha_n | n > 0, \alpha_i \in \Sigma\}$$

Then define:

$$\Sigma^\star = \Sigma^+ \cup \{\epsilon\}$$

2.1.2 Operations on Languages

Basic set operations are defined for languages as the language are sets of strings. Such basic operations on two languages L_1 and L_2 are intersection $L_1 \cap L_2$, union $L_1 \cup L_2$, complement L_1^{co} and negation $L_1 - L_2$. Some other operations on languages are defined in the following:

Definition 2.1. ***Prefix-closure:*** Let $L \subseteq \Sigma^*$, then $\overline{L} := \{s \in \Sigma^* | \exists t \in \Sigma^*, st \in L\}$. ■

The string s is then called to be a prefix of t and it is denoted by $s \leq t$. If s is a prefix of t and $s \neq t$ then we denote it with $s < t$.

Definition 2.2. ***Post-language*** of a language $L \subseteq \Sigma^*$ after trace $s \subseteq \Sigma^*$ is defined as:

$$L/s := \{t \in \Sigma^* \mid st \in L\}$$

■

Definition 2.3. ***Quotient*** of L by M with $L, M \subseteq \Sigma^*$ is defined as:

$$L/M := \{w \in \Sigma^* \mid (\exists x \in M) wx \in L\}$$

■

For a sequence s, let $|s|$ denotes the length of s, i.e., the number of events in s when $s \neq \epsilon$ and $|s| = 0$ when $s = \epsilon$.

Definition 2.4. ***Truncation*** of L to a non-negative integer N is defined as:

$$L|_N := \{t \in L \mid |t| \leq N\}$$

■

Some useful properties of the post-language operation which are employed in this thesis are presented in Lemma 2.1.

Lemma 2.1. Let $K_1, K_2 \subseteq \Sigma^*$ and $s \in \Sigma^*$. Then:

$$\begin{aligned}
1)\quad & (K_1 \cap K_2)/s = K_1/s \cap K_2/s \\
& (K_1 \cup K_2)/s = K_1/s \cup K_2/s \\
2)\quad & \overline{K_1}/s = \overline{K_1/s} \\
3)\quad & \forall s \in \overline{K_1} \Rightarrow (K1/s)K_2 \subseteq (K_1K_2)/s
\end{aligned}$$

■

2.1.3 Automata

Modeling a DES as a language is a good approach to study the theoretical aspect of discrete events systems and theory of supervisory control. However, it is not practical for developing computational algorithms. For this purpose, other modeling methods such as automaton and Petri nets are used. In this part we discuss automaton as a visual tool for representing a DES. An automaton is also called a generator. A deterministic automaton is a five-tuple:

$$G = (X, \Sigma, \delta, x_0, X_m)$$

where Σ is the set of alphabet, X is the set of states with initial state x_0, δ is the partial transition function with $\delta : X \times \Sigma \to X$ and X_m denotes the set of marked states. The marked states typically signing the completion of a task or some reset mode. The behavior of a DES G is described by its prefix-closed language and marked language.

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

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

2.1.4 Operations on Automata

Definition 2.5. ***Equivalent automata*** We say that two automata G_1 and G_2 are equivalent iff $L(G_1) = L(G_2)$ and $L_m(G_1) = L_m(G_2)$. ■

Definition 2.6. Let $G = (X, \Sigma, \delta, x_0, X_m)$. The automaton G is **reachable** if there is a path to every state, x, from the initial state, x_0. In other words, for every $x \in X$, there exists a trace s such that $\delta(x_0, s) = x$ is defined. The reachable subautomaton of G is denoted by G_r. ■

Definition 2.7. Let $G = (X, \Sigma, \delta, x_0, X_m)$. The automaton G is **coreachable** if there is a path from every state to a marked state. ■

An automaton is said to be **nonblocking** if there exists a path from every reachable state to a marked state. The formal definition of nonblocking (for deterministic automaton

) given in terms of its languages is provided below.

Definition 2.8. ***Nonblocking automaton*** An automaton G is called **nonblocking** if and only if $\overline{L_m(G)} = L(G)$. ■

Next we discuss the trim and complement operation.

Definition 2.9. ***Trim operation:*** For an automaton G, *Trim(G)* is reachable and coreachable subautomaton of G. ■

A generator is trim if all of its states are reachable and coreachable.

Definition 2.10. ***Complement:*** Let $G = (X, \Sigma, \delta, x_0, X_m)$, then G^{comp} marks $L_m(G) = \Sigma^\star - L_m(G)$ and generates $\Sigma^\star$. ■

Definition 2.11. ***Product (Meet):*** Consider two automata:

$$G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{m,1}),\ G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{m,2})$$

Then

$$G_1 \times G_2 := \textit{Reachable part of } (X_1 \times X_2, \Sigma_1 \times \Sigma_2, \delta, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

where:

$$\delta((x_1, x_2), \sigma) := \begin{cases} (\delta_1(x_1, \sigma), \delta(x_2, \sigma)) & \text{if } \delta_1(x_1, \sigma) \text{ and } \delta(x_2, \sigma) \text{ are defined} \\ undefined & \text{otherwise} \end{cases}$$ ■

By the definition at the state (x_1, x_2) the event σ appears in $G_1 \times G_2$ if and only if G_1 and G_2 can execute σ from x_1 and x_2 respectively. Consequently, it can be shown that product operation represents the intersection of languages of automata G_1 and G_2:

$$L(G_1 \times G_2) = L(G_1) \cap L(G_2)$$

$$L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$$

The synchronous product is used in modeling the joint operation of two (or more) automata.

Definition 2.12. ***Synchronous Product (Parallel Composition)*** Let $G_1 = (X_1, \Sigma_1, \delta_1, x_{0,1}, X_{m,1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{0,2}, X_{m,2})$. Then define:

$$G_1 \parallel G_2 := \text{Reachable part of}(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \delta, (x_{0,1}, x_{0,2}), X_{m,1} \times X_{m,2})$$

where

$$\delta((x_1, x_2), \sigma) := \begin{cases} (\delta_1(x_1,\sigma), \delta_2(x_2,\sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \wedge \delta_1(x_1,\sigma)! \wedge \delta_2(x_2,\sigma)! \\ (\delta_1(x_1,\sigma), x_2) & \text{if } \sigma \in \Sigma_1 - \Sigma_2 \wedge \delta_1(x_1,\sigma)! \\ (x_1, \delta_2(x_2,\sigma)) & \text{if } \sigma \in \Sigma_2 - \Sigma_1 \wedge \delta_2(x_2,\sigma)! \\ \text{undefined} & \text{otherwise} \end{cases} \quad \blacksquare$$

The supervisory control theory partitions the event set into two disjoint sets, controllable and uncontrollable, denoted by Σ_c and Σ_{uc} respectively. Assume the legal marked behavior is represented by with a language K. The objective is to design a supervisor V that limits the behavior of the plant G to its legal behavior K with ensuring nonblocking property of the system under supervision V/G, by disabling only controllable events. The supervisor itself is a map $V : \Sigma^* \to \Gamma$ where $\Gamma = \{\gamma \in 2^\Sigma \mid \gamma \supseteq \Sigma_{uc}\}$. We present a preliminary on supervisory control in the next section.

2.2 Supervisory Control

In the supervisory control theory which is introduced by Ramadge and Wonham [2], the plant is modeled by an automaton $G = (X, \Sigma, \delta, x_0, X_m)$. The theory partitions the event set Σ into two disjoint sets, controllable events and uncontrollable events denoted by $\Sigma =$

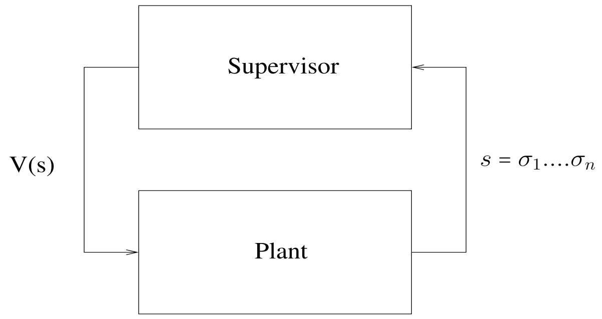


Figure 2.1: Supervisory control scheme

Σ_c and Σ_{uc} respectively. The objective is to limit the marked behavior of the plant to a desirable legal sublanguage K by disabling some controllable events while assuring that the nonblocking property holds for the system under supervision. Supervisory control is a feedback control as shown in Figure 2.1. The supervisor observes events which are generated by the plant. Considering the legal behavior K, the supervisor V makes the control decision about the enablement of events. The supervisor must also ensure that the system under supervision is nonblocking.

For $s \in L(G)$, $V(s)$ denotes the set of enabled events by the supervisor. Note that V can not disable any uncontrollable event. The language generated by the plant under supervision of supervisor V, $L(G,V)$, can be recursively defined as:

1) $\epsilon \in L(G,V)$

2) If s $\in L(G,V)$ and $\sigma \in V(s)$, then $s\sigma \in L(G,V)$

Further, the marked language of the plant under supervision, $L_m(G,V)$, is defined by the following equation:

$$L_m(G,V) = L_m(G) \cap L(G,V)$$

Nonblocking Supervisory Control Problem (NSCP)

As previously discussed supervisory control aims to design a supervisor which limits the behavior of the plant under supervision to legal language while maintaining nonblocking property for the system under supervision. The Nonblocking Supervisory Control Problem (NSCP) is described in the following.

Nonblocking Supervisory Control Problem: Let $G = (X, \Sigma, \delta, x_0, X_m)$ and $\Sigma = \Sigma_c \uplus \Sigma_{uc}$. Also suppose the legal marked behavior of the plant is described by $K \subseteq L_m(G)$ and $K \neq \varnothing$. Find a supervisor V such that:

$$\begin{aligned} &1) \quad L_m(G, V) \subseteq K \\ &2) \quad L(G, V) = \overline{L_m(G, V)} \end{aligned}$$

■

A supervisor which solves NSCP is called a **nonblocking supervisor**. There is no unique solution for NCSP in general. Define:

$$\mathcal{V} := \{V \mid V \text{ solves NSCP}\}$$

A supervisor V is called to be maximally permissive if

$$\forall V' \in \mathcal{V} : L_m(G, V) \subseteq L_m(G, V) \wedge L(G, V) \subseteq L(G, V)$$

The following definitions are necessary for introducing the solution to NSCP.

Definition 2.13. ***Controllability*** Let $\Sigma_u \subseteq \Sigma$ be the set of uncontrollable events and automaton $G = (X, \Sigma, \delta, x_0, X_m)$. A language $K \subseteq \Sigma^*$ is controllable with respect to G if

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$$

It is easy to show that if the language K is not controllable, it has a supremal controllable sublanguage denoted by $\sup C(K,G)$. (If K is controllable, then $K = \sup C(K,G)$)

Definition 2.14. $\mathbf{L_m(G)}$***-closure:*** We say that a language $K \subseteq L_m(G)$ is $\mathrm{L_m(G)}$-closed if

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

Obviously, $\varnothing$ is $\mathrm{L_m(G)}$-closed. It is easy to show that $\mathrm{L_m(G)}$-closure is preserved under the union operation of languages and consequently any language K has a supremal $\mathrm{L_m(G)}$-closed sublanguage which is denoted by $\sup R(K,G)$. The supremal $\mathrm{L_m(G)}$-closed sublanguage of K can be calculated with the formula $\sup R(K,G) = K - (L_m(G) - K)\Sigma^*$.

Lemma 2.2. [41] If a language K is $\mathrm{L_m(G)}$-closed, then $\sup C(K,G)$ remains $\mathrm{L_m(G)}$-closed.

Theorem 2.1. [41] Let $G = (X, \Sigma, \delta, x_0, X_m)$, $\Sigma_u \subseteq \Sigma$ be the set of uncontrollable events. Also suppose $K \subseteq L_m(G)$ be a nonempty language. Then there exists a supervisor V such that $L_m(G,V) = K$ and $L(G,V) = \overline{K}$ if:

$$1) \quad K \text{ is controllable w.r.t G} : \overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$$

$$2) \quad K \text{ is } \mathrm{L_m(G)}\text{-closed} : \overline{K} \cap L_m(G) = K$$

Optimal Solution to NSCP

We previously formulated the NSCP. Note that in the case of full-observation, there always exists a maximally permissive (optimal) supervisor which solves NSCP. The optimal solution for NSCP is characterized by the supremal $\mathrm{L_m(G)}$-closed, controllable sublanguage of the legal language K. By Lemma 2.2, the supremal controllable sublanguage of an $\mathrm{L_m(G)}$-closed language remains $\mathrm{L_m(G)}$-closed. Consequently, the maximally permissive solution is obtained by first finding $\sup R(K,G)$ (if K is not $\mathrm{L_m(G)}$-closed) and then applying the $\sup R(K,G)$ and then applying the operator $\sup C(\textbf{.},G)$.

2.2.1 Robust Supervisory Control

In the conventional (non-robust) supervisory control, it is assumed that the model of the system is known and it does not change over time. On the other hand, this is not always the case. In some problems the system can change over time or there might be other uncertainties in the modeling of the system. It is then necessary to take the uncertainty of modeling into account. Robust supervisory control is developed to cover these cases.

There are several ways for modeling the uncertainty and consequently for the formulation of robust supervisory control. We are interested in the problem of robust supervisory control studied in [21] and [23].

Problem Formulation

Suppose the model of the system belongs to a finite set of possible models $\mathcal{G} = \{G_1, ..., G_n\}$. Each model has its own legal marked behavior $K_i \subseteq L_m(G_i)$. It is assumed that if an event α belongs to the event sets of G_i and G_j $(i \neq j)$, then α controllable (respectively uncontrollable) for G_i implies α controllable (respectively uncontrollable) for G_j. In the Robust Nonblocking Supervisory Control Problem (RNSCP) the goal is to design a supervisor V such that $L_m(G_i, V) \subseteq K_i$ and $\overline{L_m(G_i, V)} = L(G_i, V)$ $(i = 1, ..., n)$. Let G be a DES such that $L(G) = \bigcup_{i=1}^{n} L(G_i)$ and $L_m(G) = \bigcup_{i=1}^{n} L_m(G_i)$. The answers to RNSCP can be expressed using the following sublanguages. For $K \subseteq \Sigma^*$ define:

1. Controllable sublanguages of K

$$C(K, G) = \{E \subseteq K \mid \overline{E}\Sigma_{uc} \cap L(G) \subseteq \overline{E}\}.$$

2. $L_m(G)$-closed sublanguages of K

$$R(K,G) = \{E \subseteq K \mid \overline{E} \cap L_m(G) = E\}.$$

3. G_i-nonblocking sublanguages of K

$$N_b(K,G_i) = \{E \subseteq K \mid \overline{E \cap L_m(G_i)} = \overline{E} \cap L(G_i)\}.$$

The set of controllable, $L_m(G)$-closed and G_i-nonblocking sublanguages of K is defined as:

$$RCN_b(K,\mathcal{G}) = R(K,G) \cap C(K,G) \cap N_b(K,G_1) \cap ... \cap N_b(K,G_n).$$

The set $RCN_b(K,\mathcal{G})$ of sublanguages of K is nonempty and closed under union and hence has a supremal element denoted by $K^\star$ [23].

Theorem 2.2. [23] Consider RNSCP and let

$$K = L_m(G) \cap (\bigcap_{i=1}^{n}(K_i \cup (\Sigma^* - L_m(G_i))))$$

For every nonempty element of $RCN_b(K,\mathcal{G})$, K_s, there exists a supervisor V which solves RNSCP with $L_m(G,V) = K_s$ and vice versa. In particular, $K^* = \sup RCN_b(K,\mathcal{G})$ characterizes the maximally permissive solution of RNSCP. ■

The language K is the overall specification of RNSCP which includes all the strings which are not illegal in any of the possible models. Two problems that can be treated as special cases of robust control problem are described in the following.

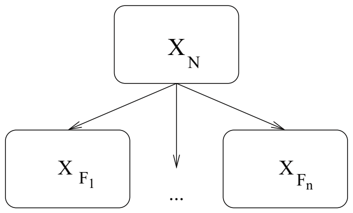
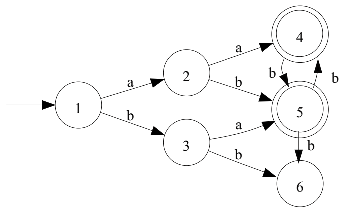


Figure 2.2: System with "*n*" permanent failure modes.

Example 2.1. Fault recovery [22], [23] in DES can be treated as a special case of Robust Nonblocking Supervisory Control Problem. The system is initially assumed to be in the normal mode where no failure has happened. Assume that there exist "*n*" permanent failure events in the system. Considering a single fault scenario, the state of the system can be in $X_n, X_{F_1}, ..., \text{or} X_{F_n}$ (Figure Figure 2.2). The normal and each faulty mode has its own legal specification. Thus the problem is an example of robust control with $\mathcal{G} = \{G_N, G_{NF_1}, ..., G_{NF_n}\}$. Here G_{NF_i} is the subautomaton containing states $X_n \cup X_{F_i}$. The goal is to design a supervisor such that each model under supervision meets its corresponding specification and be nonblocking. This problem can be formulated as a special case of RNSCP.

Example 2.2. Supervisory control with multiple sets of marked states [5] can be modeled as a RNSCP. Assume that we are given a plant $G = (X, \Sigma, x_0, \delta, X_{m_1}, ...X_{m_n})$ with multiple sets of marked states and specifications $K_i \subseteq L_m^i(G)$ where $L_m^i(G)$ represents marked language with only states in X_{m_i} marked. This problem can be treated as a robust problem with $\mathcal{G} = \{G_1, ..., G_n\}$ where $G_i = (X, \sigma, x_0, \delta, X_{m_i})$. Each model has its own specification $K_i \subseteq L_m(G_i) = L_m^i(G)$.

As an example, consider $\mathcal{G} = \{G_1, G_2\}$ as shown in Figure 2.3. Assume that all sequences are legal. We want to synthesize a robust supervisor V such that $\overline{L_m(G_i, V)} = L(G_i, V)$ for $i = 1, 2$, i.e., the plant under supervision is nonblocking with respect to $X_{m_1} = \{4, 5\}$ and X_{m_2}={5,6}


(a) $X_{m_1} = \{4, 5\}$

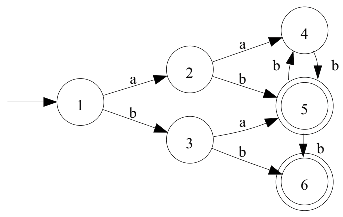
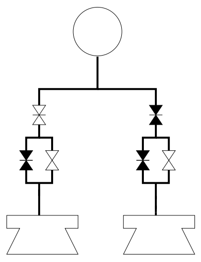
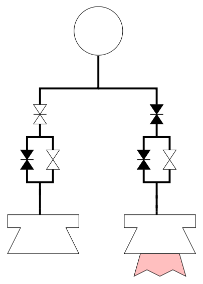

(b) $X_{m_2} = \{5, 6\}$

Figure 2.3: Supervisory control with multiple sets of marked states.

As a practical example of this kind, consider a propulsion system with two engines as depicted in Figure 2.4. The first set of marked states includes all the states where both engines are off. The second (resp. third) set of marked states includes all states where only engine one (resp. two) is fired. We want to switch from one configuration to another at anytime as required. Also, as a safety requirement, we must avoid the states where both engines are fired simultaneously. This problem can be regarded as control with multiple marked state sets and hence a special case of RNSCP.


(a)


(b)

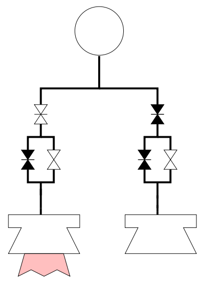

(c)

Figure 2.4: Application of supervisory control with multiple sets of marked states.

2.2.2 Online Supervisory Control

In the conventional (non-robust) supervisory control supervisor is entirely designed and implemented before the system begins operation. This approach is called *off-line supervisory control*. As an alternative to off-line approach [7] introduced an *online supervisory control* framework based on *Limited Lookahead Policies*. In this approach the control action is computed online while the plant (under supervision) is operational. The computation repeats each time the plant executes an event [7].

Limited Lookahead Policies

Limited Lookahead Policy supervisor first proposed by [7] computes the control action based on the knowledge about plant's behavior and its legal language up to $N-step$ beyond the currently executed trace. N is a fixed number which is determined before the plant starts operation. The choice of N, can for instance, be imposed by the amount of information available from future behavior of the system or by limitations of the controller such as processing power or available memory.

The prediction about future behavior of a plant is in the form of a tree expansion. A tree structure is useful and simple since it distinguishes between the states based on their event history and it makes no assumption about equivalence of the strings.

LLP supervisor: General Scenario

Let G be the automaton representation of a system under supervision. Also let $K \subseteq L_m(G)$ be the nonempty legal marked behavior of G. Σ_{uc} and Σ_c denote the set of uncontrollable and controllable events respectively. The objective is to design an online supervisor γ^N that restricts the behavior of the plant to its specification (i.e., $L_m(G, \gamma^N) \subseteq K$) while maintaining nonblocking property for the system under supervision nonblocking, i.e. $\overline{L_m(G, \gamma^N)} = L(G, \gamma^N)$. The procedure can be described in five steps Figure 2.5:

1. Based one the knowledge available about the plant, the supervisor *predicts* the possible behavior of the plant *N* step beyond the currently executed trace *s*. This is the function of block $f^N_{L(\mathcal{G})}$ to build the tree-expansion of the model.

2. The supervisor then determines which traces in the tree-expansion of G are illegal. String *st* is illegal if and only if $st \notin \overline{K}$. Removing illegal string is to be done by the block f^N_K in Figure 2.5

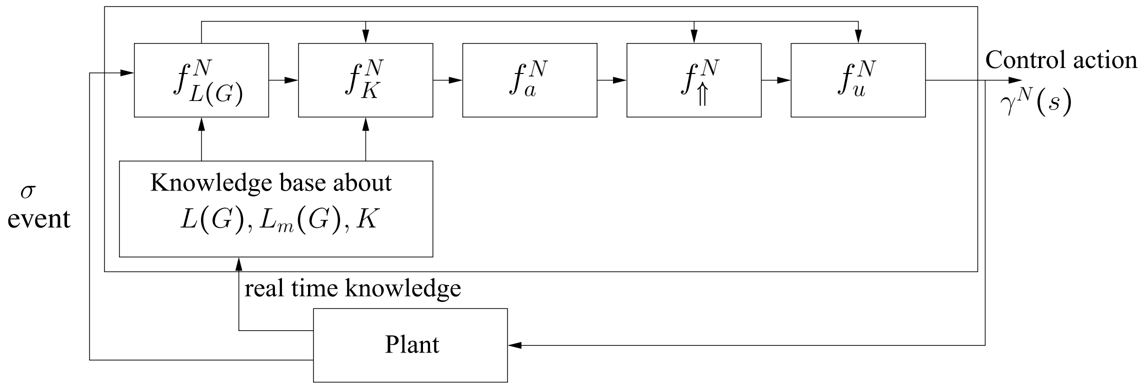


Figure 2.5: Block diagram of Limited Lookahead Supervisor

3. [7] introduced the notion of pending traces as traces of length N in tree-expansion which are not in the illegal region. A pending trace may continue into illegal region by executing uncontrollable events. Therefore, the block f^N_a modifies the tree expansion by adopting an optimistic or a conservative attitude toward the pending traces. conservative attitude treats all pending traces as illegal as they may lead to an illegal string with an uncontrollable sequence. On the contrary, the optimistic attitude treats all pending traces as legal and marked.

4. Afterward, the block $f^N_\Uparrow$ computes the supremal controllable part of the output of block f^N_a with respect to the tree-expansion $G/s|_N$.

5. Finally, the block f^N_u finds the enabled events for the currently executed string s by restricting the output of block $f^N_\Uparrow$ to its first level and then adding all the enabled uncontrollable events to it.

One can see that after generating the tree expansion of the plant, the rest is very similar to the conventional supervisory control. There are three main issues in this framework. First, what if the supremal controllable sublanguage of $K/s|_N$ becomes empty in some strings and for some length of lookahead windows. The second issue is to investigate about the effect of choosing an attitude on effectiveness of the proposed methods. Finally, the third concern is to find conditions under which the proposed method results in a maximally permissive supervision. The following results are from [7].

Definition 2.15. If $f^N(s) = \varnothing$ for some $s \in L(G, \gamma^N)$, then we say that there is a *Run-Time Error* (RTE) at s. If $s = \epsilon$ then then it is called a *Starting Error* (SE).

Definition 2.16. An LLP supervisor with control policy γ^N is called *maximally permissive* if and only if $L(G, \gamma^N) = \sup C(K, G)$.

The following lemmas address the occurrence of RTE and SE in LLP supervision.

Lemma 2.3. If there is no SE in $L(G, \gamma^N_{cons})$, then $\sup C(K, G) \neq \varnothing$. ■

The above lemma states that conservative LLP supervisor is a pessimistic supervisor. In other words, if $\sup C(K, G) = \varnothing$, then SE would definitely happens with conservative attitude. This is not necessarily the case for the optimistic LLP supervisor.

Proposition 2.1. $\overline{\sup C(K, G)} \subseteq L(G, \gamma^{N+1}_{opt}) \subseteq L(G, \gamma^N_{opt})$. ■

The above proposition says that the optimistic LLP supervisor is more permissive than the optimal supervisor. It becomes less permissive as it receives more information about the plant if longer lookahead window is used.

Proposition 2.2. In general $L(G, \gamma^N_{cons}) \subseteq L(G, \gamma^{N+1}_{cons}) \subseteq \overline{\sup C(K, G)}$. ■

In other words, the conservative LLP supervisor is less permissive compared with the optimal solution. It approaches the optimal solution as we increase the length of lookahead window.

The next issue in LLP supervision is to determine if there exist conditions for maximal permissiveness of the LLP supervisor. The conditions are presented in the following. These conditions are considered for both prefixed-closed and non-closed specifications.

Prefix-closed specification K = $\overline{\mathbf{K}}$:

In this case we do not care about the marking. The following definition formalizes the length of longest uncontrollable string in K, the parameter which plays an important role in determining the length of sufficient window size.

Definition 2.17. The length of the longest uncontrollable subtrace of strings in K is denoted by N_u and is defined as:

$$N_u(K) := \begin{cases} \max\{|s| : s \in \Sigma_u^* \wedge (\exists v, u \in \Sigma^* \mid vsu \in K)\} & \text{if exists} \\ \text{undefined otherwise} & \blacksquare \end{cases}$$

Theorem 2.3. If $K = \overline{K}$ and $N \geq N_u(K) + 2$, then γ_{opt}^N is maximally permissive. ■

Theorem 2.4. If $K = \overline{K}$, no SE happens at $L(G, \gamma_{cons}^N)$ and $N \geq N_u(K) + 2$, then γ_{cons}^N is maximally permissive. ■

General Case K ⊆ $\overline{\mathbf{K}}$:

In this more general case the marking is important and the LLP supervisor needs to ensure that the system under supervision is nonblocking. Thus, we expect the length of sufficient lookahead window to be even longer than the prefix-closed case. We need the following definitions before presenting the theorems on the length of sufficient lookahead window for optimality.

Definition 2.18. The set of all legal marked traces in which the active event set is controllable is formalized as:

$$K_{mc} = \{s \in K : (\forall \sigma \in \Sigma_u) s\sigma \notin L(G)\}$$

■

The set of all traces that cross the boundary between $\overline{K}$ and $L(G) - \overline{K}$ by executing a sequence of uncontrollable events is formalized as the language

$$K_{fc} = \big((L(G) - \overline{K})/\Sigma_u\big) \cap \overline{K}$$

Definition 2.19. N_{mcfc} is the length of the longest subtrace in K which leads the plant to an illegal state from the initial state or a marked state without generating any marked legal controllable or illegal strings.

$$N_{mcfc} := \begin{cases} \max\{|t| : \exists s \in (K_{mc} \cup \{\epsilon\}) st \in K_{fc} \wedge (\forall \epsilon < v < t) sv \notin (K_{fc} \cup K_{mc})\} & \text{if exists} \\ \text{undefined} & \text{otherwise} \quad \blacksquare \end{cases}$$

Definition 2.20. K_{mcmc} is the length of longest subtrace in K which leads the plant from the initial state or marked controllable strings to their neighbor legal marked controllable strings.

$$N_{mcmc} := \begin{cases} \max\{|t| : \exists s \in (K_{mc} \cup \{\epsilon\}) st \in K_{mc} \wedge (\forall \epsilon < v < t) sv \notin K_{mc}\} & \text{if exists} \\ \text{undefined} & \text{otherwise} \quad \blacksquare \end{cases}$$

Theorem 2.5. [7] Assume $\sup C(K,G) \neq \varnothing$ and $N \geq N_{mcfc}(K) + 1$ then γ_{opt}^N is a valid LLP supervisor.

Theorem 2.6. [7] Assume no SE happens in $L(G, \gamma_{cons}^N)$ and $N \geq N_{mcmc}(K) + 1$. Then γ_{cons}^N is a maximally permissive LLP supervisor and $L(G, \gamma_{cons}^N) = \overline{\sup C(K,G)}$. $\blacksquare$

Theorem 2.6 formalizes the most useful case in the LLP supervision framework since:

- It deals with the general case $K \neq \overline{K}$.
- Conservative supervisor is always an admissible supervisor while optimistic LLP supervisor may lead the system to blocking or even illegal region since $\overline{\sup C(K,G)} \subseteq$

$L(G, \gamma_{opt}^N)$

As a consequence, in this thesis we focus our efforts to generalize Theorem 2.6 to the the case of robust supervisory control.

Other Preliminaries

The following lemmas are also needed to prove the results of this thesis.

Lemma 2.4. [6] For $L, K \subseteq \Sigma^*$ we have $\overline{L} \cup L\overline{K} = \overline{LK}$. ■

Lemma 2.5. *For three sets A, B, C*

$$(A - B) \cap C = (A \cap C) - (B \cap C)$$

■

2.3 Conclusion

This chapter provides the background that is needed in the developments of this thesis. A brief background in discrete event models, robust supervisory control, and online supervisory control has been covered.

Chapter 3

Limited Lookahead Policy for Robust Nonblocking Supervisory Control: A Linguistic Approach

As with many other areas of control theory, the notion of online control has been introduced to supervisory control of discrete event systems by the lookahead policies. In this setting, it is assumed that the behavior of the plant under supervision is known a few steps beyond the currently executed trace. The most one word setting of online supervisory control in term of implementation and putting into practice is the Limited Lookahead Policies (LLP), where all possible behaviors of the plant and its legal behavior are known up to N steps beyond the currently executed trace (where N is fixed number). In Chapter 2, we discussed the LLP for supervisory control of DES introduced in [7], where it is assumed that the exact model of plant is known and there is no uncertainty associated with the modeling. In this chapter, we extend the results of [7] to the case of robust control where the plant model is assumed to belong to a given finite set of DES models.

3.1 Problem Formulation

This section is intended to formulate the Robust Nonblocking Supervisory Control Problem based on Limited Lookahead policy (RNSCP-LL). The problem studied here is an extension of the problem studied in [7] to the case of robust control. Consider a DES whose true model is one of the $G_1, ..., G_n$. Let Σ_i and $\Sigma_{c,i}$ denote the event set and the set of controllable events of G_i. The event sets are not necessarily the same; however it is assumed that all plants agree on the controllability status of the events. This means that if $\sigma \in \Sigma_i \cap \Sigma_j$, then σ is controllable in G_i if and only if σ is controllable in G_j. The set of all events is denoted by $\Sigma = \sum_{i=1}^{n} \Sigma_i$, and $\Sigma_c = \sum_{i=1}^{n} \Sigma_{c_i} \subseteq \Sigma$ is the set of all controllable events.

For each plant it is assumed that a nonempty language $K_i \subseteq L_m(G_i)$ describes the legal behavior for marked traces. The problem is to design an online supervisor based on the available knowledge about the behavior of the plants a fixed "N" steps beyond the currently executed trace s. The supervisor is supposed to generate the control law based on the available information in the tree expansions of G_is. The resulting Robust Limited Lookahead (RLL) supervisor must ensure the legality and nonblocking properties of the plants under supervision:

(1) $L(G_i, \gamma^N) \subseteq K_i, \qquad i = 1, ..., n$

(2) $\overline{L_m(G_i, \gamma^N)} = L(G_i, \gamma^N), \qquad i = 1, ..., n$

where $L(G_i, \gamma^N)$ and $L_m(G_i, \gamma^N)$ are the closed and marked behaviors of G_i under supervision of the RLL supervisor γ^N.

3.2 Robust Limited Lookahead Supervision

3.2.1 General Scenario

A new online approach for Robust Nonblocking Supervisory Control (RNSC) problem based on LLP is introduced in this section. It is an extension of LLP in [7] to the case of robust control. Figure 3.1 shows the block diagram of the RLL supervisor. Suppose that the process has been executed trace s so far. The online control functions in Figure 3.1 are described in the following:

1. Suppose that the plant under supervision has executed string s so far. Based on the knowledge about the behavior of G_i, $i = 1, ..., n$, the supervisor *determines* whether plant G_i is compatible with the currently executed trace s. This is the function of block $f^s_{\mathcal{G}}$; without loss of generality, suppose that the first n' plants are compatible with the trace s.

2. On the basis of knowledge available about the plants, the supervisor *predicts* the possible behavior of all compatible plant models N step beyond the currently executed trace s. The function of block $f^N_{L(\mathcal{G})}$ is to build tree-expansions of the models.

3. The supervisor then determines which traces in the tree expansion of G_i are illegal. String s is illegal if and only if $s \notin \overline{K}$. We define K according to:

$$K = L_m(G) \cap (\bigcap_{i=1}^{n}(K_i \cup (\Sigma^* - L_m(G_i)))) \tag{3.1}$$

   The plant G is defined to be the union model of the plants: $L(G) = \bigcup\limits_{i=1}^{n} L(G_i)$ and $L_m(G) = \bigcup\limits_{i=1}^{n} L_m(G_i)$. Removing illegal strings is to be done by the block f^N_K in Figure 3.1.

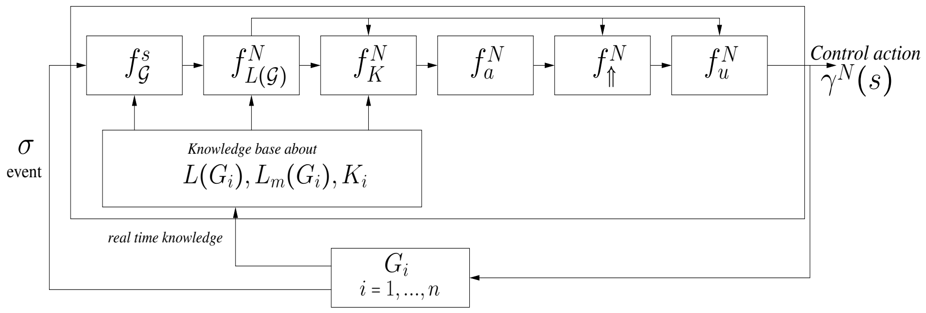


Figure 3.1: Block diagram of Robust Limited Lookahead Supervisor

4. [7] introduced the notion of pending traces as traces of length N in tree-expansion which are not in the illegal region. A pending trace may continue into illegal region by executing uncontrollable events. Therefore the block f_a^N modifies the tree expansion by adopting a conservative attitude toward the pending traces, which means that all pending traces as treated as illegal [7].

5. Afterward, the block $f_{\Uparrow}^N$ computes the supremal $\mathrm{L_m(G)}$-closed, controllable, and G_i/s-nonblocking part of the output of block f_a^N with respect to the tree expansions of G_i/s.

6. Finally, the block f_u^N finds the enabled events for the currently executed string s, by determining the element in every sequence in the output of the block $f_{\Uparrow}^N$.

3.2.2 Formalizing RLL Supervisor

The previously mentioned six steps are formally expressed in this section.

1. $f_{\mathcal{G}}^s$ determines if $s \in L(G_i)$ (G_i is compatible with s).

2. $f_{L(\mathcal{G})}^N \circ f_{\mathcal{G}}^s = \big(L(G/s|_N), L_m(G/s|_N)\big)$ where $G/s|_N$ is the union of tree expansion of G_i, $i = 1, ..., n'$ up to N steps after the execution of trace s. The block also passes $G/s|_N$ and $G_i/s|_N$ to the other blocks for the computation of supremal elements.

3. $f_K^N \circ f_{L(\mathcal{G})}^N \circ f_{\mathcal{G}}^s = (\overline{K}/s|_N, K/s|_N)$ where K is the legal behavior of robust problem which is described by Equation 3.1.

4. $f_a^N \circ f_K^N \circ f_{L(\mathcal{G})}^N \circ f_{\mathcal{G}}^s = K/s|_N - (\overline{K}/s|_N - \overline{K}/s|_{N-1}) = K/s|_{N-1}$ which means that we treat all the pending traces as illegal.

5. $f^N(s) \coloneqq f_{\Uparrow}^N \circ f_a^N \circ f_K^N \circ f_{L(\mathcal{G})}^N \circ f_{\mathcal{G}}^s = \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N)$

   computes the supremal $L_m(G/s|_N)$-closed, controllable, and $G_i/s|_N$-nonblocking $(i = 1, ..., n')$ part of $K/s|_{N-1}$[1].

6. Finally, the control action $\gamma^N(s)$ is determined through f_u^N block $\gamma^N(s) \coloneqq f_u^N \circ f^N(s) \coloneqq \overline{f^N(s)} \cap \Sigma$.

So the closed and marked languages of the plants under supervision can be defined recursively as follows:

- $\varepsilon \in L(G_i, \gamma^N(s))$;
- $\forall s \in \Sigma^*, \sigma \in \Sigma : s \in L(G_i, \gamma^N(s)), s\sigma \in L(G_i), \sigma \in \gamma^N(s) \Rightarrow s\sigma \in L(G_i, \gamma^N(s))$.

3.3 Properties of RLL Supervisor

In this section we prove some general properties of the RLL supervisor for a given lookahead window size N. In the following section, we will discuss how N can be chosen.

The following lemma and proposition are required for the proof of the monotonicity property (Theorem 3.1).

Lemma 3.1. Let $s \in \Sigma^*$. Then:

$$\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) = \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_{N+1})$$

[1]For $n' < i \leq n$, G_i/s is an empty automaton and G_i/s-nonblocking condition holds trivially.

Proof: Let

$$LHS := \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N)$$
$$RHS := \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_{N+1})$$
$$\Delta L_i = L(G_i/s|_{N+1}) - L(G_i/s|_N)$$
$$\Delta L_{m,i} = \Delta L_i \cap L_m(G_i/s|_{N+1})$$
$$\Delta L = L(G/s|_{N+1}) - L(G/s|_N)$$
$$\Delta L_m = \Delta L \cap L_m(G/s|_{N+1})$$

Note that $\forall s \in \Delta L_i \rightarrow |s| = N + 1$. First we prove that $LHS \subseteq RHS$. For this we prove that LHS is relative closed and controllable with respect to $G/s|_{N+1}$ and nonblocking with respect to $G_i/s|_{N+1}$, $i = 1, ..., n'$.

i) G-nonblocking: Since LHS is $G_i/s|_N$-nonblocking we have:

$$\overline{LHS} \cap L(G_i/s|_N) = \overline{LHS \cap L_m(G_i/s|_N)}$$

Also $\overline{LHS} \cap \Delta L_i = LHS \cap \Delta L_{m,i} = \varnothing$. Thus:

$$\begin{aligned}
\overline{LHS \cap L_m(G_i/s|_{N+1})} &= \overline{LHS \cap (L_m(G_i/s|_N) \cup \Delta L_{m,i})} \\
&= \overline{LHS \cap L_m(G_i/s|_N)} \cup \overline{LHS \cap \Delta L_{m,i}} \\
&= \left(\overline{LHS} \cap L(G_i/s|_N)\right) \cup \left(\overline{LHS} \cap \Delta L_i\right) \\
&= \overline{LHS} \cap L(G_i/s|_{N+1})
\end{aligned}$$

ii) Controllability: By assumption, $\overline{LHS}\Sigma_u \cap L(G/s|_N) \subseteq \overline{LHS}$. Also, we know $\forall t \in$

LHS the $|t| \leqslant N-1$. So for all $t \in LHS$ and $\sigma \in \Sigma$ $|t\sigma| \leq N$ and thus $\overline{LHS}\Sigma_u \cap \Delta L = \varnothing$.

$$\begin{aligned}
\overline{LHS}\Sigma_u \cap L(G/s|_{N+1}) &= \overline{LHS}\Sigma_u \cap \big(L(G/s|_N) \cup \Delta L\big) \\
&= \big(\overline{LHS}\Sigma_u \cap L(G/s|_N)\big) \cup \big(\overline{LHS}\Sigma_u \cap \Delta L\big) \\
&\subseteq \overline{LHS}
\end{aligned}$$

iii) $L_m(G)$-Closure: We know that $\overline{LHS} \cap L_m(G/s|_N) = LHS$ and $\overline{LHS} \cap \Delta L_m = \varnothing$. Thus we can easily conclude that $\overline{LHS} \cap L_m(G/s|_{N+1}) = \overline{LHS} \cap \big(L_m(G/s|_N) \cup \Delta L_m\big) = LHS$.

Now we prove the second part $RHS \subseteq LHS$ following a similar procedure.

i) G-nonblocking: Since RHS is $G_i/s|_{N+1}$-nonblocking, we have:

$$\overline{RHS} \cap L(G_i/s|_{N+1}) = \overline{RHS \cap L_m(G_i/s|_{N+1})}$$

Also $\overline{RHS} \cap \Delta L_i = RHS \cap \Delta L_{m,i} = \varnothing$. Thus:

$$\begin{aligned}
\overline{RHS \cap L_m(G_i/s|_N)} &= \overline{RHS \cap (L_m(G_i/s|_N) \cup \Delta L_{m,i})} \\
&= \overline{RHS \cap L_m(G_i/s|_{N+1})} \\
&= \overline{RHS} \cap L(G_i/s|_{N+1}) \\
&= \overline{RHS} \cap \big(L(G_i/s|_N) \cup \Delta L_i\big) \\
&= \overline{RHS} \cap L(G_i/s|_N)
\end{aligned}$$

ii) Controllability: By assumption, $\overline{RHS}\Sigma_u \cap L(G/s|_{N+1}) \subseteq \overline{RHS}$. Also, we know $\forall t \in$

RHS, $|t| \leqslant N-1$. So for all $t \in RHS$ and $\sigma \in \Sigma$ $|t\sigma| \leq N$ and thus $\overline{RHS}\Sigma_u \cap \Delta L = \varnothing$.

$$\begin{aligned}
\overline{RHS}\Sigma_u \cap L(G/s|_N) &= \overline{RHS}\Sigma_u \cap \big(L(G/s|_N) \cup \Delta L\big) \\
&= \overline{RHS}\Sigma_u \cap L(G/s|_{N+1}) \\
&\subseteq \overline{RHS}
\end{aligned}$$

iii) $L_m(G)$-Closure: By assumption, $\overline{RHS} \cap L_m(G/s|_{N+1}) = RHS$ and $\overline{RHS} \cap \Delta L_m = \varnothing$. Thus we can easily conclude that $\overline{RHS} \cap L_m(G/s|_N) = \overline{RHS} \cap \big(L_m(G/s|_N) \cup \Delta L_m\big) = \overline{RHS} \cap L_m(G/s|_{N+1}) = RHS$.

■

Similar to Lemma 3.1 we can derive the following proposition.

Proposition 3.1. Let $s \in \Sigma^*$. Then:

$$\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) = \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s)$$

Proof: The proof is very similar to Lemma 3.1 and it is removed for brevity. ■

The following theorem establishes the expected results that a larger lookahead window results in a more permissive supervision. The theorem generalizes the result which was first presented as Theorem 4.5 in [7] and [27].

Theorem 3.1. (Monotonicity) $L(G, \gamma^N) \subseteq L(G, \gamma^{N+1})$

Proof: It is enough to show that $(\forall s \in \Sigma^*)$ $f^N(s) \subseteq f^{N+1}(s)$. By definition:

$$\begin{aligned}
f^N(s) &:= \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) \\
&= \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_{N+1}) \quad \text{(by Lemma 3.1)}
\end{aligned}$$

But we know that

$$\begin{aligned}(K/s|_{N-1}) \subseteq (K/s|_N) \quad &\Rightarrow \quad supRCN_b(K/s|_{N-1}, \mathcal{G}/s|_{N+1}) \subseteq supRCN_b(K/s|_N, \mathcal{G}/s|_{N+1}) \\ &\Rightarrow \quad supRCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) \subseteq supRCN_b(K/s|_N, \mathcal{G}/s|_{N+1}) \\ &\Rightarrow \quad f^N(s) \subseteq f^{N+1}(s)\end{aligned}$$

■

Remark 3.1. The monotonicity property of union model under supervision can be easily extended to each of the plants with exactly the same procedure and hence $L(G_i, \gamma^N) \subseteq L(G_i, \gamma^{N+1})$, $i = 1, ..., n$.

The following lemma generalizes Theorem A.1 of [7]. It will be needed later to prove Theorem 3.2.

Lemma 3.2.

$$1)\, (\forall s \in \Sigma^*)\ \sup RCN_b(K, \mathcal{G})/s \subseteq \sup RCN_b(K/s, \mathcal{G}/s)$$
$$2)\, (\forall s \in \overline{\sup RCN_b(K, \mathcal{G})})\ \ \sup RCN_b(K, \mathcal{G})/s = \sup RCN_b(K/s, \mathcal{G}/s)$$

Proof: For notational simplicity let

$$H := \sup RCN_b(K, \mathcal{G})$$

1) We need to show that i) $H/s \subseteq K/s$, ii) H/s is nonblocking with respect to G_i/s, $i = 1, ..., n$, iii) H/s is relative closed with respect to G/s and iv) H/s is controllable with respect to G/s. Clearly, $H/s \subseteq K/s$. We first show that H/s is nonblocking with respect to

G_i/s.

$$\begin{aligned}
\overline{H \cap L_m(G_i)} = \overline{H} \cap L(G_i) \quad &\Rightarrow \quad \big(\overline{H \cap L_m(G_i)}\big)/s = \big(\overline{H} \cap L(G_i)\big)/s \\
&\Rightarrow \quad \overline{\big(H \cap L_m(G_i)\big)/s} = \big(\overline{H} \cap L(G_i)\big)/s \text{ (by Lemma 2.1)} \\
&\Rightarrow \quad \overline{H/s \cap L_m(G_i/s)} = \overline{H}/s \cap L(G_i/s) \\
&\Rightarrow \quad \overline{H/s \cap L_m(G_i/s)} = \overline{H/s} \cap L(G_i/s)
\end{aligned}$$

Next we show that H/s is relative closed with respect to G/s.

$$\begin{aligned}
\overline{H/s} \cap L_m(G/s) \;&=\; \overline{H}/s \cap L_m(G)/s \;\text{ (part 2, Lemma 2.1)} \\
&=\; (\overline{H} \cap L_m(G))/s \;\text{ (part 1, Lemma 2.1)} \\
&=\; H/s \;\text{ (since H is } L_m(G)\text{-closed)}
\end{aligned}$$

Finally, we show that H/s is controllable with respect to G/s.

$$\begin{aligned}
\overline{H/s}\Sigma_u \cap L(G/s) \;&=\; \overline{H}/s\Sigma_u \cap L(G/s) \;\text{ (part 2, Lemma 2.1)} \\
&\subseteq\; \overline{H}\Sigma_u/s \cap L(G/s) \;\text{ (part 3, Lemma 2.1)} \\
&=\; \big(\overline{H}\Sigma_u \cap L(G)\big)/s \;\text{ (part 1, Lemma 2.1)} \\
&\subseteq\; \overline{H}/s \;\text{ (since H is controllable)} \\
&=\; \overline{H/s}
\end{aligned}$$

2) Based on the result of part 1, we only need to prove the reverse containment:

$$\sup RCN_b(K/s, \mathcal{G}/s) \subseteq \sup RCN_b(K, \mathcal{G})/s$$

For simplicity let $H' := \sup RCN_b(K/s, \mathcal{G}/s)$. So, we need to show $H' \subseteq H/s$ or equivalently $sH' \subseteq H$. Since H is supremal relative-closed, controllable with respect to G , and

G_i-nonblocking sublanguage of K, it is enough to show that $H \cup sH'$ is also relative-closed, and controllable with respect to G and G_i-nonblocking sublanguage of K. This means that we need to show that $H \cup sH' \subseteq \mathcal{H}$. First we show that $H \cup sH'$ is G_i-nonblocking. Note that H' is G_i/s-nonblocking.

$$\begin{aligned}
\overline{(H \cup sH') \cap L_m(G_i)} &= \overline{H \cap L_m(G_i)} \cup \overline{(sH' \cap L_m(G_i))} \\
&= \overline{H \cap L_m(G_i)} \cup \overline{s(H' \cap L_m(G_i/s))} \\
&= \overline{H \cap L_m(G_i)} \cup \left(\overline{s} \cup s\,\overline{\left(H' \cap L_m(G_i/s)\right)}\right) \text{ (by Lemma 2.4)} \\
&= \left(\overline{H} \cap L(G_i)\right) \cup \overline{s} \cup s\left(\overline{H'} \cap L(G_i/s)\right) \\
&= \left(\overline{H} \cap L(G_i)\right) \cup \left((\overline{s} \cup s\overline{H'}) \cap \left(\overline{s} \cup L(G_i)\right)\right) \\
&= \left(\overline{H} \cap L(G_i)\right) \cup \left(\overline{sH'} \cap \left(\overline{s} \cup L(G_i)\right)\right) \\
&\supseteq \left(\overline{H} \cap L(G_i)\right) \cup \left(\overline{sH'} \cap L(G_i)\right) \\
&= \overline{(H \cup sH')} \cap L(G_i)
\end{aligned}$$

Next we show that $H \cup sH'$ is controllable with respect to G. Since $s \in \overline{H}$ we have $\overline{s}\Sigma_u \cap L(G) \subseteq \overline{H}$. Also, H' is controllable with respect to G/s or equivalently $s\overline{H'}\Sigma_u \cap L(G) \subseteq s\overline{H'}$. Thus:

$$\begin{aligned}
(\overline{s} \cup s\overline{H'})\Sigma_u \cap L(G) &\subseteq (\overline{H} \cup s\overline{H'}) \\
\overline{sH'}\Sigma_u \cap L(G) &\subseteq \overline{H \cup sH'} \text{ (by Lemma 2.4)}
\end{aligned}$$

Since H is controllable with respect to G itself, we can easily see that $\overline{H \cup sH'}\Sigma_u \cap L(G) \subseteq \overline{H \cup sH'}$. Next we show that $H \cup sH'$ is relative closed with respect to G. $\overline{s} \cap L_m(G) \subseteq H$ since $s \in H$ and H is relative-closed with respect to G. Also by assumption we have $\overline{H'} \cap L_m(G)/s \subseteq H'$ or equivalently, $s\overline{H'} \cap L_m(G) \subseteq sH'$. Considering the union of two equations we have:

$$\begin{aligned}\left(\overline{s} \cup s\overline{H'}\right) \cap L_m(G) &\subseteq H \cup sH' \\ \overline{sH'} \cap L_m(G) &\subseteq H \cup sH' \text{ (by Lemma 2.4)}\end{aligned}$$

By assumption H is relative-closed with respect to G and consequently, $\overline{H} \cap L_m(G) = H$. Therefore $(\overline{H} \cup \overline{sH'}) \cap L_m(G) \subseteq H \cup sH'$ and finally:

$$\overline{H \cup sH'} \cap L_m(G) \subseteq H \cup sH'$$

Since the reverse containment always holds, we have proved that $H \cup sH'$ is relative-closed with respect to G. This completes the proof.

■

The following lemmas are required to prove Theorem 3.2 and Theorem 3.3. Theorem 3.2 explains the admissibility of the RLL supervisor.

Lemma 3.3. Let $s \in \Sigma^*$. Then:

$$\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) \subseteq \sup RCN_b(K/s, \mathcal{G}/s)$$

Proof:

$$\begin{aligned}\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) &= \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s) \text{ (by Proposition 3.1)} \\ &\subseteq \sup RCN_b(K/s, \mathcal{G}/s) \text{ (since } K/s|_{N-1} \subseteq K/s)\end{aligned}$$

■

Lemma 3.4. Let $s \in \Sigma^*$. Then:

$$\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N)/\sigma \subseteq \sup RCN_b(K/s\sigma|_{N-1}, \mathcal{G}/s\sigma|_N)$$

Proof:

$$\begin{aligned}
\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N)/\sigma &= \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_{N+1})/\sigma \text{ (by Lemma 3.1)} \\
&\subseteq \sup RCN_b(K/s|_N, \mathcal{G}/s|_{N+1})/\sigma \text{ (since } K/s|_{N-1} \subseteq K/s|_N) \\
&\subseteq \sup RCN_b(K/s|_N/\sigma, \mathcal{G}/s|_{N+1}/\sigma) \text{ (by Lemma 3.2, part 1)} \\
&= \sup RCN_b(K/s\sigma|_{N-1}, \mathcal{G}/s\sigma|_N)
\end{aligned}$$

■

In the following we define the notions of *starting error* and *run-time error* similar to [7].

Definition 3.1. We say that there is a *run-time error* (RTE) in $L(G, \gamma^N)$ at string $s \in L(G, \gamma^N)$ if $f^N(s) = \varnothing$. An RTE is said to be a *Starting Error* (SE) if $s = \epsilon$. We say there is no RTE in $L(G, \gamma^N)$ if no RTE happens for all $s \in L(G, \gamma^N)$. ■

The following theorem states that in the absence of SE, the RLL supervisor is always nonblocking and admissible.

Theorem 3.2. If there is no SE in $L(G, \gamma^N)$, then $L(G, \gamma^N) \subseteq \overline{\sup RCN_b(K, \mathcal{G})}$.

Proof: We use induction on the length of s to prove this. For $|s| = 0$, $s = \epsilon$. By definition $\epsilon \in L(G, \gamma^N)$. Also, since there is no SE, $\sup RCN_b(K, \mathcal{G}) \neq \varnothing$; this implies that $\epsilon \in \overline{\sup RCN_b(K, \mathcal{G})}$. So we have the base step by assumption. For the induction step, let $s = s'\sigma$ where $\sigma \in \Sigma$. Since $L(G, \gamma^N)$ is prefix closed we have $s' \in L(G, \gamma^N)$. On the other hand, we have $s' \in \overline{\sup RCN_b(K, \mathcal{G})}$ by the induction hypothesis. Also, $s = s'\sigma \in L(G, \gamma^N)$. By definition we have $\sigma \in \gamma^N(s')$, which means that $\sigma \in \overline{f^N(s)} \cap \Sigma$. So, we need to show that if $\sigma \in \overline{f^N(s)} \cap \Sigma$, then $\sigma \in \overline{\sup RCN_b(K,\mathcal{G})}/\, s'$. Consider the case of $\sigma \in \overline{f^N(s')} \cap \Sigma$.

$$\begin{aligned}
\overline{f^N(s')} \cap \Sigma &= \overline{\sup RCN_b(K/s'|_{N-1}, \mathcal{G}/s'|N)} \cap \Sigma \\
&\subseteq \overline{\sup RCN_b(K/s', \mathcal{G}/s')} \cap \Sigma \quad \text{(by Lemma 3.3)} \\
&= \overline{\sup RCN_b(K,\mathcal{G})/s'} \cap \Sigma \quad \text{(by Lemma 3.2, part 2)} \\
&= \overline{\sup RCN_b(K,\mathcal{G})}/s' \cap \Sigma \quad \text{(by Lemma 2.1, part 2)}
\end{aligned}$$

So σ also belongs to $\overline{\sup RCN_b(K,\mathcal{G})}/\, s'$. This completes the proof. ■

The next theorem explains the relation between SE and RTE for a system under supervision of an RLL supervisor.

Theorem 3.3. If there is no SE in $L(G,\gamma^N)$, then there is no RTE in $L(G,\gamma^N)$.

Proof: In order to prove this theorem we use induction on the length of traces. Since there is no SE in $L(G,\gamma^N)$, there is no RTE at $s = \epsilon$. In other words the base step holds by assumption for $|s| = 0$. For the induction step, suppose that $s = s'\sigma$ with $\sigma \in \Sigma$, where we know that there is no RTE in s' based on the induction hypothesis or equivalently:

$$\sup RCN_b(K/s'|_{N-1}, \mathcal{G}/s'|_N) \neq \varnothing$$

This implies that:

$$\epsilon \in \overline{\sup RCN_b(K/s'|_{N-1}, \mathcal{G}/s'|_N)}$$

Since $\sigma \in L(G,\gamma^N)/s'$, we have:

$$\sigma \in \gamma^N(s') = \overline{f^N(s')} \cap \Sigma$$

We can conclude that:

$$\begin{aligned}\sigma \in \overline{f^N(s')} &= \overline{\sup RCN_b(K/s'|_{N-1}, \mathcal{G}/s'|_N)} \\ &\Rightarrow \sup RCN_b(K/s'|_{N-1}, \mathcal{G}/s'|_N)/\sigma \neq \emptyset\end{aligned}$$

So from Lemma 3.4 we can say that:

$$\sup RCN_b(K/s'\sigma|_{N-1}, \mathcal{G}/s'\sigma|_N) \neq \emptyset$$

which means that there is no RTE at $s = s'\sigma$. This completes the proof. ■

3.4 Maximally Permissive RLL Supervisor

In this section we provide a set of conditions under which N can be chosen to ensure that RLL is maximally permissive.

Definition 3.2. An RLL supervisor with control policy γ^N is called *maximally permissive* if $L(G, \gamma^N) = \overline{\sup RCN_b(K, \mathcal{G})}$. ■

For an RLL supervisor to be maximally permissive the size of lookahead window should be large enough to cover the minimum number of safe traces in the lookahead window for every prefix of legal marked behavior. By a safe trace we mean a trace which is legal and G_i-nonblocking with respect to each of the plants. Also transitions of a safe trace should be controllable. We call such traces to be *nonblocking frontier traces* as a generalization of *frontier traces* in [7].

Definition 3.3. Given a set of plants G_i, $i = 1, ..., n$, the set of *nonblocking traces* is defined as:

$$K_{nb} := \{t \in K \mid \overline{\bar{t} \cap L_m(G_i)} = \bar{t} \cap L(G_i) \wedge \forall \sigma \in \Sigma_u : [t\sigma \notin L(G)]\}$$

Also, we define the set of *nonblocking frontier traces* after the trace $s \in \overline{K}$ as:

$$\begin{aligned}(K/s)_{nf} &:= K_{nb}/s \\ &= \{t \in K/s \mid \overline{\overline{st} \cap L_m(G_i)} = \overline{st} \cap L(G_i) \wedge \forall \sigma \in \Sigma_u : [t\sigma \notin L(G)/s]\}\end{aligned}$$

Further the set of *neighboring nonblocking controllable frontier traces* is defined as follows:

$$(K/s)_{nnf} := \{t \in (K/s)_{nf} \mid t \neq \epsilon \wedge [\forall t' < t : t' \notin (K/s)_{nf}]\}$$

where $t' < t$ means $t \leq t'$ and $t \neq t'$. ■

As will be discussed later, the length of the longest *neighboring nonblocking controllable frontier trace* plays a role in the length of sufficient lookahead window. N_{nnf} denotes the length of longest *neighboring nonblocking frontier trace*, and is defined as follows:

$$N_{nnf} := \begin{cases} \max_{s \in \overline{K}} \left\{ \max_{t \in (K/s)_{nnf}} |t| \right\} & \text{if it exists} \quad (3.2) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that N_{nnf} is considered undefined if at least one of the sets in (3.2) is unbounded or if all the sets become empty. Also, in (3.2) max of empty set is taken to be zero.

The following lemmas are used to prove the main result.

Lemma 3.5. Let $H \in RCN_b(K/s, \mathcal{G}/s)$ and $s \in \overline{\sup RCN_b(K, \mathcal{G})}$. Suppose for $t \in H$:

1. $\overline{\overline{t} \cap L_m(G_i/s)} = \overline{t} \cap L(G_i/s)$

2. $\forall \sigma \in \Sigma_u : [t\sigma \notin L(G/s)]$

Then $H - \big(t(H/t) - t\big) \in RCN_b(K/s, \mathcal{G}/s)$.

Proof: $H' = t(H/t) - t$ and $H'' = t(\overline{H/t}) - t$. First we show that $\overline{H - H'} = \overline{H} - H''$.

$$\begin{aligned}
\overline{H} - H'' &= [\overline{H - H'} \cup \overline{H'}] - (t\overline{H/t} - t) \\
&= [\overline{H - H'} - (t\overline{H/t} - t)] \cup [\overline{H'} - (t\overline{H/t} - t)] \\
&= \overline{H - H'} \cup [\overline{H'} - (t\overline{H/t} - t)] \\
&\qquad (\overline{H - H'} \text{ and } (t\overline{H/t} - t) \text{ are disjoint}) \\
&= \overline{H - H'} \cup [\overline{t(H/t) - t} - (t\overline{H/t} - t)] \\
&= \overline{H - H'} \cup [\big(\overline{t} \cup t\overline{H/t}\big) - \big(t\overline{H/t} - t\big)] \\
&\qquad \text{(by Lemma 2.4)} \\
&= \overline{H - H'} \cup [\overline{t} \cup \big(t\overline{H/t} - (t\overline{H/t} - t)\big)] \\
&= \overline{H - H'} \cup \overline{t} \cup t \\
&= \overline{H - H'} \cup \overline{t} \\
&= \overline{H - H'}
\end{aligned}$$

To prove the G_i/s – nonblocking property consider two different cases.

1) $t \notin L_m(G_i)/s$: By Lemma 2.5 we have

$$\overline{(H - H') \cap L_m(G_i/s)} = \overline{(H \cap L_m(G_i/s)) - (H' \cap L_m(G_i/s))}$$

Since, in this case $H' \cap L_m(G_i/s) = \varnothing$ we conclude

$$\overline{(H - H') \cap L_m(G_i/s)} = \overline{H \cap L_m(G_i/s)}$$

Furthermore

$$\begin{aligned}
\overline{H - H'} \cap L(G_i/s) &\subseteq \overline{H} \cap L(G_i/s) \\
&= \overline{H \cap L_m(G_i/s)} \\
&= \overline{(H - H') \cap L_m(G_i/s)}
\end{aligned}$$

So, $\overline{H - H'} \cap L(G_i/s) \subseteq \overline{(H - H') \cap L_m(G_i/s)}$. Since the reverse containment always holds we have $\overline{H - H'} \cap L(G_i/s) = \overline{(H - H') \cap L_m(G_i/s)}$

2) $t \in L_m(G_i)/s$: Therefore $t \in L(G_i)/s$. Now, consider:

$$\begin{aligned}
(\overline{H - H'}) \cap L(G_i/s) &= \left(\overline{H} - (t\overline{H/t} - t)\right) \cap L(G_i/s) \\
&= \left(\overline{H} \cap L(G_i/s)\right) - \left((t\overline{H/t} - t) \cap L(G_i/s)\right) \\
&= \left(\overline{H} \cap L(G_i/s)\right) - \Delta L_L \qquad (3.3)
\end{aligned}$$

where $\Delta L_L := (t\overline{H/t} - t) \cap L(G_i/s)$. Next, let $R = \overline{(H - H') \cap L_m(G_i/s)}$ and consider:

$$\begin{aligned}
\overline{H \cap L_m(G_i/s)} &= R \cup \overline{H' \cap L_m(G_i/s)} \\
&= R \cup \overline{(t(H/t) - t) \cap L_m(G_i/s)} \\
&= R \cup \overline{t(H/t - \epsilon) \cap L_m(G_i/s)} \\
&= R \cup \left(t\overline{(H/t - \epsilon) \cap L_m(G_i/st)} \cup \overline{t}\right) \\
&= R \cup t\overline{H/t \cap L_m(G_i/st)} \ \left(\text{since } \overline{t} \subseteq R\right) \\
&= R \uplus \left(t\overline{H/t \cap L_m(G_i/st)} - t\right)
\end{aligned}$$

Let $\Delta L_R = \big(t\overline{H/t \cap L_m(G_i/st)} - t\big)$, then:

$$\overline{(H - H') \cap L_m(G_i/s)} = \overline{H \cap L_m(G_i/s)} - \Delta L_R \tag{3.4}$$

Note that:

$$\begin{aligned}
\Delta L_R &= t\overline{H/t \cap L_m(G_i/st)} - t \\
&= t\overline{H \cap L_m(G_i/s)}/t - t \quad \text{(by Lemma 2.1)} \\
&= t(\overline{H} \cap L(G_i/s))/t - t \quad \text{(H is } L_m(G_i/s)\text{-closed)} \\
&= t(\overline{H/t} \cap L(G_i/st)) - t \quad \text{(by Lemma 2.1)} \\
&= \big(t\overline{H/t} \cap L(G_i/s)\big) - t \\
&= (t\overline{H/t} - t) \cap L(G_i/s) \\
&= \Delta L_L
\end{aligned}$$

From (3.3), (3.4), and G_i/s-nonblocking property of H it follows that $H - H'$ is $G_i/s -$ nonblocking. For controllability we need to ensure that the behavior of $L(G/s)$ can be limited to $\overline{H - H'}$. This goal can be simply achieved by disabling the events after the trace t, which we know are all controllable. Finally, for relative-closure:

$$\begin{aligned}
\overline{H - H'} \cap L_m(G/s) &= \big(\overline{H} - (t\overline{H/t} - t)\big) \cap L_m(G/s) \\
&= \big(\overline{H} \cap L_m(G/s)\big) - \big((t\overline{H/t} - t) \cap L_m(G/s)\big) \\
&= H - \big(\big(t\overline{H/t} \cap L_m(G/s)\big) - t\big) \quad \big(\text{since H is } L_m(G/s)\text{-closed}\big) \\
&= H - \big(t\big(\overline{H/t} \cap L_m(G/st)\big) - t\big) \\
&= H - (t(H/t) - t) \quad \big(\text{since } \overline{H}/t \cap L_m(G/s)/t = H/t) \\
&= H - H'
\end{aligned}$$

This completes the proof. ■

Lemma 3.6. Suppose $s \in \overline{\sup RCN_b(K, \mathcal{G})}$ and $H = \sup RCN_b(K/s, \mathcal{G}/s)$. Then

$$(K/s)_{nnf} \cap \overline{H} \subseteq H.$$

Proof: We know that $(K/s)_{nnf} \subseteq (K/s)_{nf} \subseteq K/s \subseteq L_m(G/s)$. Therefore:

$$\begin{aligned}(K/s)_{nnf} \cap \overline{H} &\subseteq L_m(G/s) \cap \overline{H} \\ &= H \text{ (since H is } L_m(G/s)\text{-closed)}\end{aligned}$$

■

Theorem 3.4. Assume $\overline{K_{nb}} = \overline{K}$ and that there is no SE in $L(G, \gamma^N)$. If $N \geq N_{nnf} + 1$, then $L(G, \gamma^N) = \overline{\sup RCN_b(K, \mathcal{G})}$.

Proof: To prove the theorem we start with the language $H := \sup RCN_b(K/s, \mathcal{G}/s)$ and then construct the language:

$$H' := H - (K/s)_{nnfc}\Sigma^+$$

where $\Sigma^+ = \Sigma^* - \{\epsilon\}$. By considering Lemma 3.6 and the definition of $(K/s)_{nnf}$ we know that all traces in $(K/s)_{nnf} \cap \overline{H}$ satisfy the conditions for Lemma 3.5. Thus, by Lemma 3.5 H' is also G_i/s – nonblocking, controllable, and $L_m(G/s)$-closed. In the following, we first show that $\overline{H'}|_1 = \overline{H}|_1$ and $H' = H'|_{N-1}$ where $N = N_{nnf}+1$. Consider the two following cases:

1. $(K/s)_{nf} - \{\epsilon\} \neq \varnothing$: In this case $(K/s)_{nnf} \neq \varnothing$. Obviously, $\overline{H'}|_1 \subseteq \overline{H}|_1$. So we only need to show the reverse containment. Suppose $t \in \overline{H}|_1$ and therefore $t \in \overline{H}$ and $|t| = 1$. Thus there exists t' such that $tt' \in H$. If $tt' \notin (K/s)_{nnf}\Sigma^+$, then $tt' \in H'$ and

$t \in \overline{H'}|_1$. If $tt' \in (K/s)_{nnf}\Sigma^+$, then there exists $t'' < t'$ such that $tt'' \in (K/s)_{nnf} \cap \overline{H} \subseteq H'$ and therefore $t \in \overline{H'}|_1$. Consequently:

$$\overline{H'}|_1 = \overline{H}|_1$$

Now suppose that $N = N_{nnf} + 1$ and $\overline{K_{nb}} = \overline{K}$. Suppose there exists $t \in H, H'$ such that $|t| \geq N$, then $t \in H \subseteq \overline{H} \subseteq \overline{K/s} = \overline{K_{nb}/s}$. Also, by Definition 3.3 $K_{nb}/s = (K/s)_{nf}$. Consequently, $t \in \overline{(K/s)_{nf}}$. Thus there exists t' such that $tt' \in (K/s)_{nf}$. This is equivalent to the existence of $t'' < tt'$ such that $t'' \in (K/s)_{nnf}$. Since $N = N_{nnf} + 1$, then $|t''| \leq N - 1$. Furthermore, since $t'' < t$ we conclude that $t \in (K/s)_{nnf}\Sigma^+$. This contradicts the first assumption of $t \in H'$. Thus $|t| < N$ and consequently:

$$H' = H'|_{N-1} \subseteq K/s|_{N-1}$$

2. $(K/s)_{nf} - \{\epsilon\} = \varnothing$: In this case $K_{nnf}/s = \varnothing$. This means that $H' = H$. Consequently:

$$\overline{H'}|_1 = \overline{H}|_1$$

On the other hand, $H' = H \subseteq K/s \subseteq \overline{K/s} \subseteq \overline{(K/s)_{nf}} = \{\epsilon\}$. This implies that:

$$H' = H'|_{N-1} \subseteq K/s|_{N-1}$$

Therefore, similar to the Proposition 3.1 we can prove that:

$$H' = \sup RCN_b(H', \mathcal{G}/s) = \sup RCN_b(H', \mathcal{G}/s|_N)$$

And finally:

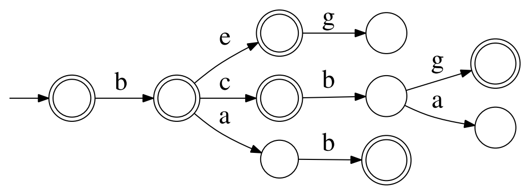
$$\begin{aligned}
& H' \subseteq K/s|_{N-1} \\
\Rightarrow\quad & H' = \sup RCN_b(H', \mathcal{G}/s|_N) \\
& \quad \subseteq \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) := f^N(s) \\
\Rightarrow\quad & \overline{H'}|_1 \subseteq \overline{f^N(s)}|_1 \\
\Rightarrow\quad & \overline{H}|_1 \subseteq \overline{f^N(s)}|_1
\end{aligned}$$

By induction on the length of traces of $\overline{\sup RCN_b(K, \mathcal{G})}$ and consideration of the above result for the induction step, we can conclude that $L(G, \gamma^N) = \overline{\sup RCN_b(K, \mathcal{G})}$. Note that the absence of SE is necessary to establish the base step. ■

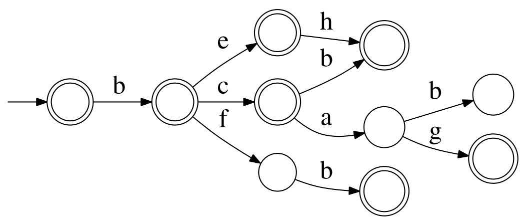
Remark 3.2. The reader should note that in the special case of standard supervisory control (number of plants under supervision is $n = 1$) the conditions in Theorem 3.4 reduce to the conditions of Theorem 5.5 in [7]. Since K_{nb} reduces to K_{mc} (which is the *marked controllable* sub-language of K) and N_{nnf} becomes equal to N_{mcmc} (the maximum distance between two subsequent members of K_{mc}).

3.5 Example 1

Suppose $\mathcal{G} = \{G_1, G_2\}$ as shown in Figure 3.2, and the legal behavior of the plants are described by $K_1 = \{\epsilon, b, bc, bcbg\}$ and $K_2 = \{\epsilon, b, bc, bcag\}$. The marked states in Figure 3.2 are shown by double circles. Assume that $\Sigma_c = \{a, b, c, d, e, f\}$ and $\Sigma_{uc} = \{g, h\}$.



(a) G1



(b) G2

Figure 3.2: Plant automata

In the following we first present the off-line solution for the above mentioned problem.

3.5.1 Off-line Solution

First we synthesize a plant G where $L(G) = L(G_1) \cup L(G_2)$ and $L_m(G) = L_m(G_1) \cup L_m(G_2)$. The automaton of G is shown in Figure 3.3. The overall specification which is calculated based on Equation 3.1 is $K = \{\epsilon, b, bc, bcag, bcbg\}$.

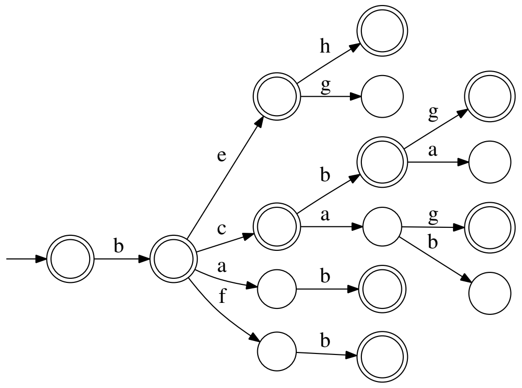


Figure 3.3: Union plant model G

We then solve the problem based on the algorithm presented in [23] to find $\sup RCN_b(K, \mathcal{G})$. The supervisor can be realized by automaton S as shown in Figure 3.4.

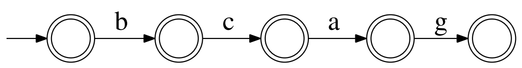


Figure 3.4: Realization of supervisor S

Now that we know how the off-line solution works (i.e. allows only the sequence $bcag$), we can go forward to the online solution.

3.5.2 Online Solution

In this example we focus on finding the maximally permissive RLL supervisor. First we determine N_{nnf}.

Length of Lookahead Window

To determine N_{nnf} we have to determine $(K/s)_{nf}$ and $(K/s)_{nnf}$ for every $s \in K$:

$$\begin{aligned}
(K/\epsilon)_{nf} &= \{\epsilon, b, bc, bcag, bcbg\} \quad , \quad (K/\epsilon)_{nnf} = \{b\} \\
(K/b)_{nf} &= \{\epsilon, c, cag, cbg\} \quad , \quad (K/b)_{nnf} = \{c\} \\
(K/bc)_{nf} &= \{\epsilon, ag, bg\} \quad , \quad (K/bc)_{nnf} = \{ag, bg\} \\
(K/bcb)_{nf} &= \{\epsilon, g\} \quad , \quad (K/bcb)_{nnf} = \{g\} \\
(K/bca)_{nf} &= \{\epsilon, g\} \quad , \quad (K/bca)_{nnf} = \{g\} \\
(K/bcbg)_{nf} &= \{\epsilon\} \quad , \quad (K/bcbg)_{nnf} = \{\} \\
(K/bcag)_{nf} &= \{\epsilon\} \quad , \quad (K/bcag)_{nnf} = \{\}
\end{aligned}$$

Having the sets $(K/s)_{nnf}$ we find $N_{nnf} = 2$.

Control Action Computation

Setting lookahead window to $N = 3$ and considering that $\overline{K_{nb}} = \overline{K}$, we expect $L(G, \gamma^3)$ to be maximally permissive. Starting from $s = \epsilon$, both models are compatible with the *currently executed trace*; the output of $f^N_{\Uparrow}$ block is shown in Figure 3.5.

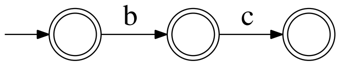


Figure 3.5: $f_N(\epsilon)$

So, the control action is $\gamma^3(\epsilon) = \{b\}$. After executing event b, with $s = b$, the control action is $\gamma^3(b) = \{c\}$, and the output of block $f^N_{\Uparrow}$ is as Figure 3.6. One can see that executing further events, and going ahead through the lookahead window causes new events to be enabled. The control action following string $s = bc$ is $\gamma^3(bc) = \{a\}$.

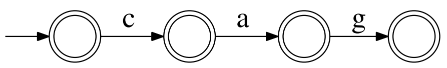


Figure 3.6: $f_N(b)$

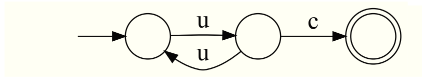
Continuing with execution of the events the next control action would be $\gamma^3(bca) = \{g\}$. This shows that by choosing the window size long enough

$$L(G, \gamma^N) = \overline{\sup RCN_b(K, \mathcal{G})} = \overline{bcag}.$$

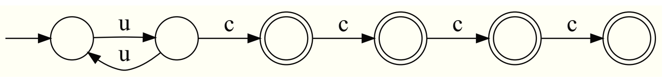
3.6 Example 2

As previously discussed, a sufficient lookahead window size for optimality of RLL supervisor is defined if and only if N_{nnf} is defined and bounded. However, this is not always the case. In some problems the N_{nnf} may become undefined or unbounded. This affects the performance of RLL supervisor. The next example illustrates a case where N_{nnf} is unbounded.

Example 3.1. Consider $\mathcal{G} = \{G_1, G_2\}$ as shown in Figure 3.7. Assume that there is no illegal behavior ($K_i = L_m(G_i)$). If $s = \epsilon$, then $(K/\epsilon)_{nf} = \{u^m c, u^m c^2, u^m c^3, u^m c^4 \mid m \text{ is odd}\}$ and $(K/\epsilon)_{nnf} = \{u^m c \mid m \text{ is odd}\}$. Since there exists $t \in (K/\epsilon)_{nnf}$ such that $|t| = \infty$ N_{nnf} is unbounded.



(a) G1



(b) G2

Figure 3.7: RNSCP with unbounded N_{nnf}

To explain more about necessity of an unbounded window, consider the tree-expansion of the union model after $s = \epsilon$ as shown in Figure 3.8. We observe that for any bounded length of lookahead window, there exists a path of uncontrollable events from the initial state to a pending trace. As we are treating the pending traces to be illegal, for any bounded window size we encounter a SE. However the off-line solution is non-empty. In the next chapter, we propose an RLL supervisor with state information in order to address this issue in the next chapter.

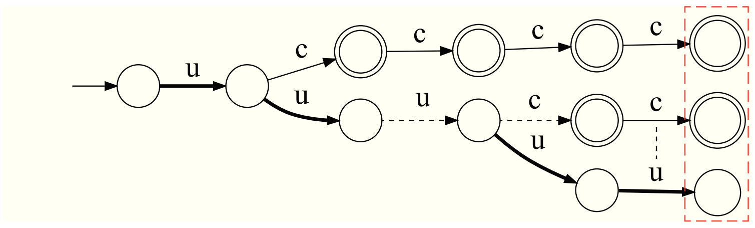


Figure 3.8: Tree-expansion of the RNSCP with unbounded N_{nnf}

3.7 Example 3: Computational Complexity

In this section we investigate the effectiveness of the proposed RLL supervision in handling computational complexity. For this we solve a problem of supervisory control with multiple marking set (which was reviewed in Example 2.2 Chapter 2) using first the off-line approach. Next we solve the problem in an online set up using RLL policy. For this problem a random acyclic automaton is generated and assigned two different sets of marked states. Next the automaton is used to formulate a Robust Nonblocking Supervisory Control Problem with Multiple Marking (RNSCP-MM). All plant sequences are considered legal ($K_i = L_m(G_i)$) and the objective is to ensure the nonblocking property. We obtain the length of "N" to generate maximal permissiveness and then solve the robust control problem for various tree expansions and report the average and maximum sizes of three expansions.

The results are reported in Table 3.1. As we can see the proposed RLL supervisor is a very effective approach for handling the computational complexity associated with the RNSCP. The effectiveness becomes more pronounced when the number of states and transitions in the plant increase. Considering the last row in Table 3.1, we observe that the off-line problem with about 2500 states and 10000 transitions has reduced to an online problem with an average size of 184 states and 487 transitions. Specifically, the average state size of the online problem (i.e. tree expansion) is about $\%7$ of the off-line problem (i.e. plant state size).

# states	# transitions	Window Size	Tree expansion size			
			avg (states)	max (states)	avg (trans)	max (trans)
415	1490	3	8	29	10	48
751	2865	6	61	338	128	920
1296	5461	6	73	359	174	1056
1885	7343	4	18	85	22	137
2545	10199	8	184	862	487	2985

Table 3.1: Comparison between online and off-line complexity.

Finally, we study the effect of three different parameters on the sufficient lookahead window size.

i) Maximum distance between two states of the automaton: First, as explained before a random acyclic automaton is generated with two plants. Next we marked $\%30$ of the states of each automaton. The maximum distance between two states of the automaton was set to $N_D = 6, 10, 14, 18$ and 22. The size of automata was selected proportional to N_D. The results are reported in Table 3.2. The reported window size is the sample mean of the required lookahead window over 25 experiments (rounded to the nearest integer). We observe that the length of sufficient lookahead window is highly correlated with the maximum distance between the states of the automata.

# states	# transitions	Max distance	Window Size	Tree expansion size			
				avg (states)	max (states)	avg (trans)	max (trans)
170	634	6	3	10	59	15	119
639	1288	10	5	43	142	91	390
591	2161	14	8	137	343	411	1205
768	3005	18	10	193	399	650	1478
989	4038	22	14	310	590	1167	2352

Table 3.2: Effect of maximum distance between two states on the sufficient window size.

ii) The size of automata: A random automaton for RNSCP-MM was generated with two sets of marked stats. We randomly marked $\%30$ of the states of each automaton. The maximum distance between two states of the automaton was fixed to $N_D = 10$. The size of the automata was increased gradually (5 times), and the required length of lookahead window was determined. The results are reported inn Table 3.3. We observe that there is no high correlation between the length of sufficient lookahead window and the size of automaton if the maximum distance between the states remains unchanged.

# states	# transitions	Max distance	Window Size	Tree expansion size			
				avg (states)	max (states)	avg (trans)	max (trans)
369	1288	10	5	43	142	91	390
435	1748	10	4	40	134	77	349
738	2891	10	5	74	333	138	835
964	3530	10	6	111	461	304	1683
1241	4659	10	6	113	564	251	1732

Table 3.3: Effect of size of automata on the sufficient window size.

iii) The ratio of marked states: A random automaton for RNSCP-MM was generated with two sets of marked states. The maximum distance between two states of the automaton was $N_D = 10$ and the two automata did not change during the experiment. We randomly marked $30\%, 40\%, 60\%, 80\%$ and 90% of the states of each automaton in five experiments. We monitored the sufficient window size. The results are reported in Table 3.4. We observe that by increasing the percentage of marked states the required window size increases and it settles at a number (10). As an explanation, we can say that increasing the number of marked states causes the worst case scenario (i.e. the longest path between two consecutive nonblocking states) to be inside the legal marked behavior.

# states	# transitions	Marked (%)	Window Size	Tree expansion size			
				avg (states)	max (states)	avg (trans)	max (trans)
415	1490	30%	5	58	151	135	442
415	1490	40%	7	90	256	258	914
415	1490	60%	10	134	415	367	1490
415	1490	80%	10	134	415	367	1490
415	1490	90%	10	134	415	367	1490

Table 3.4: Effect of the percentage of marked states on the sufficient window size.

3.8 Conclusion

In this chapter, Robust Limited Lookahead (RLL) supervisor has been proposed as an extension of the conventional (non-robust) limited lookahead policy. The chapter proves that under certain condition, the optimality of RLL supervisor can be guaranteed if the window size is sufficiently long. A case where the required window size for optimality of the RLL supervisor becomes unbounded has been discussed. The next chapter presents the state-based RLL supervisor in order to overcome the issue of unbounded required window size.

Chapter 4

Limited Lookahead Policy for Robust Nonblocking Supervisory Control: A State-based Approach

In Chapter 3 we solved Robust Nonblocking Supervisory Control Problem (RNSCP) using an online approach called Robust Limited Lookahead (RLL) supervision. The RLL supervisor is language-based and the control domain presented there is based on the strings. Under certain condition, the optimality of RLL supervisor can be guaranteed if the window size is sufficiently long. As we discussed at the end of Chapter 3, a sufficient length for lookahead window of optimal RLL supervisor is not always available.

In this chapter, we study the state-based RNSCP problem (RNSCP-S) in which the design specifications are given in terms of legal/illegal states and the supervisory map is a state feedback control law. We will develop an RLL algorithm for RNSCP-S. Furthermore, we provide a set of conditions under which the RLL supervisor with State information (referred to as RLL-S supervisor) is maximally permissive.

In Section 4.1 we first discuss the requirements for control domain to be taken as states.

Then we formulate Robust Nonblocking Supervisory Control Problem with State information (referred to as RNSCP-S). We also show that the state-based control requirements do not impose any limitation on RNSCP-S by presenting a procedure for conversion of any RNSCP problem to RNSCP-S if the requirements are not met as a priori.

In Section 4.2 we present the implementation of RLL-S supervisor. Next, we discuss the properties of RLL-S supervisor and find a set of conditions for the optimality of RLL-S supervisor. We present an illustrative example at the end.

4.1 State-based Supervisory Control

This section is intended to set up state-based RNSCP (RNSCP-S). First, we formulate state-based supervisory control problem and discuss its requirements. Next, we study the conversion of any RNSCP problem to an equivalent RNSCP-S problem.

4.1.1 State-based Conventional Supervisory Control Problem

State-based supervisory control assumes (without loss of generality) that the specification K is marked by an automaton H which is a subautomaton of the plant G. This means that for a plant $G = (\Sigma, X, \delta, x_0, X_m)$ the specification should be in the form of an automaton $H = (\Sigma, X_H.\delta_H, x_0, X_{mH})$, where $X_H \subseteq X$, $X_{mH} \subseteq X_m$, δ_H is a restriction of δ and $K = L_m(H)$ [9]. If the conditions are not satisfied, [31] presents a procedure to transform the plant and its specification appropriately so that the control domain can be taken as states.

The goal is to design a *state-feedback supervisory controller* $V : X \to 2^{\Sigma}$ which determines the set of events to be enabled at each state of the plant under supervision such that:

$$
\begin{aligned}
&1)\quad L_m(G,V) \subseteq L_m(H)\\
&2)\quad L(G,V) = \overline{L_m(G,V)}
\end{aligned}
$$

Note that the state set of the closed-loop system is the same as G, however, not all the states are necessarily reachable [30].

4.1.2 State-based Robust Supervisory Control Problem

Definition 4.1. Mutually Refined Automata: Consider the two automata:

$$G_1 = (\Sigma_1, X_1, \delta_1, x_{01}, X_{m1}) \text{ and } G_2 = (\Sigma_2, X_2, \delta_2, x_{02}, X_{m2})$$

Assume that there is a one-to-one relation between a subset of X_1 and X_2. For simplicity, assume the corresponding states have the same label. We say that G_1 and G_2 are mutually refined if $\delta_1(x_{01}, s) = \delta_2(x_{02}, s)$ for $s \in L(G_1) \cap L(G_2)$. Also, $\delta_1(x_{01}, s) \neq \delta_2(x_{02}, t)$ if $s \in L(G_1) - L(G_2)$ for all $t \in L(G_2)$, and $\delta_1(x_{01}, t) \neq \delta_2(x_{02}, s)$ if $s \in L(G_2) - L(G_1)$ for all $t \in L(G_1)$.

The requirement for taking the control domain of robust nonblocking supervisory control problem as states is given in [6] as follows.

Theorem 4.1. [6] If any two automata in $\{G_1, ..., G_n, H_1, ..., H_n\}$ are mutually refined then the control domain of robust nonblocking supervisory control problem can be taken as states. ■

State-based RNSCP Problem Formulation (RNSCP-S): Suppose the model of the system belongs to a finite set of possible models $\mathcal{G} = \{G_1, ..., G_n\}$. Each model has its own specification which is marked by the automaton H_i, $i = 1, .., n$ where H_i is a subautomaton of G_i. Assume any two automata in $\{G_1, ..., G_n, H_1, ..., H_n\}$ are mutually refined so the control action can be taken as states. The goal is to design a state feedback law $V : \bigcup_i^n X_i \to 2^{\Sigma}$ so that:

$$\begin{aligned} &1) \quad L_m(G_i, V) \subseteq L_m(H_i) \\ &2) \quad L(G_i, V) = \overline{L_m(G_i, V)} \end{aligned}$$

In this Chap we find a lookahead solution for this problem.

4.1.3 Converting RNSCP to RNSCP-S

Generally speaking, the sufficient condition in Theorem 4.1 may not be satisfied in an RNSCP. This by no means causes a limitation to supervisory control with state information as we can always refine the automata appropriately to satisfy the conditions.

The following definition is a generalization of biased synchronous product presented in [31] and is required for the conversion of RNSCP to RNSCP-S.

Definition 4.2. Multiple biased synchronous product [6] Given a set of automata: $R = \{R_1, R_2, ..., R_n\}$ with $R_i = (X_i, \Sigma_i, \delta_i, x_{0i}, X_{mi})$ the multiple biased synchronous product of R_k is defined as:

$$R_k \parallel_{mr} (R - \{R_k\}) \coloneqq Ac(X_1 \times ... \times X_n, \Sigma_k, \delta, (x_{01}, ...x_{0n}), X_1 \times ... \times X_{k-1} \times X_{mk} \times X_{k+1} \times X_n)$$

where:

$$\delta\big((x_1, ..., x_n), \sigma\big) = \begin{cases} (x_1', ...x_n') & \text{if } \sigma \in \Sigma_{R_k}(x_k) \\ \text{undefined} & \text{otherwise} \end{cases}$$

and

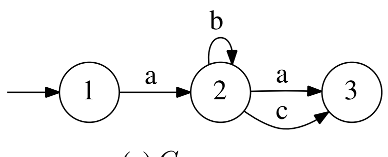
$$x_i' = \begin{cases} \delta_i(x_i,\sigma) & \text{if } \sigma \in \Sigma_{R_i}(x_i) \\ x_i & \text{if } \sigma \notin \Sigma_{R_i}(x_i) \end{cases}$$

The key point is that the multiple biased production of R_k does not alter its generated or marked languages. Thus [6] presents the following procedure for mutual refinement of a set of plants and their specifications. It shows that the closed and marked languages of the resulting automata remain unchanged and that any two automata are mutually refined. Consequently, the resulting problem can be treated as a RNSCP-S.

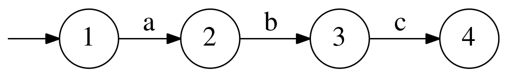
Procedure 4.1.

1. Define the set $R := \{G_1, ..., G_n, H_1, ..., H_n\}$. Add a dump state to each automaton R_i and add transitions $\Sigma - \Sigma_{R_i}(x)$ from each state x to the dump state. Then, add self-loop Σ to the dump state. The resulting automata are denoted as $R' := \{G_1', ..., G_n', H_1', ..., H_n'\}$
2. Replace G_i' in R' with G_i and derive $G_i'' = G_i \parallel_{mr} (R' - \{G_i\})$ for all $i = 1, ..., n$.
3. Replace H_i' in R' with H_i and derive $H_i'' = H_i \parallel_{mr} (R' - \{H_i\})$ for all $i = 1, ..., n$.

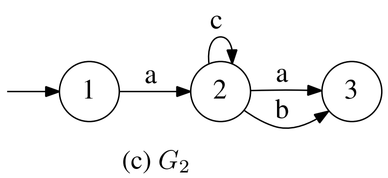
Example 4.1. Consider $\mathcal{G} = \{G_1, G_2\}$ as shown in Figure 4.1 and the corresponding specifications K_1 and K_2. Initially, the control domain can not be taken as states. For example, in G_1 we observe that $\delta(1, abc) = \delta(1, aa)$ considering the fact that abc is legal while aa is illegal we understand that the control domain can not be taken as states.



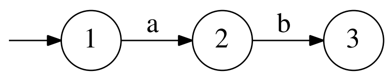
(a) G_1



(b) K_1



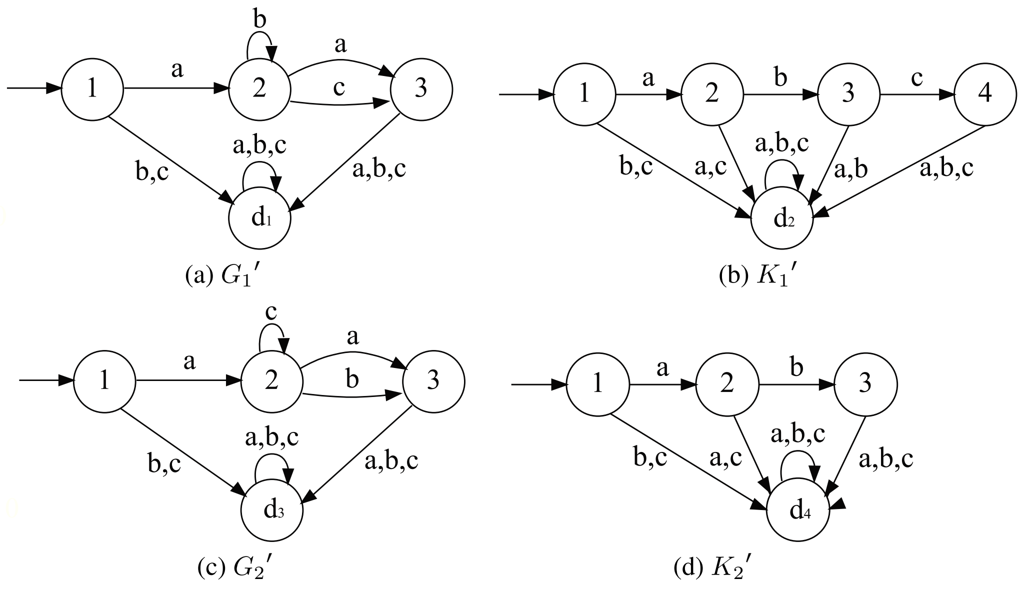
(c) G_2



(d) K_2

Figure 4.1: Plants and specifications before refinement

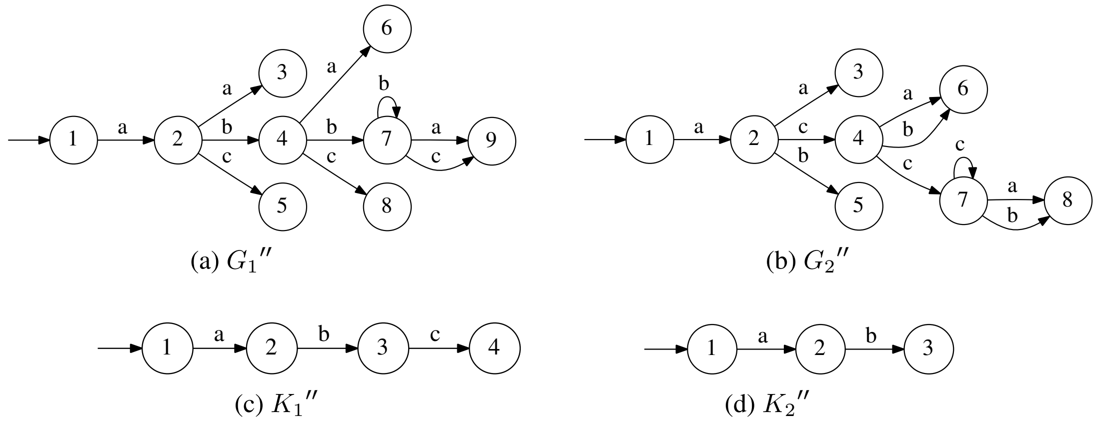
Following the Procedure 4.1 we first add a dump state to all the generators in order to have a closed-behavior equal to Σ^*. The resulting automata are shown in Figure 4.2



(a) $G_1{}'$ (b) $K_1{}'$ (c) $G_2{}'$ (d) $K_2{}'$

Figure 4.2: Modified automata: Example 4.1

Next we find the set of refined automata as $G_i'' = G_i \,\|_{mr}\, (R' - \{G_i\})$ and $H_i'' = H_i \,\|_{mr}\, (R' - \{H_i\})$ where $i = 1, 2$. The set of refined automata is shown in Figure 4.3. We can verify that the refined problem satisfies the assumption of an RNSCP-S problem.



(a) $G_1{}''$ (b) $G_2{}''$ (c) $K_1{}''$ (d) $K_2{}''$

Figure 4.3: Refined automata: Example 4.1

4.2 Robust Limited Lookahead Supervision with State Information

4.2.1 Linguistic Preliminaries

Let $G = (X, \Sigma, \delta, x_0, X_m)$ and $L(G) := \{s \in \Sigma^* : \delta(x_0, s)$ is defined$\}$. We define $[x]$ as the set of all traces leading to state x from the initial state x_0:

$$[x] = \{s \in \Sigma^* \mid \delta(x_0, s) = x \ \wedge (\nexists\, t < s \ : \ \delta(x_0, t) = x)\}$$

Definition 4.3. Post-automaton $G/_{[x]}$ is the subautomaton of G which is reachable from the state $x \in X_G$. $L(G/_{[x]})$ denotes the language of $G/_{[x]}$.

$$L(G/_{[x]}) := \{t \in \Sigma^* \mid s \in [x] \wedge \delta(x_0, st) \in X_G\}$$

Obviously, $L(G/_{[x]}) = L(G/s)$. ■

Definition 4.4. Truncation of *G* after [x] The truncation of $G/_{[x]}$ to N is the subautomaton of G which is reachable within N steps from x and it is denoted by $G/_{[x]}|_N$.

In general we can see that $L(G/s)|_N \subseteq L(G/_{[x]})|_N$. If $t \in L(G/_{[x]})|_N$ and $t \notin L(G/s)|_N$, then $|t| > N$.

Expansion as a subgraph The main difference between RLL and RLL-S supervisors is the expansion procedure. The RLL supervisor estimates the behavior of a plant under supervision using an N-step tree expansion. A tree is a structure which distinguishes states based on their event history without making any assumption about the equivalence of the states. Thus tree structure is a useful tool since it permits to compute the control action with no need of state information, while it is also limiting because the size of state space of a lookahead tree increases quickly with increasing the length of lookahead window [26]. It may also cause the length of required window for the optimality of RLL supervisor to be unbounded.

These problems can be resolved if we replace the tree structure with a subgraph. Similar to the RLL supervision we assume that we are given $L(G_i/s|N)$ and $L(K/s|N)$ for any executed trace s along with the state following each event. The information is enough to estimate the future behavior of each model as a subgraph. We should note that the size of state space for expansion of an automaton as a subgraph is bounded by the automaton's number of states which is always finite in the case of finite state automata.

Example 4.2. Consider the automaton in Figure 4.4. Let the length of lookahead window to be N = 2. The subgraph expansion at state 1 of the automaton is shown in Figure 4.5.

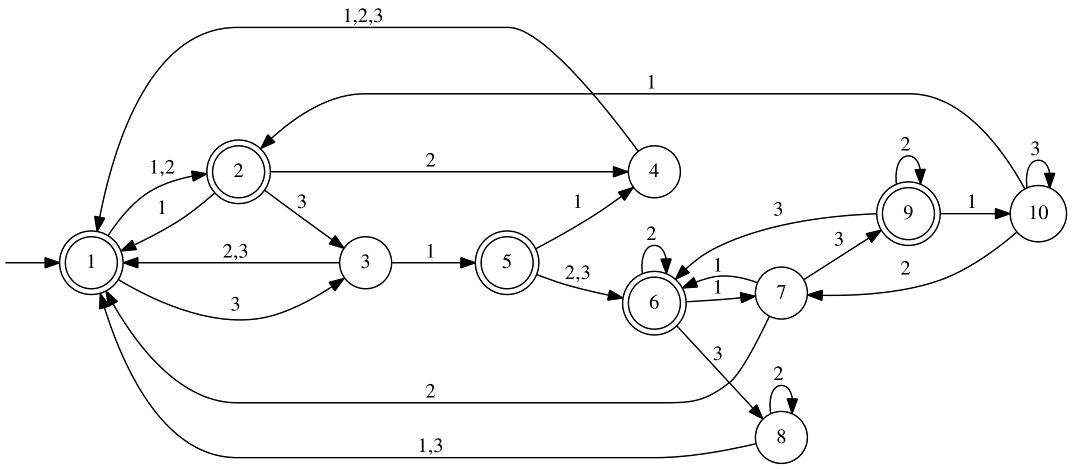


Figure 4.4: Automaton G: Example 4.2

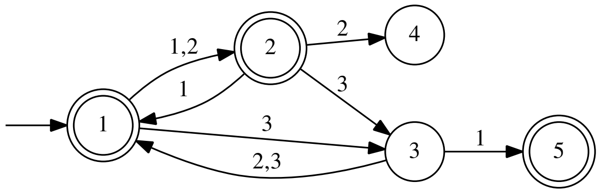


Figure 4.5: Subgraph expansion of G: Example 4.2

Now, if we expand the plant G as a tree, the outcome would be the automaton shown in Figure 4.6. Comparing the tree-expansion with the subgraph expansion we observe that

(i) the language of the tree expansion is a subset of the language of the subgraph expansion and (ii) the size of state space for three expansion is larger. ■

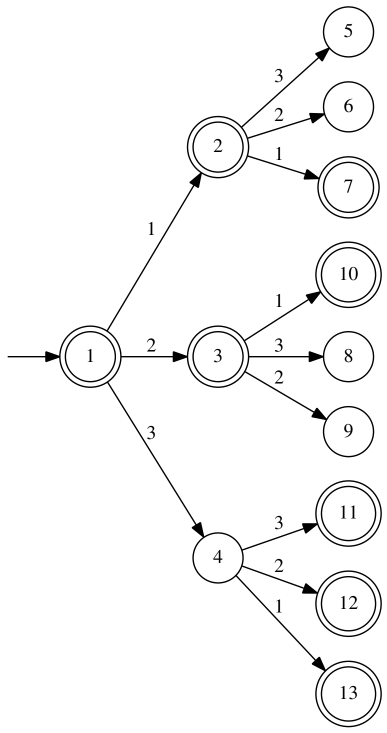


Figure 4.6: Tree expansion of G: Example 4.2

Definition 4.5. Pending State A state is pending if it is only encountered at the frontier of $L(G/s|_N)$. It is formally defined as:

$$X_{pending} = \{x \in X_G \mid x \in X_{G/[x]|_N} \wedge x \notin X_{G/[x]|_{N-1}}\}$$

■

In the preceding example, states 4 and 5 are designated as pending states. Taking a conservative approach these states must be treated as illegal when solving a problem in online fashion.

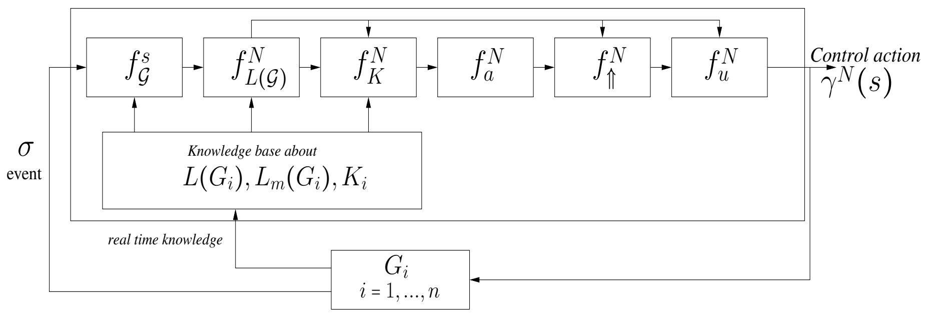


Figure 4.7: Block diagram of State-based Robust Limited Lookahead Supervisor

4.2.2 RLL-S Supervisor

In this section we discuss Robust Limited Lookahead supervisor with State information (RLL-S). Compared with the RLL supervisor, the procedure steps remain unchanged except the expansion procedure. Here we briefly explain how the RLL supervisor should be modified to properly accommodate the problem of undefined required window size associated with the RLL supervisor. Further, we briefly discuss the properties of the RLL-S supervisor compared with the RLL supervisor. We also find a lower bound for the lookahead window size which guarantees the optimality of the RLL-S supervisor.

Consider a problem which is formulated as RNSCP-S and thus the control domain can be taken as states. Referring to Figure 4.7, the RLL-S supervisor is briefly described. Similar to RLL supervisor, suppose that the process has executed the trace $s \in [x]$ so far. The online control functions in Figure 4.7 are described in the following:

1. Similar to RLL supervisor the block $f^s_{\mathcal{G}}$ identifies the plants which are compatible with the currently executed trace s where $s \in [x]$ and x is current state of the plant. Without loss of generality suppose that the first n' models are compatible with s (i.e. G_i, $i = 1, ..., n'$).

2. On the basis of knowledge available about the plants, the supervisor *predicts* the possible behavior (<u>states and language</u>) of all compatible plant models *N* step beyond the current state *x*. It is the function of block $f^N_{L(\mathcal{G})}$ to build subautomaton $G_i/_{[x]}|_N$ of

all models.

3. The supervisor then determines which states in the subautomaton of $G_i/_{[x]}|_N$ are illegal. State x is illegal if it is illegal in some plant model. In other words, x is illegal if $x \in \bigcup_{i=1}^{n}\big(X(G_i) - X(H_i)\big)$. Removing illegal states is to be done by the block f_K^N in Figure 4.7.

4. Next we take the conservative attitude toward the pending states since they may reach into illegal region by executing uncontrollable events. This is to be done by the block f_a^N.

5. Afterwards, the block $f_{\Uparrow}^N$ computes the supremal relative-closed, controllable, and $\mathcal{G}/_{[x]}|_N$-nonblocking part of the output of block f_a^N with respect to the $G_i/_{[x]}|_N$, $i = 1, ...n'$, and $G/_{[x]}|_N$.

6. Finally, the block f_u^N finds the enabled events for the currently executed string s, by restricting the output of block $f_{\Uparrow}^N$ to single-event sequences.

4.2.3 Properties of RLL-S Supervisor

In this section we study some of the properties of the RLL-S supervisor. We also compare the RLL-S with RLL supervisor.

Lemma 4.1. Let $s \in \Sigma^*$ and $s \in [x]$. Then:

$$\sup RCN_b(K/s|_{N-1}, \mathcal{G}/_{[x]}|_N) \subseteq \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}|_N)$$

The proof follows the fact that $K/s|_{N-1} \subseteq K/_{[x]}|_{N-1}$. Note that we are applying the same monotone operator on both sides of the inclusion, thus the inclusion would be preserved under the operation. ■

Lemma 4.2. Let $s \in \Sigma^*$ and $s \in [x]$. Then:

$$\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) = \sup RCN_b(K/s|_{N-1}, \mathcal{G}/_{[x]}|_N)$$

Proof: Let

$$\begin{aligned}
&LHS := \sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N)\\
&RHS := \sup RCN_b(K/s|_{N-1}, \mathcal{G}/_{[x]}|_N)\\
&\Delta L_i = L(G_i/_{[x]}|_N) - L(G_i/s|_N)\\
&\Delta L_{m,i} = \Delta L_i \cap L_m(G_i/_{[x]}|_N)\\
&\Delta L = L(G/_{[x]}|_N) - L(G/s|_N)\\
&\Delta L_m = \Delta L \cap L_m(G/_{[x]}|_N)
\end{aligned}$$

Note that $\forall t \in \Delta L_i \rightarrow |t| \geq N+1$. Also $\forall t' \in RHS$ and LHS, then $|t'| \leq N-1$. First we prove that $LHS \subseteq RHS$. For this we prove that LHS is relative closed and controllable with respect to $G/_{[x]}|_N$ and nonblocking with respect to $G_i/_{[x]}|_N$, $i = 1, ..., n'$.

i) G-nonblocking: Since LHS is $G_i/s|_N$-nonblocking we have

$$\overline{LHS} \ \cap \ L(G_i/s|_N) = \overline{LHS \cap L_m(G_i/s|_N)}$$

Also, $\overline{LHS} \cap \Delta L_i = \overline{LHS \cap \Delta L_{m,i}} = \varnothing$. Thus:

$$\begin{aligned}
\overline{LHS \cap L_m(G_i/_{[x]}|_N)} &= \overline{LHS \cap (L_m(G_i/s|_N) \cup \Delta L_{m,i})}\\
&= \overline{LHS \cap L_m(G_i/s|_N)} \cup \overline{LHS \cap \Delta L_{m,i}}\\
&= \left(\overline{LHS} \cap L(G_i/s|_N)\right) \cup \left(\overline{LHS} \cap \Delta L_i\right)\\
&= \overline{LHS} \cap L(G_i/_{[x]}|_N)
\end{aligned}$$

ii) Controllability: By assumption, $\overline{LHS}\Sigma_u \cap L(G/s|_N) \subseteq \overline{LHS}$. Also, $\forall t \in LHS$, $|t| \leqslant N-1$. So for all $t \in LHS$ and $\sigma \in \Sigma$, $|t\sigma| \leq N$ and thus $\overline{LHS}\Sigma_u \cap \Delta L = \varnothing$.

$$\begin{aligned}\overline{LHS}\Sigma_u \cap L(G/_{[x]}|_N) &= \overline{LHS}\Sigma_u \cap \big(L(G/s|_N) \cup \Delta L\big)\\ &= \big(\overline{LHS}\Sigma_u \cap (L(G/s|_N)\big) \cup \big(\overline{LHS}\Sigma_u \cap \Delta L\big)\\ &\subseteq \overline{LHS}\end{aligned}$$

iii) $L_m(G)$-closure: We know that $\overline{LHS} \cap L_m(G/s|_N) = LHS$ and $\overline{LHS} \cap \Delta L_m = \varnothing$. Thus we can easily conclude that $\overline{LHS} \cap L_m(G/_{[x]}|_N) = \overline{LHS} \cap \big(L_m(G/s|_N) \cup \Delta L_m\big) = LHS$.

Now we prove the other inclusion which is $RHS \subseteq LHS$ following a similar procedure.

i) G-nonblocking: Since RHS is $G_i/_{[x]}|_N$-nonblocking, we have:

$$\overline{RHS} \cap L(G_i/_{[x]}|_N) = \overline{RHS \cap L_m(G_i/_{[x]}|_N)}$$

Also $\overline{RHS} \cap \Delta L_i = RHS \cap \Delta L_{m,i} = \varnothing$. Thus:

$$\begin{aligned}\overline{RHS \cap L_m(G_i/s|_N)} &= \overline{RHS \cap (L_m(G_i/s|_N) \cup \Delta L_{m,i})}\\ &= \overline{RHS \cap L_m(G_i/_{[x]}|_N)}\\ &= \overline{RHS} \cap L(G_i/_{[x]}|_N)\\ &= \overline{RHS} \cap \big(L(G_i/s|_N) \cup \Delta L_i\big)\\ &= \overline{RHS} \cap L(G_i/s|_N)\end{aligned}$$

ii) Controllability: By assumption, $\overline{RHS}\Sigma_u \cap L(G/_{[x]}|_N) \subseteq \overline{RHS}$. Also, we know $\forall t \in$

RHS, $|t| \leqslant N-1$. So for all $t \in RHS$ and $\sigma \in \Sigma$, $|t\sigma| \leq N$ and thus $\overline{RHS}\Sigma_u \cap \Delta L = \varnothing$.

$$\begin{aligned}
\overline{RHS}\Sigma_u \cap L(G/s|_N) &= \overline{RHS}\Sigma_u \cap \big(L(G/s|_N) \cup \Delta L\big) \\
&= \overline{RHS}\Sigma_u \cap L(G/_{[x]}|_N) \\
&\subseteq \overline{RHS}
\end{aligned}$$

iii) $L_m(G)$-closure: By assumption, $\overline{RHS} \cap L_m(G/_{[x]}|_N) = RHS$ and $\overline{RHS} \cap \Delta L_m = \varnothing$. Thus we can easily conclude that $\overline{RHS} \cap L_m(G/s|_N) = \overline{RHS} \cap \big(L_m(G/s|_N) \cup \Delta L_m\big) = \overline{RHS} \cap L_m(G/_{[x]}|_N) = RHS$.

■

Lemma 4.3. Let $s \in \Sigma^*$ and $s \in [x]$. Then:

$$\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) \subseteq \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}|_N)$$

Proof:

$$\begin{aligned}
\sup RCN_b(K/s|_{N-1}, \mathcal{G}/s|_N) &\subseteq \sup RCN_b(K/s|_{N-1}, \mathcal{G}/_{[x]}|_N) && \text{(by Lemma 4.2)} \\
&\subseteq \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}|_N) && \text{(by Lemma 4.1)}
\end{aligned}$$

■

Property 4.1. It follows from Lemma 4.3 that, in general, the RLL-S supervisor is more permissive than the RLL supervisor:

$$L(\mathcal{G}, \gamma^N_{RLL}) \subseteq L(\mathcal{G}, \gamma^N_{RLL-S})$$

Lemma 4.4. Let $s \in \Sigma^*$ and $s \in [x]$. Then:

$$\sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}) \subseteq \sup RCN_b(K/_{[x]}, \mathcal{G}/_{[x]})$$

Proof: The proof follows the fact that $K/_{[x]}|_{N-1} \subseteq K/_{[x]}$. ■

Lemma 4.5. Let $s \in \Sigma^*$ and $s \in [x]$. Then:

$$\sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}|_N) = \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]})$$

Proof: Let

$$LHS := \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}|_N)$$
$$RHS := \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]})$$
$$\Delta L_i = L(G_i/_{[x]}) - L(G_i/_{[x]}|_N)$$
$$\Delta L_{m,i} = \Delta L_i \cap L_m(G_i/_{[x]})$$
$$\Delta L = L(G/_{[x]}) - L(G/_{[x]}|_N)$$
$$\Delta L_m = \Delta L \cap L_m(G/_{[x]})$$

Further, note that:

$$\Delta L_{m,i} \cap L_m(G_i/_{[x]}|_N) = \varnothing, LHS \subseteq L_m(G_i/_{[x]}|_N) \to \Delta L_{m,i} \cap LHS = \varnothing$$
$$\Delta L_i \cap L(G_i/_{[x]}|_N) = \varnothing, \overline{LHS} \subseteq L(G/_{[x]}|_N) \to \Delta L_i \cap \overline{LHS} = \varnothing$$
$$\Delta L \cap L(G/_{[x]}|_{N-1})\Sigma_u = \varnothing, \overline{LHS}\Sigma_u \subseteq L(G/_{[x]}|_{N-1})\Sigma_u \to \Delta L \cap \overline{LHS}\Sigma_u = \varnothing$$
$$\Delta L \cap L(G/_{[x]}|_N) = \varnothing, \overline{LHS} \subseteq L(G/_{[x]}|_N) \to \Delta L \cap \overline{LHS} = \varnothing \to \Delta L_m \cap \overline{LHS} = \varnothing$$

Similar statements can be driven for *RHS*. Having the above mentioned statements the rest of proof follows a same procedure as the Lemma 4.2 and its removed for brevity.

■

Lemma 4.6. Let $s \in \Sigma^*$ and $s \in [x]$. Then:

$$\sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}|_N) \subseteq \sup RCN_b(K/s, \mathcal{G}/s)$$

Proof:

$$\begin{aligned}
\sup RCN_b(K/s, \mathcal{G}/s) &= \sup RCN_b(K/_{[x]}, \mathcal{G}/_{[x]})) \qquad \text{(since } s \in [x]\text{)} \\
&\supseteq \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}) \qquad \text{(by Lemma 4.4)} \\
&= \sup RCN_b(K/_{[x]}|_{N-1}, \mathcal{G}/_{[x]}|_N) \qquad \text{(by Lemma 4.5)}
\end{aligned}$$

■

Property 4.2. In general the RLL and RLL-S supervisors are both legal and the behavior of the system under supervision of these supervisors is a subset of the behavior of the system under supervision of the optimal off-line supervisor V.

$$L(\mathcal{G}, {\gamma^N}_{RLL}) \subseteq L(\mathcal{G}, {\gamma^N}_{RLL-S}) \subseteq L(\mathcal{G}, V)$$

Proposition 4.1. Assume that the RLL supervisor is maximally permissive $L(\mathcal{G}, {\gamma^N}_{RLL}) = L(\mathcal{G}, V)$. Then the RLL-S supervisor too would be maximally permissive $L(\mathcal{G}, {\gamma^N}_{RLL-S}) = L(\mathcal{G}, V)$.

Proof: The proof directly follows from Property 4.2.

■

4.2.4 Maximally Permissive RLL-S Supervisor

In this section, we would like to find a set of sufficient conditions which guarantee that RLL-S is maximally permissive. We assume that the problem is already converted to

RNSCP-S form. The set of plants and their specifications is $\{G_1, ..., G_n, H_1, ..., H_n\}$, the union model is G and the overall specification is characterized by the automaton H.

Definition 4.6. Given a set of plants G_i, $i = 1, ..., n$, the set of *nonblocking states* is defined as:

$$H_{nb} := \{x \in H \mid \overline{\overline{[x]} \cap L_m(G_i)} = \overline{[x]} \cap L(G_i) \wedge \big(\forall \sigma \in \Sigma_u : [x]\sigma \notin L(G)\big)\}.$$

Next we define $w(x, s)$ which contains the set of *reachable states from the state x through execution of string s*, if none of these states belongs to H_{nb}. $w(x, s)$ is defined for each given $x \in X_H$ and $s \in L(H/_{[x]})$ as follows:

$$w(x,s) := \begin{cases} \{\delta(x,t) : t \leq s\} & \text{if } \delta(x,s) \in X_{H_{nb}} \wedge (\forall \epsilon < t < s) : \delta(x,t) \notin X_{H_{nb}} \\ \varnothing & \text{otherwise} \end{cases}$$

The desired bound for optimality of the RLL-S supervisor, N, is then defined as follows:

$$N_B := \max_{x \in X_H, s \in L(H/_{[x]})} |w(x,s)|$$

Note that N_B is always defined in case of finite state automata. This is not always true for N_{nnf} which determines the required bound for RLL supervisor. We should also note that if N_{nnf} is defined then $N_{nnf} + 1 = N_B$ (Because, we count the number encountered states between two states in X_H, instead of counting the events in between).

The following lemma generalizes Lemma 3.5 to the case of state-based supervisory control. Note that for notational simplicity we use H instead of $L(H)$.

Lemma 4.7. *$H \in RCN_b(K/_{[y]}, \mathcal{G}/_{[y]})$ and $[x] \subseteq \overline{\sup RCN_b(K, \mathcal{G})}$. Suppose for $[x] \subseteq H$:*

1. $\overline{\overline{[x]} \cap L_m(G_i/_{[y]})} = \overline{[x]} \cap L(G_i/_{[y]})$

2. $\forall \sigma \in \Sigma_u : \big[\sigma \notin L(G/_{[y]})\big]$

Then $H - \big([x](H/_{[x]}) - [x]\big) \in RCN_b(K/_{[y]}, \mathcal{G}/_{[y]})$.

Proof: $H' = [x](H/_{[x]}) - [x]$ and $H'' = [x](\overline{H/_{[x]}}) - [x]$. First, we show $\overline{H - H'} = \overline{H} - H''$.

$$\begin{aligned}
\overline{H} - H'' &= [\overline{H - H'} \cup \overline{H'}] - ([x]\overline{H/_{[x]}} - [x]) \\
&= [\overline{H - H'} - ([x]\overline{H/_{[x]}} - [x])] \cup [\overline{H'} - ([x]\overline{H/_{[x]}} - [x])] \\
&= \overline{H - H'} \cup [\overline{H'} - ([x]\overline{H/_{[x]}} - [x])] \\
&\qquad (\overline{H - H'} \text{ and } ([x]\overline{H/_{[x]}} - [x]) \text{ are disjoint}) \\
&= \overline{H - H'} \cup [\overline{[x](H/_{[x]}) - [x]} - ([x]\overline{H/_{[x]}} - [x])] \\
&= \overline{H - H'} \cup [\big(\overline{[x]} \cup [x]\overline{H/_{[x]}}\big) - \big([x]\overline{H/_{[x]}} - [x]\big)] \\
&\qquad \text{(by Lemma 2.4)} \\
&= \overline{H - H'} \cup [\overline{[x]} \cup \big([x]\overline{H/_{[x]}} - ([x]\overline{H/_{[x]}} - [x])\big)] \\
&= \overline{H - H'} \cup \overline{[x]} \cup [x] \\
&= \overline{H - H'} \cup \overline{[x]} \\
&= \overline{H - H'}
\end{aligned}$$

To prove the $G_i/_{[y]}$ – nonblocking property, consider two different cases.

1. $[x] \notin L_m(G_i/_{[y]})$: By Lemma 2.5 we have:

$$\overline{(H - H') \cap L_m(G_i/_{[y]})} = \overline{(H \cap L_m(G_i/_{[y]})) - (H' \cap L_m(G_i/_{[y]}))}$$

Since, in this case $H' \cap L_m(G_i/_{[y]}) = \varnothing$ we conclude

$$\overline{(H - H') \cap L_m(G_i/_{[y]})} = \overline{H \cap L_m(G_i/_{[y]})}$$

Furthermore

$$\begin{aligned}
\overline{H - H'} \cap L(G_i/_{[y]}) &\subseteq \overline{H} \cap L(G_i/_{[y]}) \\
&= \overline{H \cap L_m(G_i/_{[y]})} \\
&= \overline{(H - H') \cap L_m(G_i/_{[y]})}
\end{aligned}$$

So, $\overline{H - H'} \cap L(G_i/_{[y]}) \subseteq \overline{(H - H') \cap L_m(G_i/_{[y]})}$. Since the reverse containment always holds, we have $\overline{H - H'} \cap L(G_i/_{[y]}) = \overline{(H - H') \cap L_m(G_i/_{[y]})}$.

2. $[x] \subseteq L_m(G_i)/_{[y]}$: Therefore $t \in L(G_i/_{[y]})$. Now, consider:

$$\begin{aligned}
(\overline{H - H'}) \cap L(G_i/_{[y]}) &= \big(\overline{H} - ([x]\overline{H/_{[x]}} - t)\big) \cap L(G_i/_{[y]}) \\
&= \big(\overline{H} \cap L(G_i/_{[y]})\big) - \big(([x]\overline{H/_{[x]}} - [x]) \cap L(G_i/_{[y]})\big) \\
&= \big(\overline{H} \cap L(G_i/_{[y]})\big) - \Delta L_L \qquad (4.1)
\end{aligned}$$

where $\Delta L_L := ([x]\overline{H/[x]} - [x]) \cap L(G_i/_{[y]})$. Next, let $R = \overline{(H - H') \cap L_m(G_i/_{[y]})}$ and consider:

$$\begin{aligned}
\overline{H \cap L_m(G_i/_{[y]})} &= R \cup \overline{H' \cap L_m(G_i/_{[y]})} \\
&= R \cup \overline{([x](H/_{[x]}) - [x]) \cap L_m(G_i/_{[y]})} \\
&= R \cup \overline{[x](H/_{[x]} - \epsilon) \cap L_m(G_i/_{[y]})} \\
&= R \cup \big([x]\overline{(H/_{[x]} - \epsilon) \cap L_m(G_i/_{[y][x]})} \cup \overline{t}\big) \\
&= R \cup [x]\overline{H/_{[x]} \cap L_m(G_i/_{[y][x]})} \quad \big(\text{since } \overline{[x]} \subseteq R\big) \\
&= R \uplus \big([y]\overline{H/_{[x]} \cap L_m(G_i/_{[y][x]})} - [x]\big)
\end{aligned}$$

Let $\Delta L_R = \big([x]\overline{H/_{[x]} \cap L_m(G_i/_{[y][x]})} - [x]\big)$, then:

$$\overline{(H - H') \cap L_m(G_i/_{[x]})} = \overline{H \cap L_m(G_i/_{[x]})} - \Delta L_R \tag{4.2}$$

Note that:

$$\begin{aligned}
\Delta L_R &= [x]\overline{H/_{[x]} \cap L_m(G_i/_{[y][x]})} - [x] \\
&= [x]\overline{H \cap L_m(G_i/_{[y]})}/_{[}x] - [x] \quad \text{(by Lemma 2.1)} \\
&= [x](\overline{H} \cap L(G_i/_{[y]}))/[x] - [x] \quad (\text{H is } L_m(G_i/_{[y]})\text{-closed}) \\
&= [x](\overline{H/_{[x]}} \cap L(G_i/_{[y][x]})) - [x] \quad \text{(by Lemma 2.1)} \\
&= \big([x]\overline{H/_{[x]}} \cap L(G_i/_{[y]})\big) - [x] \\
&= ([x]\overline{H/_{[x]}} - [x]) \cap L(G_i/_{[y]}) \\
&= \Delta L_L
\end{aligned}$$

From (4.1), (4.2), and $G_i/_{[y]}$-nonblocking property of H it follows that $H - H'$ is $G_i/_{[y]}$ – nonblocking.

For controllability we need to ensure that the behavior of $L(G/_{[y]})$ can be limited to $\overline{H - H'}$. This goal can be simply achieved by disabling the events after the state x which we know are all controllable. Finally, for relative-closure:

$$\begin{aligned}
\overline{H - H'} \cap L_m(G/_{[y]}) &= \big(\overline{H} - ([x]\overline{H/_{[x]}} - [x])\big) \cap L_m(G/_{[y]}) \\
&= \big(\overline{H} \cap L_m(G/_{[y]})\big) - \big(([x]\overline{H/[x]} - [x]t) \cap L_m(G/_{[y]})\big) \\
&= H - \big(([x]\overline{H/_{[x]}} \cap L_m(G/_{[y]})) - [x]\big) \ \big(\text{H is } L_m(G)\text{-closed}\big)
\end{aligned}$$

$$= \quad H - ([x](\overline{H/_{[x]}} \cap L_m(G/_{[y][x]})) - [x]) \quad \left(\overline{H}/_{[x]} \cap L_m(G_i/_{[y]})/_{[x]} = H/_{[x]}\right)$$

$$= \quad H - ([x](H/_{[x]}) - x)$$

$$= \quad H - H'$$

This completes the proof.

■

Finally the next theorem formalizes a set of sufficient conditions that guarantees the RLL-S supervisor is maximally permissive.

Theorem 4.2. Assume $\overline{H} = \overline{H_{nb}}$ and that there is no SE in $L(G, \gamma^N)$. If $N \geq N_B$ then $L(G, \gamma^N_{RLL-S}) = \overline{\sup RCN_b(K, \mathcal{G})}$.

Proof: To prove the theorem we start with the automaton $H := \sup RCN_b(K/_{[x]}, \mathcal{G}/_{[x]})$ and then construct H' with the following procedure:

- Expand H' as a subgraph of H.
- Stop the expansion in every state $x \in X_{nb}$.

This is equivalent to removing all strings after a nonblocking controllable state. According to Lemma 4.7, the language represented by H' remains $\mathcal{G}_i/_{[x]}$-nonblocking, controllable and relative-closed with respect to $G/_{[x]}$.

We now claim that the expansion of H' terminates in $N_B - 1$ steps. Since $\overline{H} = \overline{H_{nb}}$ we conclude that all states in H are coreachable with respect to states in H_{nb}. If this was not the case then the condition $\overline{H} = \overline{H_{nb}}$ would be violated. The distance between every state and a nonblocking state $x \in H_{nb}$ is bounded by $N_B - 1$ and we know that the expansion terminates in all branches. Consequently, this termination happens the worst case in $N_B - 1$ steps. This means that:

$$H' = H'/_{[x]} = H'/_{[x]}|_{N_B-1} = H'/_{[x]}|_{N-1} \subseteq K/_{[x]}$$

Next we show that

$$L(H'|_1) = L(H|_1)$$

Obviously, $L(H'|_1) \subseteq L(H|_1)$ so we only need to show the reverse containment. Suppose $t \in L(H|_1)$ and therefore $|t| = 1$. Also let $\delta(t, x) = x'$ where $x' \in X_H$. If x' is a nonblocking controllable state, then $t \in L(H')$ and consequently $L(H'|_1) = L(H|_1)$. If x' is not a nonblocking controllable state then there exists a t' such that $\delta(x', t') = x''$ such that x'' is nonblocking controllable. Again this means $t \in L(H')$ and thus $L(H'|_1) = L(H|_1)$. Finally similar to Lemma 4.2 we can show that:

$$\begin{aligned} H' &= \sup RCN_b(H', \mathcal{G}/_{[x]}) && \text{(by Lemma 4.7)} \\ &= \sup RCN_b(H', \mathcal{G}/_{[x]}|_N) && \text{(Proof similar to Lemma 4.2)} \end{aligned}$$

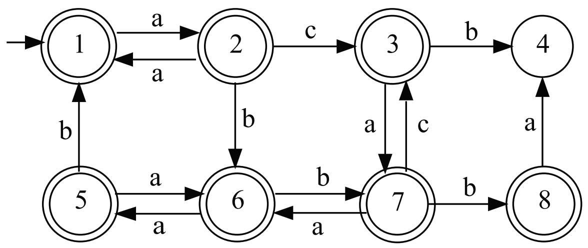
■

Remark 4.1. Let $E = \sup RCN_b(K, \mathcal{G})$. The condition $\overline{H} = \overline{H_{nb}}$ can be replaced by $\overline{E} = \overline{E_{nb}}$ as we only remove the post-language of nonblocking states in $\sup RCN_b(K_{[x]}, \mathcal{G}/_{[x]})$. Thus, it is not necessary for all states in K to be coreachable to with respect to a nonblocking state.

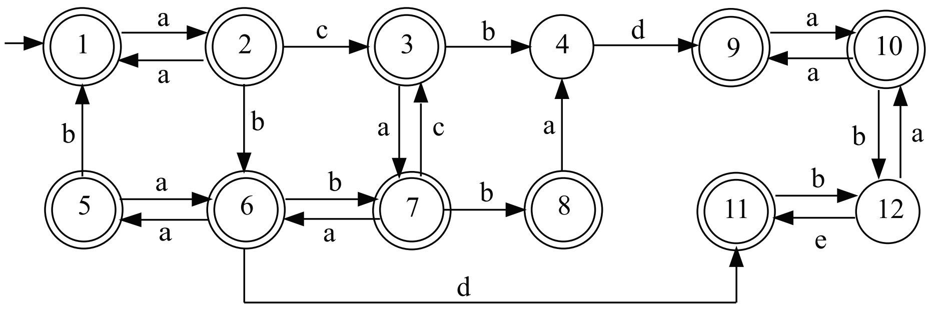
4.3 Illustrative Example

In this section we present a simple example in order to clarify the presented results.

Example 4.3. Consider two mutually refined automata $\mathcal{G} = \{G_1, G_2\}$ as shown in Figure 4.8 with $\Sigma_c = \{a, b, c\}$, $\Sigma_{uc} = \{d, e\}$, and the illegal states 3, 10. The marked states in Figure 4.8 are shown by double circles.



(a) G1



(b) G2

Figure 4.8: Example 4.3: Plant automata.

In the following we first determine the off-line solution to the problem.

4.3.1 Off-line Solution

First we consider the union model of the plant which is equal to G_2 in this problem ($G = G_2$). The overall specification, H, is obtained by removing illegal states and all the attached edges and it is shown in Figure 4.9.

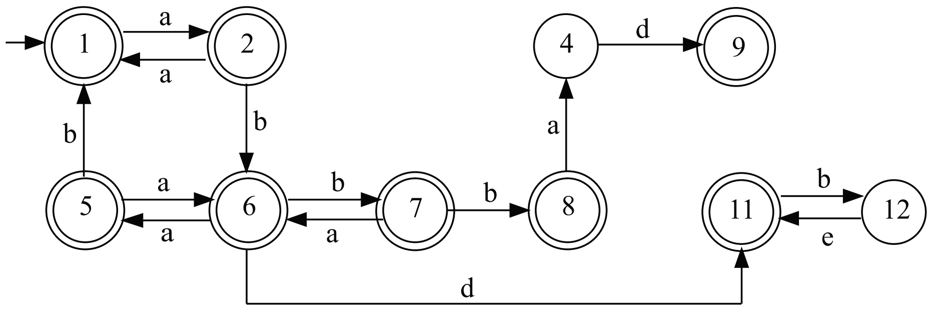


Figure 4.9: Example 4.3: The overall specification H.

We then solve the problem based on the algorithm presented in [23] to find $\sup RCN_b(K, \mathcal{G})$. For this purpose we use the MATLAB function $supRCN$ which is listed in Appendix B. The supervisor can be realized by the automaton shown in Figure 4.10.

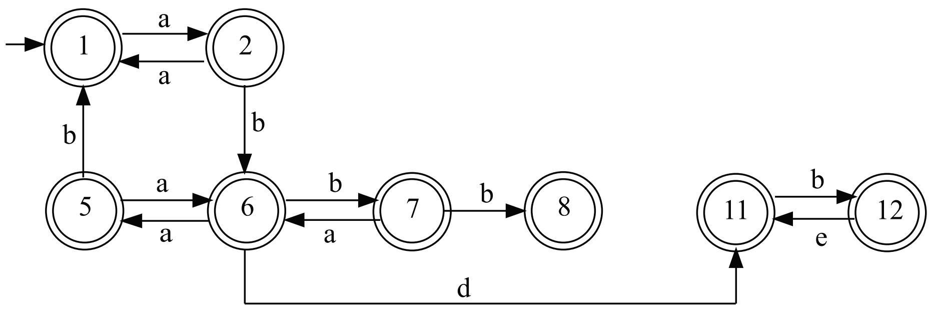


Figure 4.10: Example 4.3: Realization of off-line state-based supervisor.

Next we work out to find the online solution to the problem.

4.3.2 Online Solution

The focus of this example is on finding the maximally permissive RLL-S supervisor. With this purpose, we first determine N_B.

Length of Lookahead Window

To determine N_B we have to determine H_{nb} and $w(x, s)$ for every $x \in X_H$ and $s \in L(H)/_{[x]}$. The bolded states in Figure 4.11 characterize H_{nb}.

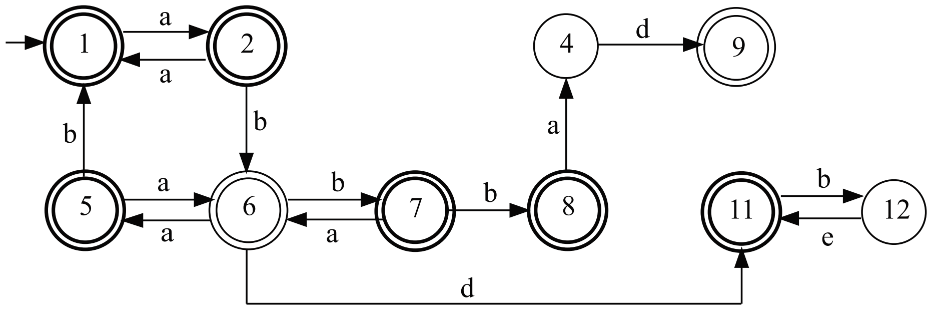


Figure 4.11: H_{nb}: Example 4.3

Next, we form $w(x, s)$ for all $x \in X_H$ and $s \in L(H)/_{[x]}$ that $w(x, s)$ is nonempty.

$$w(1, a) = \{1, 2\}$$

$$w(2, ba) = \{2, 5, 6\}, w(2, bb) = \{2, 6, 7\}, w(2, bd) = \{2, 6, 11\}$$

$$w(5, b) = \{1, 5\}, w(5, ab) = \{5, 6, 7\}, w(5, ad) = \{5, 6, 11\}$$

$$w(6, b) = \{6, 7\}, w(6, a) = \{5, 6\}, w(6, d) = \{6, 11\}$$

$$w(7, b) = \{7, 8\}, w(7, aa) = \{5, 6, 7\}, w(6, d) = \{6, 11\}$$

$$w(11, be) = \{11, 12\}$$

Having the sets $w(x, s)$ we find $N_B = 3$ which is the required window size for optimality of RLL-S supervisor.

Control Action Computation

According to Theorem 4.2 and Remark 4.1 we expect $L(G, \gamma^3)$ to be maximally permissive. We investigate the performance of the supervisor along the path abb. Starting from $s = \epsilon$, both models are compatible with the *currently executed trace*; the output of $f_{\Uparrow}^{N}$ block is shown in Figure 4.12.

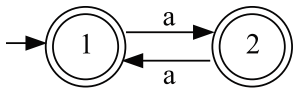


Figure 4.12: $f_N(\epsilon)$: Example 4.3

So, the control action is $\gamma^3(\epsilon) = \{a\}$. After executing event a, with $s = a$ and $x = 2$, the control action is $\gamma^3(a) = \{a, b\}$, and the output of block $f_{\Uparrow}^N$ is as Figure 4.13. One can see that executing further events, and going ahead through the lookahead window causes new events to be enabled. The control action following string $s = bc$ is $\gamma^3(bc) = \{a\}$.

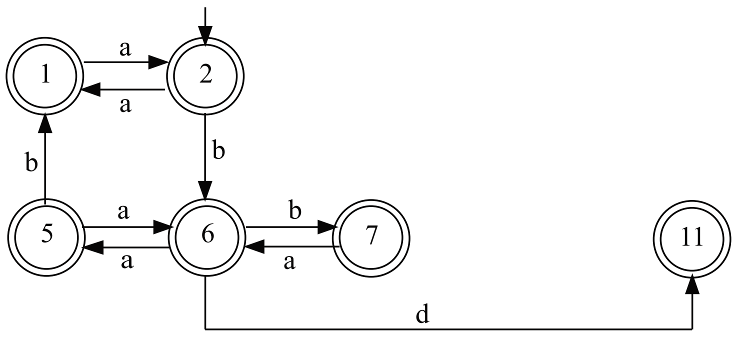


Figure 4.13: $f_N(a)$: Example 4.3

Continuing with execution of the events the next control action would be $\gamma^3(ab) = \{a, b, d\}$. This shows that by choosing the window size long enough, we can achieve $L(G, \gamma^N) = \sup RCN_b(K, \mathcal{G})$.

4.4 Conclusion

In this chapter, the state-based RNSCP problem in which the specifications are in terms of legal/illegal states and the supervisor is a state feedback control law has been studied. State-based RLL (RLL-S) supervisor has been proposed as a dual of the linguistic approach presented in Chapter 3. Furthermore, a set of conditions under which the RLL-S supervisor is maximally permissive has been provided. In the next chapter, the proposed online RLL-S supervision will be applied to the robust control of spacecraft propulsion systems.

Chapter 5

Application Examples

Robustness of a spacecraft is an important criterion due to the long time operation of the spacecraft with no maintenance in between. It is essential to take a robust, fault-tolerant control strategy in case of designing an autonomous spacecraft. In this section we consider the propulsion systems of two spacecraft: Viking and Cassini. The systems studied here are subject to some simplifications.

Spacecraft are subject to numerous failures, which means that off-line supervisor requires a large amount of memory for implementation onboard. We apply the proposed online RLL-S supervision for robust control of the spacecraft propulsion systems.

5.1 Viking Spacecraft

The simplified propulsion system of Viking orbiter is depicted in Figure 5.1 [35]. It consists of a high pressure Helium tank, a pyro-ladder consisting five pyro-valves (PV_1, PV_3, PV_5 are normally closed; PV_2 and PV_4 are normalcy open), a regulator valve to pressurize fuel tanks, and an engine usable for trajectory correction and orbiter insertion maneuvers. The pyro-valves can operate only once, meaning that if a normally-closed (resp. normally-open) turns open (resp. turned close), it remains open (resp. closed) for the rest of mission and the action is not reversible. In Figure 5.1, the pyro-valves in solid black are normally closed, while the others are normally open.

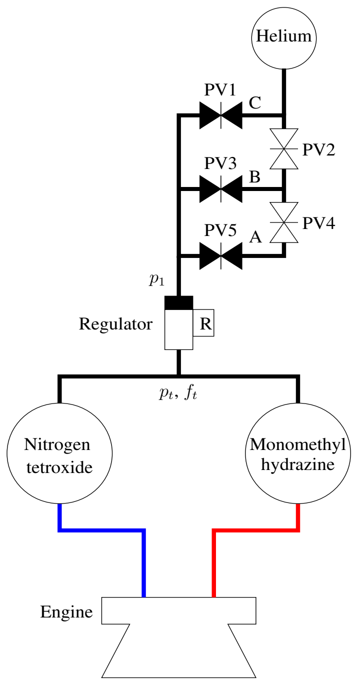


Figure 5.1: Simplified pressure control assembly of Viking orbiter.

As Viking 1 was approaching to the Mars, a major anomaly was detected in its pressure control assembly. On June 7, 1976, PV_3 was opened to pressurize the fuel tanks for the Approach Course Manoeuver (ACM). However, the tanks pressure continued to rise after the regulator threshold was reached due to regulator leakage. The obvious solution was to fire PV_2 in order to cut-off the pressure from the regulator. This would leave PV_1 as the only path for pressurizing the tanks prior to Mars Orbit Insertion (MOI). If PV_1 failed to open (get stock closed), the whole mission would be lost because of low fuel pressure [37].

The adopted solution, which was uplinked to the spacecraft from ground station, was to split ACM into two parts. Thus they could keep the tanks' pressures below rapture pressure during the 12 days remaining to MOI. Such decisions cannot be made autonomously and they must be made by human operators.

The Viking's design did not have enough redundancy for onboard autonomous decisions. The design was later modified in Mars Global Surveyor (MGS) spacecraft [35] ,[40]. The schematic diagram of MGS propulsion system is shown in Figure A.1. A simplified version of the design is shown in Figure 5.2.

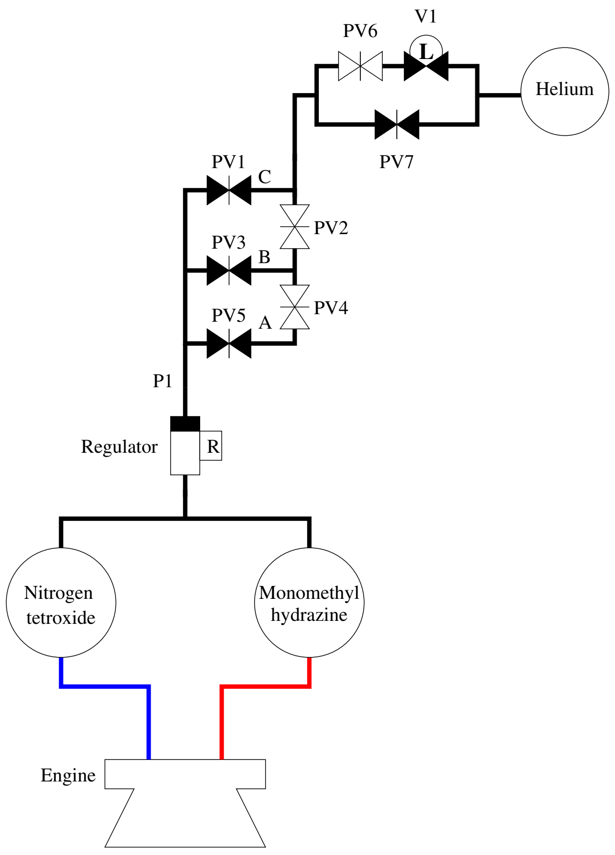


Figure 5.2: Simplified MGS propulsion system

The new design has enough redundancy for single-failure scenarios. In this section we aim to design a supervisor which handles the the same situation happened for Viking spacecraft. This means that at the start of the ACM, PV_3, PV_4 and PV_5 are already fired, V_1 is open, and the regulator is leaking. In the next section, we find the DES model of the system.

5.1.1 Discrete Event Modeling

In this section we present DES model of different components and the interactions between them. Then we construct the complete model of the system by combining the components and interactions. Table 5.1 describes all the events which appears in the modeling. For simplicity we assume that only V_1 and PV_1 are subject to failure.

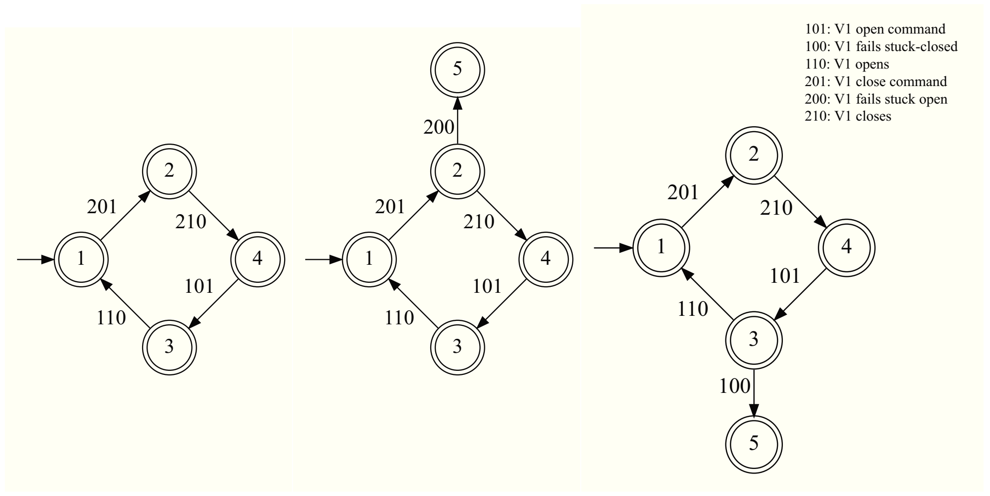
Event	Tag	Description
101	$V_{1,OC}$	Valve 1 open command (controllable)
100	$V_{1,SC}$	Valve 1 fails stuck-closed (uncontrollable)
110	$V_{1,O}$	Valve 1 opens (uncontrollable)
201	$V_{1,CC}$	Valve 1 close command (controllable)
200	$V_{1,SO}$	Valve 1 fails stuck-open (uncontrollable)
210	$V_{1,C}$	Valve 1 closes (uncontrollable)
10	$PV_{1,OC}$	Pyro-valve 1 open command (controllable)
11	$PV_{1,O}$	Pyro-valve 1 opens (uncontrollable)
12	$PV_{1,SC}$	Pyro-valve 1 fails stuck-closed (uncontrollable)
$i1$	$PV_{i,OC}$	Pyro-valve *i* open command, i = $3, 5, 7$ (controllable)
$i2$	$PV_{i,O}$	Pyro-valve *i* opens, i = $3, 5, 7$ (uncontrollable)
$i0$	$PV_{i,CC}$	Pyro-valve *i* close command, i = $2, 4, 6$ (controllable)
$i1$	$PV_{i,C}$	Pyro-valve *i* closes, i = $2, 4, 6$ (uncontrollable)
1000	$P_{1,L}$	P_1 goes down (uncontrollable)
1001	$P_{1,H}$	P_1 goes high (uncontrollable)

Table 5.1: Event list for Viking propulsion system

The DES models for normal and normal-faulty modes of V_1 are shown in the following following figure.

The valve is prone to two different failures (stuck-open and stuck-closed). Thus, we would have two different normal-faulty models as depicted in Figure 5.3.

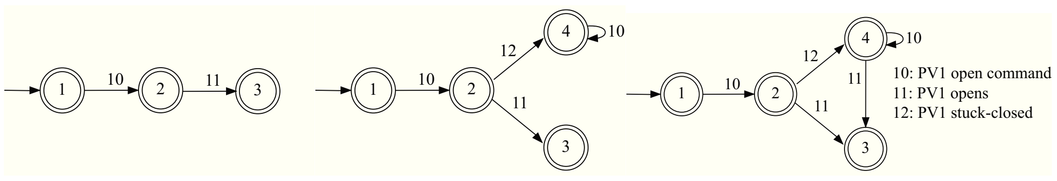
The next model is for the pyro-valves. For simplicity, we assume that only PV_1 is subject to failure. Once PV_1 is commanded open, it may turn-on or it may get stuck-closed. If the valve becomes stuck, there is still a chance of operation by issuing more open commands. This means that PV_1 has three possible DES models: 1- the normal model with no failure event 2- normal-faulty model, where the valve fails stuck-closed and it remains stuck-closed (further commands are not effective) 3- normal-recovery model,



(a) $V1_N$ (b) $V1_{NF_1}$ (c) $V1_{NF_2}$

Figure 5.3: DES models of isolation valve V_1

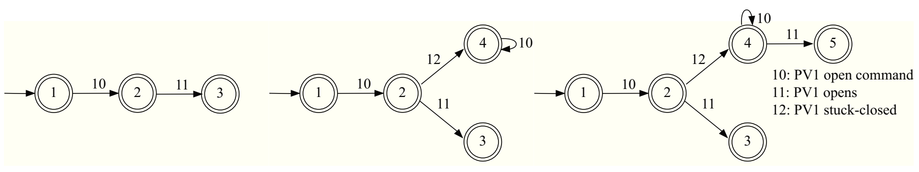
where the valve fails after issuing the first command but it turns-on after issuing more open commands (has a recovery option). The DES models for PV_1 are show in Figure 5.4.



(a) $PV1_N$ (b) $PV1_{NF_1}$ (c) $PV1_{NF_1R}$

Figure 5.4: DES models of pyro-valve PV_1

Remark 5.1. In order to conform to the state-domain control requirements normal and normal-faulty models of every components need to be mutually refined. Thus, the DES models of PV_1 need to be modified as shown in Figure 5.5.



(a) $PV1_N$ (b) $PV1_{NF_1}$ (c) $PV1_{NF_1R}$

Figure 5.5: DES models of pyro-valve PV_1

For the rest of pyro-valves, the DES model are similar to $PV1_N$ and are shown in Figure 5.6.

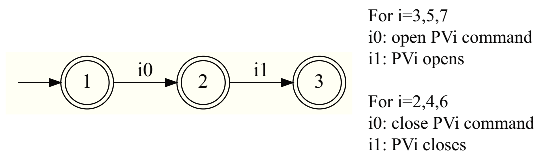


Figure 5.6: PV_i: DES model of pyro-valve $i = 2, 3, .., 7$

Finally, we need the DES model of pressure sensor P_1 as shown in Figure 5.7. Note that the pressure is initially high.

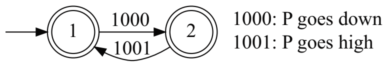


Figure 5.7: P_1: DES model of pressure sensor

Now that we have the DES models of the components, the next step is to model the interaction between them. In this example, pressure P_1 depends on the state of the valves. Pressure P_1 is high if and only if:

- PV_1 AND PV_7 are open,
- OR PV_1 AND V_1 AND PV_6 are open,
- OR PV_2 AND PV_7 are open,
- OR PV_2 AND V_1 AND PV_6 are open,

This can be described as a DES model obtained from sync operation of $sync(V_1, PV_1, PV_2, PV_6, PV_7)$ and then adding self-loops of appropriate pressure events to each state. The obtained DES has $6 \times 4 \times 3^3 = 648$ states. We refer to the DES model of the interactions as *INT*.

Lastly, we should determine the set of marked states in our model. We would like to bring down the pressure P_1 (in the upstream of the regulator), so we prevent the regulator from leaking (this keeps tank pressures below the rapture pressure). Then we have to bring up the pressure P_1 and keep it high in order to pressurize the tanks for MOI maneuver. Thus the following automaton presents the set marked strings in this problem. To determine the appropriate marking, it would be enough to sync the automaton with DES model of the other components and interactions.

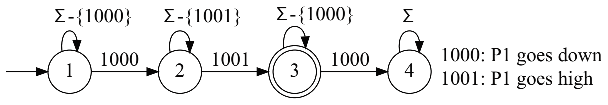


Figure 5.8: M: DES model of the appropriate marking

Considering a single-failure scenario the set of possible models for RNSCP will be as follows. The pair of (# states,# transitions) is shown for each model. The union model has 2592 states and 11394 transitions.

$$
\begin{aligned}
G_N &= sync(V1_N, PV1_N, PV_2, ..., PV_7, P, INT, M) && (1296, 6556)\\
G_{NF_1} &= sync(V1_N, PV1_{NF_1}, PV_2, PV_6, PV_7, P, INT, M) && (1728, 7596)\\
G_{NF_1R} &= sync(V1_N, PV1_{NF_1R}, PV_2, PV_6, PV_7, P, INT, M) && (1728, 8028)\\
G_{NF_2} &= sync(V1_{NF_1}, PV1_N, PV_2, PV_6, PV_7, P, INT, M) && (1944, 9834)\\
G_{NF_3} &= sync(V1_{NF_2}, PV1_N, PV_2, PV_6, PV_7, P, INT, M) && (1944, 9834)
\end{aligned}
$$

5.1.2 Design Specification

The design specification does not identify any certain state as illegal in the plant model. Thus, the main objective is to ensure that the system under supervision is nonblocking. The set of marked states includes the states that reach after bringing the pressure P_1 down and then bringing and keeping it up again. This corresponds to completion of the task.

5.1.3 Robust Formulation of the Problem

As described in Chapter 1, fault recovery problem can modeled as a RNSCP. In fault recovery problem, we aim to design a robust supervisor which guarantees that the system under supervision meets the safety requirements and it is nonblocking in both normal and failure modes.

If we apply conventional (non-robust) supervisory control to this problem, pyro-valve PV_2 will be allowed to close in order to bring down the pressure P_1 and stop regulator leakage. However, if PV_1 fails to open, then the system under supervision of non-robust supervisor may reach a deadlock; since the recovery event in PV_1 is not forcible. As a result of the failure of PV_1 failure P_1 can not rise anymore and a deadlock happens in the failure mode. This show that non-robust supervisory control is not able to meet nonblocking requirement and thus, the problem must be formulated in the robust control framework.

Problem formulation: Let $\mathcal{G} = \{G_N, G_{NF_1}, ..., G_{NF_4}\}$ be the set of normal and normal-failure models of MGS propulsion system. $G_{union} = \bigcup G_i$ with $G_i \in \mathcal{G}$. Assuming that all the states are legal, the overall specification is $K = L_m(G_{union})$. Design a maximally permissive nonblocking state-based supervisor such that $L(G_N, \gamma) = \overline{L_m(G_N, \gamma)}$ and $L(G_{NF_i}, \gamma) = \overline{L_m(G_{NF_i}, \gamma)}$ for $i = 1, ..., 3$ and $L(G_{NF_1R}, \gamma) = \overline{L_m(G_{NF_1R}, \gamma)}$. Note that the plant models are mutually refined. ■

5.1.4 Off-line Solution

In this section, we summarize the procedure for the computation of off-line robust supervisor. The process which is presented here is developed in MATLAB environment using the DECK toolbox [43]. The implemented procedures are presented in Appendix B.

1. After building DES model of every component, the problem is setup by constructing the set of possible models $\mathcal{G} = \{G_N, G_{NF_1}, ..., G_{NF_3}\}$. The overall specification K is set equal to the union plant model since there is no illegal state.

2. Using the procedure $E = \sup RCN(K, G_{union}, \Sigma_u, G_N, G_{NF_1}, ..., G_{NF_3})$, we find the *supremal relative-closed*, *controllable G_i-nonblocking* part of K.

3. The off-line supervisor is characterized by the closed-behavior of the automaton E. The resulting off-line supervisor has 763 states and 2361 transitions.

We use off-line solution to validate maximal permissiveness of the corresponding RLL-S supervisor.

5.1.5 Online Solution

In order to design a maximally permissive RLL-S supervisor we have to (i) First, determine N_B which is the required length of lookahead window and (ii) verify the condition of Theorem 4.2 and Remark 4.1. Here we present the functions which are developed for solving the online problem.

- In the first step, we determine H_{nb} as defined in Chapter 4. This is to be done using the function *detKnb($H, \Sigma_u, G_N, G_{NF_1}, ..., G_{NF_4}$)*. The function examines every marked controllable state in *H* to determine the set of nonblocking states.

- Next, the output of *detKnb(H_{nb}, Σ_u)* is used to determine N_B through *detNSB* function. *detNSB* determines the maximum distance between every two neighboring nonblocking states (two states which are nonblocking and are connected through a sequence of events. The event sequence does not go through any nonblocking states in between). In this example N_B is determined to be 10.

- Next, the maximum distance between every two states of G_{union} is determined. This is accomplished using *expandLLSB(G_{union}, S, N)* function. The function expands G_{union} (as a subgraph) from every state S and for various lengths of lookahead window, N. The maximum of required length for discovering the farthest state is equal to N_{max}. Comparing N_B with N_{max} is an indication of the effectiveness of online supervision. In this example $N_{max} = 21$.

- Having the set of nonblocking states, we verify the condition in Remark 4.1. In other words, we verify that every state in the off-line supervisor coreachable with respect to a nonblocking state.

- Finally, we simulate the plant under supervision of RLL-S supervisor (γ^{10}) using $RLL(G_{union}, K, \Sigma_u, N_B, G_N, G_{NF_1}, ..., G_{NF_4})$ function. The function prompts a set of *enabled events* after execution of each event and asks the user to select the next event in the sequence.

To study the performance of the designed RLL-S supervisor, we monitor the set of enabled events by the supervisor along two sample paths. Each path contains a different failure for two different single-failure scenarios.

Path 1:

The first selected path is the event sequence $S_1 = 201 - 200 - 60 - 61 - 1000 - 70 - 71 - 1001$ which contains the failure V_1 stuck open. The enabled/disabled events by the RLL-S supervisor, γ^{10}, is shown in Figure 5.9. In each step, the enabled events are shown by solid line while disabled events are shown using dashed line. We observe the RLL-S supervisor disables the events 20 and 70 in the normal mode. The events are later enabled when V_1 gets stuck open.

Path 2:

The second selected path is the event sequence $S_2 = 201 - 210 - 1000 - 10 - 12 - 11 - 70 - 71$ which contains the failure PV_1 stuck closed. The enabled/disabled events by the RLL-S supervisor, γ^{10}, is shown in Figure 5.10. We observe the RLL-S supervisor disables the event 20 in the normal mode. This means that the supervisor prevents does not allow PV_2 to get closed as there is a risk of failure in PV_1.

Finally, we investigate the size of the expansions which are encountered during online supervision. With this purpose, we construct subgraph expansions of the union model and the overall specification, starting from every state. The obtained information is shown in Table 5.2. We observe that the online supervisor is capable of reducing computational complexity associated with off-line supervision. Specifically, the off-line problem with 3240 states and 14364 transitions has been reduced to an online problem with an average size 200 states and 700 transitions.

Table 5.2: Average sizes of expanded automata

Automata	# states	# transitions	# states in expansions			# transitions in expansions		
			min	avg	max	min	avg	max
G	3240	14634	1	200	2070	0	700	8282
K	3240	14634	1	189	1785	0	648	6938

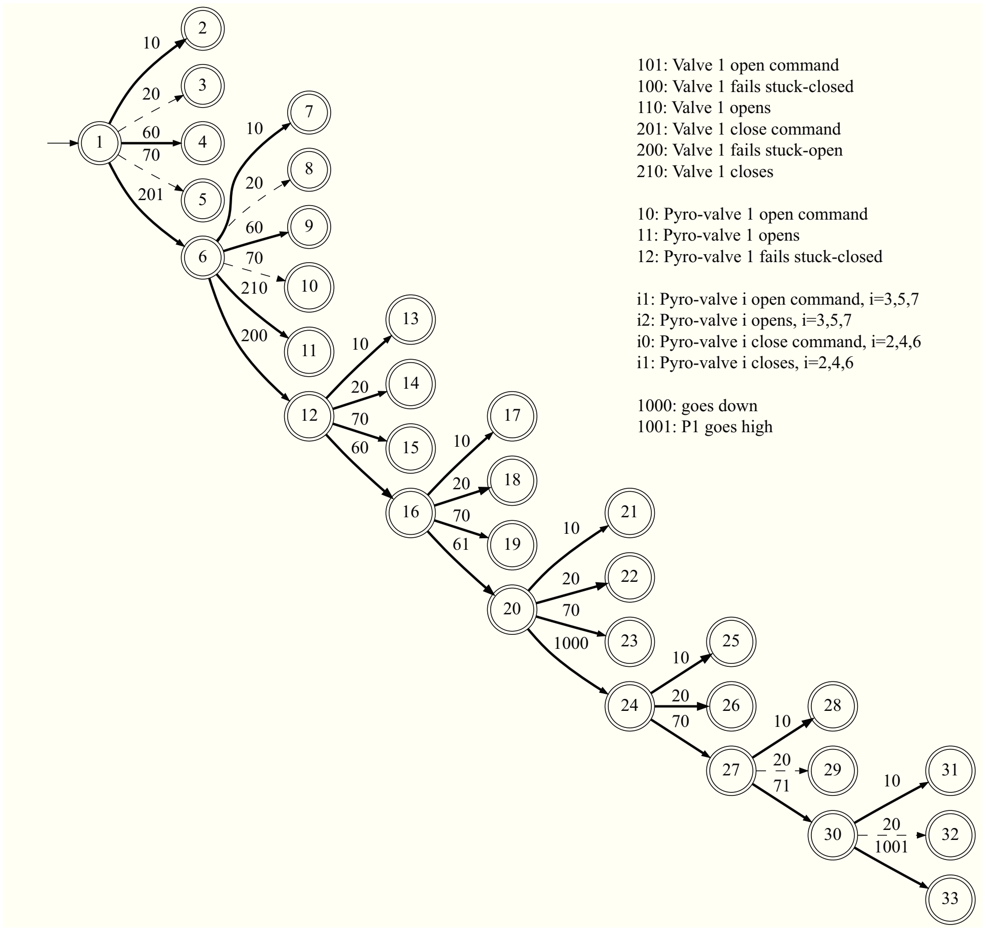

101: Valve 1 open command
100: Valve 1 fails stuck-closed
110: Valve 1 opens
201: Valve 1 close command
200: Valve 1 fails stuck-open
210: Valve 1 closes
10: Pyro-valve 1 open command
11: Pyro-valve 1 opens
12: Pyro-valve 1 fails stuck-closed
i1: Pyro-valve i open command, i=3,5,7
i2: Pyro-valve i opens, i=3,5,7
i0: Pyro-valve i close command, i=2,4,6
i1: Pyro-valve i closes, i=2,4,6
1000: goes down
1001: P1 goes high

Figure 5.9: Events enabled by γ^{10} along S_1: Viking spacecraft

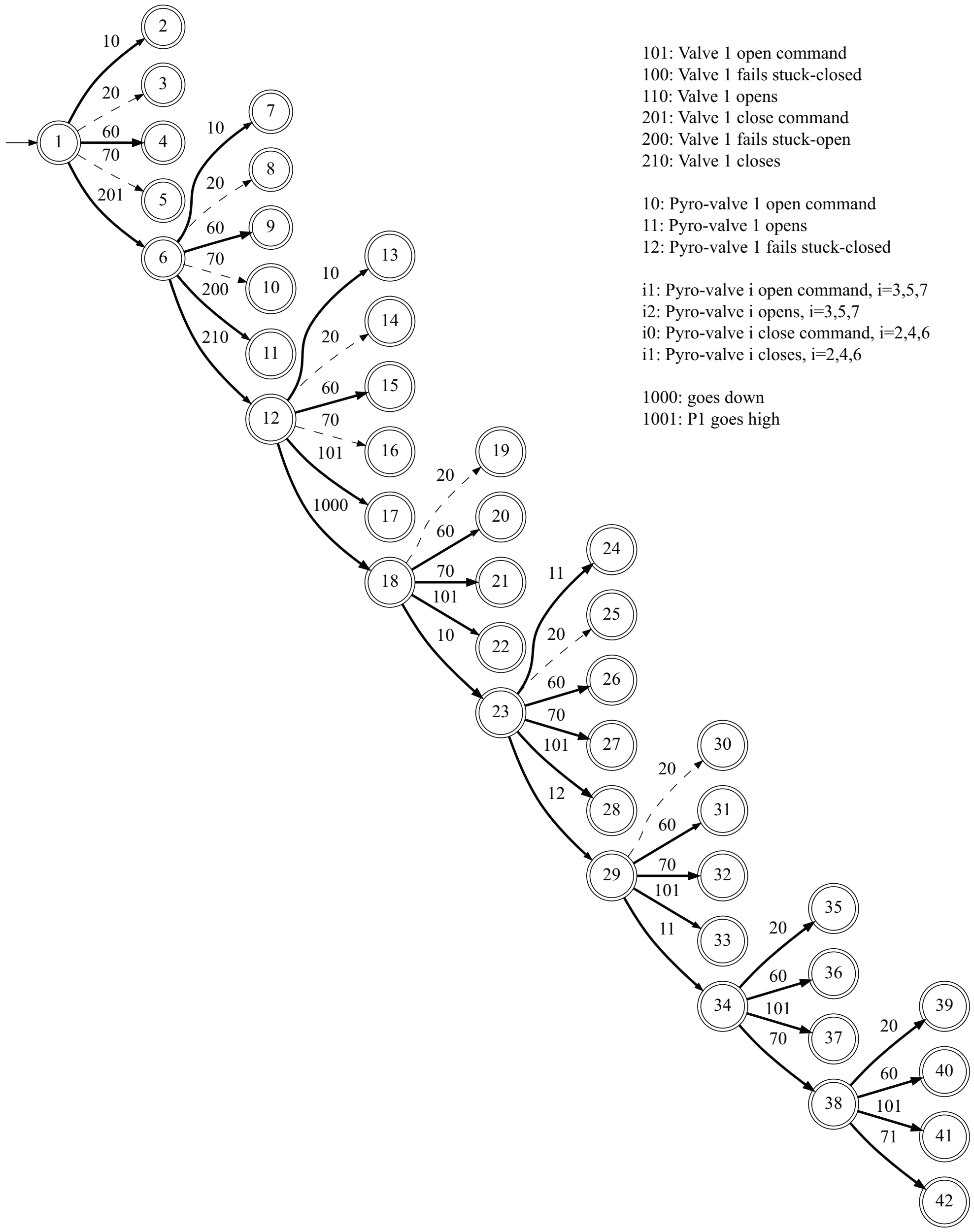

101: Valve 1 open command
100: Valve 1 fails stuck-closed
110: Valve 1 opens
201: Valve 1 close command
200: Valve 1 fails stuck-open
210: Valve 1 closes
10: Pyro-valve 1 open command
11: Pyro-valve 1 opens
12: Pyro-valve 1 fails stuck-closed
i1: Pyro-valve i open command, i=3,5,7
i2: Pyro-valve i opens, i=3,5,7
i0: Pyro-valve i close command, i=2,4,6
i1: Pyro-valve i closes, i=2,4,6
1000: goes down
1001: P1 goes high

Figure 5.10: Events enabled by γ^{10} along S_2: Viking spacecraft

5.2 Cassini Spacecraft

In this example, we consider a simplified version of the propulsion system of Cassini spacecraft propulsion system. The Cassini spacecraft is equipped with a bipropellant propulsion system with a high number of redundant parts. Figure A.2 shows a P&ID map of the propulsion system. A schematic of the CMPS (not simplified) is presented in Figure 5.11.

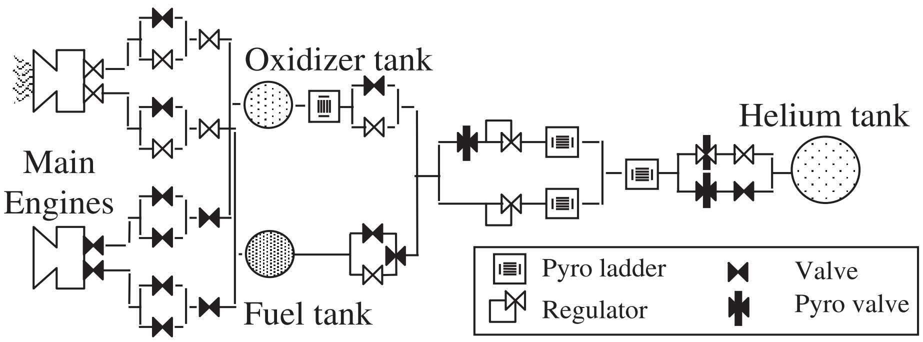


Figure 5.11: Cassini Main Propulsion System. [33]

Referring to Figure A.2, CMPS consists of two major subsystems (i) Pressure Control Assembly (PCA) which controls the pressure in propellant tanks (ensuring that the pressure remains in the normal range) and (ii) Pressure Isolation Assembly (PIA) which isolates the tanks pressures from the engines. In this example we consider the design of a supervisor for the PIA, assuming that the tanks have enough pressure to provide fuel to the engines while maintaining safety requirements. This system is shown in Figure 5.12.

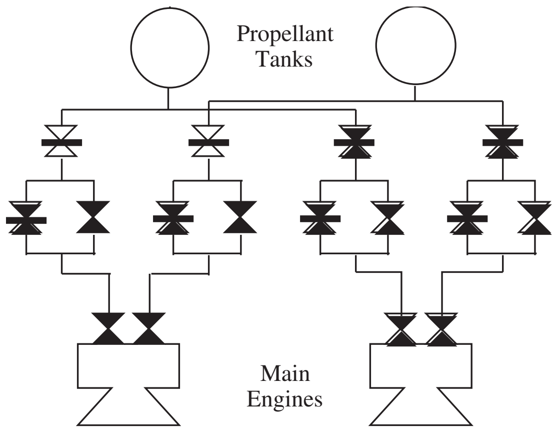


Figure 5.12: Cassini Pressure Isolation Assembly. [34]

For simplicity, we do not consider the coordination between the assemblies of the fuel and oxidizer and reduce the model to a mono-propellant propulsion system. The schematic diagram of the system studied here is shown in Figure 5.13. It consists of a fuel tank, two latch valves V_1 and V_2, four pyro-valves and two engines.

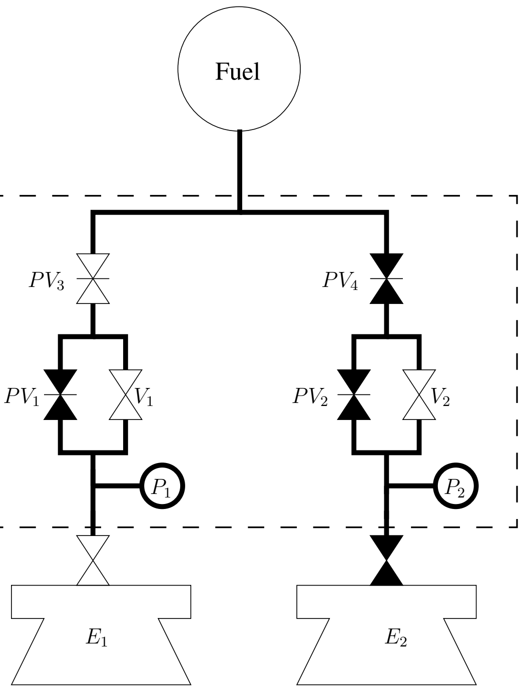


Figure 5.13: Simplified CMPS

The normal operation of the plant is as follows: initially the pyro-valves PV_1, PV_2 and PV_4 are closed. The pyro-valve PV_3 is normally open. The regular valves can be opened or closed in normal operation. The safety requirement is to avoid increasing the pressures P_1 and P_2 simultaneously as we never want to fire both engines together. Furthermore, we want to be able to switch between the engines if it is required. This means that we must be able to increase P_1 while decreasing P_2 and vice versa. Thus, the problem in normal mode is an RNSCP with Multiple set of Marked states (RNSCP-MM) [6].

The failure event f_1 (resp. f_2) is defined as the regular valve V_1 (resp. V_2) get stuck in closed position. In this failure recovery mode, the goal is to keep the pressure P_2 (resp. P_1) high while reducing P_1 (resp. P_2) to zero. This means that we want the engine in healthy side E_2 (resp. E_1) to be fully functional. The failure events and pressure sensor events are uncontrollable. Note that the events associated with pyro-valves are not reversible. Being an expensive spacecraft, CMPS has sufficient redundancy in the mission operation to stand multiple failures. However, we only consider single fault scenarios for simplicity. (It is worth mentioning that CMPS is still fully functional after 16 years of operation.)

5.2.1 Discrete Event Modeling

This section presents the DES models of the components and the interactions between them. The dynamics of the components are presented based on [35], [36], [38]. Table 5.3 describes the events which appearing in the models.

Event	Tag	Description
$i0$	V_{iC}	Valve *i* closes, $i = 1, 2$ (controllable)
$i1$	V_{iO}	Valve *i* opens, $i = 1, 2$ (controllable)
$i3$	V_{iSC}	Valve *i* fails stuck close, $i = 1, 2$ (uncontrollable)
$i11$	PV_{iF}	Pyro-valve *i* fires, $i = 1, 2, 3, 4$ (controllable)
$i00$	P_{iL}	Pressure *i* goes low, $i = 1, 2$ (uncontrollable)
$i01$	P_{iH}	Pressure *i* goes high, $i = 1, 2$ (uncontrollable)

Table 5.3: Event list for CMPS

The DES model for the regular valves are shown in the following figure. The valves are prone to failure in closed position. The failure events are assumed permanent and thus the corresponding valve never returns to normal mode.

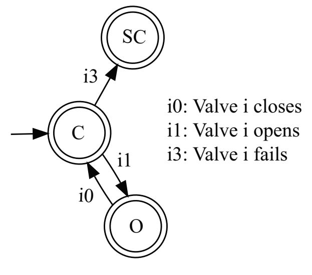
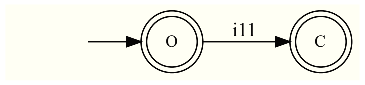
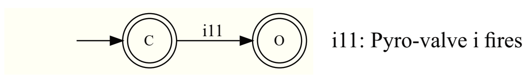


Figure 5.14: Vi_{NF}: Regular Valve (Normal-faulty mode)

The next model is for the pyro-valves assumed fault-free.



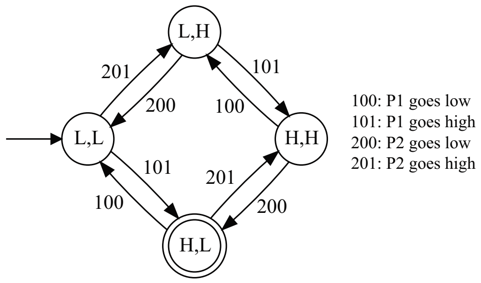
(a) Normally open pyro-valve



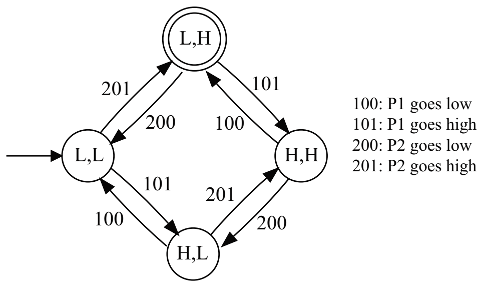
(b) Normally closed pyro-valve

Figure 5.15: PV_i: Pyro-Valves

Next, we model the sensor readings as a DES. The two different objectives on the pressures P_1 and P_2 require two different sets of marked states for sensor readings.



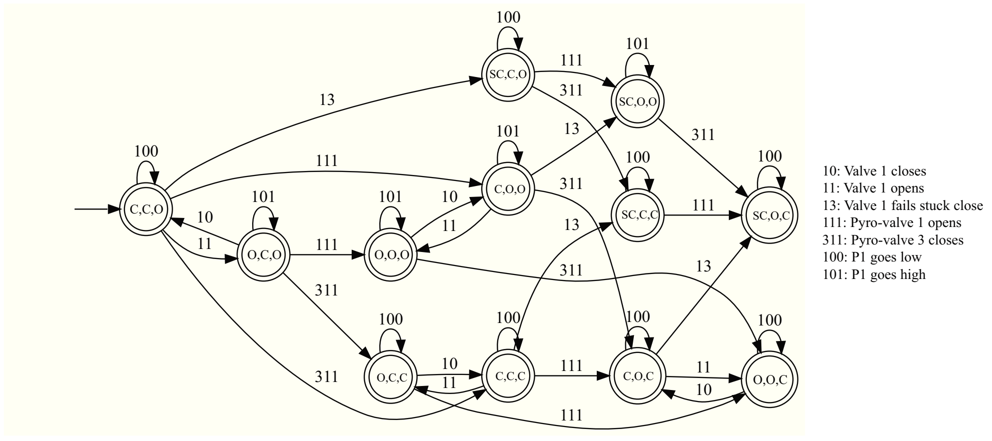
(a) P_{M_1}: P_1: high, P_2: low



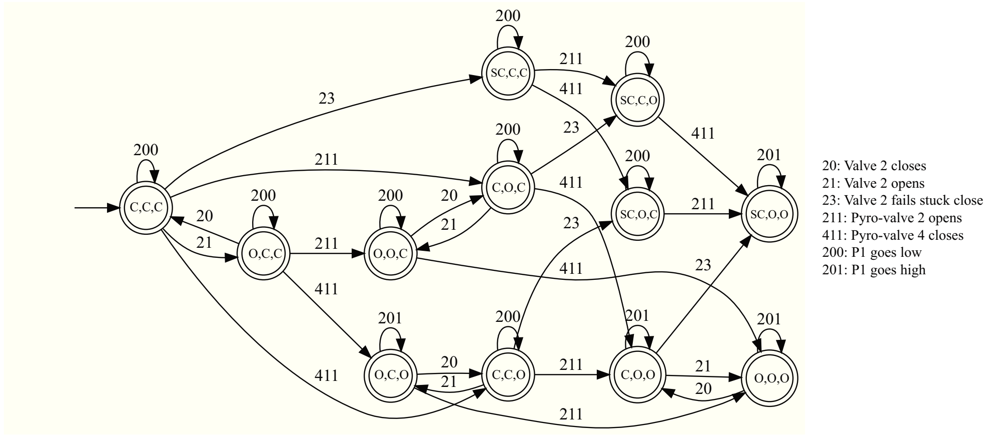
(b) P_{M_2}: P_1: low, P_2: high

Figure 5.16: Sensor reading with multiple set of marked states

Finally, we model the interaction the between state of the valves and the sensor readings. Pressure P_1 (resp. P_2) is high if and only if PV_3 (resp. PV_4) is open ***AND*** at least one of the PV_1 (resp. PV_2) ***OR*** V_1 (resp. V_2) is open. This can be described as the DES models which are shown in the Figure 5.17. The models have been obtained by adding self-loops of pressure events to $sync(V_1, PV_1, PV_3)$ (resp. $sync(V_2, PV_2, PV_3)$.



(a) $INT1_{NF}$: Interaction between P_1 and $sync(V_1, PV_1, PV_3)$ (Normal-faulty mode)



(b) $INT2_{NF}$: Interaction between P_2 and $sync(V_2, PV_2, PV_4)$ (Normal-faulty mode)

Figure 5.17: Interaction between pressures and valves status

We use DECK [43] to compute the possible plant models. In this problem, the set of possible models $\mathcal{G}$ contains two normal models (with different sets of marked states) and two normal-failure models. The models are:

$$\begin{aligned}
G_{\{N,1\}} &= sync(V1_N, V2_N, PV_1, PV_2, PV_3, PV_4, P_{M_1}, INT1_N, INT2_N)\\
G_{\{N,2\}} &= sync(V1_N, V2_N, PV_1, PV_2, PV_3, PV_4, P_{M_2}, INT1_N, INT2_N)\\
G_{\{NF,1\}} &= sync(V1_{NF}, V2_N, PV_1, PV_2, PV_3, PV_4, P_{M_2}, INT1_N, INT2_N)\\
G_{\{NF,2\}} &= sync(V1_N, V2_{NF}, PV_1, PV_2, PV_3, PV_4, P_{M_1}, INT1_N, INT2_N)
\end{aligned}$$

where normal models ($V1_N$, $V2_N$, $INT1_N$ and $INT2_N$) are obtained by removing the failure events from the DES model of components. $G_{\{N,1\}}$ and $G_{\{N,2\}}$ (the two normal models) contain 192 states and 896 transitions. $G_{\{NF,1\}}$ and $G_{\{NF,2\}}$ (the two normal-faulty models) contain 288 states and 1344 transitions. The marked states for $G_{\{N,1\}}$ and $G_{\{NF,2\}}$ (respectively $G_{\{N,2\}}$ and $G_{\{NF,1\}}$) are states where P_1 is high and P_2 is low (respectively P_2 is high and P_1 is low).

5.2.2 Design Specification

The safety specification is that P_1 and P_2 should not be high at the same time at it is a common specification for all the four plants. In addition, the closed-loop behavior should be nonblocking for both normal models ($G_{\{N,1\}}, G_{\{N,2\}}$) and normal-faulty models ($G_{\{NF,1\}}, G_{\{NF,2\}}$). The safety specification is marked by the automaton *SPEC* shown in Figure 5.18.

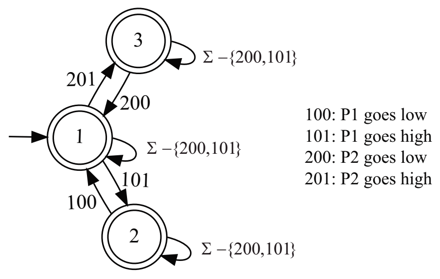


Figure 5.18: *SPEC*: Overall specification automaton

The specifications K_i are then obtained by finding the intersection of the language by *SPEC* and closed language of the plant.

$$\begin{aligned}
K_{\{N,1\}} &= product(SPEC, G_{\{N,1\}}) \\
K_{\{N,2\}} &= product(SPEC, G_{\{N,2\}}) \\
K_{\{NF,1\}} &= product(SPEC, G_{\{NF,1\}}) \\
K_{\{NF,2\}} &= product(SPEC, G_{\{NF,2\}})
\end{aligned}$$

Having plant models G_is and K_is, we can find the overall specification by Equation 3.1. The specification K has 348 states and 1594 transitions.

In order to formulate the problem as RNSCP-S we have to form the specifications K_is as a subautomaton of G_is. This can be accomplished by removing illegal states, where both of the pressures are high, from G_is. Therefore, the specification can be in terms of illegal states where both pressures are high. We consider a few scenarios of executed events in order to examine the validity of the RLL-S supervision.

5.2.3 Offline Supervisor

As described in [6] and [23] both multiple marking and fault recovery problems can be treated as special cases of robust supervision. Our example is a combination of both multiple marking and fault recovery problems. The supervisor should satisfy the safety requirement while ensuring that (1) we can achieve two different goals in normal mode (2) the system is nonblocking in both normal and faulty modes. Having the specification K_is as a subautomaton of G_is and considering the fact that the fault recovery problem can be considered with state set as the control domain [6], we model our problem as an RNSCP-S.

The procedures presented here are developed in MATLAB environment using Discrete Event Control Kit (DECK) [43] and are presented in Appendix B.

1. The problem is first initialized by building $G_{\{N,1\}}$, $G_{\{N,2\}}$, $G_{\{NF,1\}}$, $G_{\{NF,2\}}$ and also

the specifications $K_{\{N,1\}}$, $K_{\{N,2\}}$, $K_{\{NF,1\}}$, $K_{\{NF,2\}}$ by removing illegal states from the plant models.

2. Next, the procedure $E = \sup RCN(K, G, \Sigma_u, G_1, ..., G_4)$ applies *supremal relative-closed*, *supremal controllable* and *supremal G_i-nonblocking* operators iteratively until it finds the fixed point of Ω defined by $\Omega(Z) = \sup \mathcal{R}_G(\sup \mathcal{N}_{\mathcal{G}}(\sup \mathcal{C}_G(Z)))$ [21].

3. The offline supervisor is obtained by marking all the states of the automaton E from the previous step. The procedures $isNonblocking$, $isRelativeClosed$ and $controllable$ are used to verify that the system under supervision of the offline supervisor is nonblocking and admissible.

The resulting supervisor has 181 states and 647 transitions. We used the off-line supervisor in order to confirm that the RLL-S is valid and maximally permissive.

5.2.4 Online Solution

Next we consider the CMPS under supervision of the RLL-S supervisor. Unlike the Viking problem, the conditions of Theorem 4.2 and/or Remark 4.1 do not hold for CMPS. However, we would like to show that the problem is still solvable using the online approach, noting that the conditions in 4.2 are sufficient but not necessary. Here, we present an explanation of why the conditions in Theorem 4.2 do not hold for CMPS problem: In modeling of CMPS components we assumed that the failure events (which are uncontrollable) may occur at any time while the system is operating. This is not the case for the Viking problem. In the Viking problem, a failure may happen only after issuing a controllable command (e.g. commanding a valve to open/close). Thus, there are a few nonblocking (controllable) states in CMPS problem, as a failure may happen in most of the states.

Although the sufficient conditions do not hold for the CMPS problem, the problem is still solvable using the online approach. Starting from a lookahead window size of 2 and increasing it step-by-step we found a minimum required window length for maximal permissiveness of RLL-S supervisor. We verified the maximally permissiveness of RLL-S supervisor by comparing the enabled events in every states with off-line supervisor. The required window length is determined to be 8.

The procedures which are developed for implementation of RLL-S supervisor were previously explained and are the same as those of the Viking problem.

To observe the maximally permissiveness of RLL-S supervisor, we monitor the enabled events along two different sample paths. The sample path s_1 = V_{1O}-P_{1H}-V_{2O}-V_{1C}-P_{1L}-V_{1SC} is a sequence which contains the failure of valve V_1. We also consider another sample path s_2 = PV_{4F}-V_{2O}-P_{2H}-V_{2C}-V_{2SC}-P_{2L} which contains the failure of valve V_2.

By observing the enabled events along the path S_1 and comparing it with the behavior of the off-line supervisor, we can confirm the maximal permissiveness of the RLL-S supervisor. We can also observe that the RLL-S supervisor disables events PV_{1F}, PV_{2F} and PV_{3F} prior to occurrence of the failure. These events are later enabled for fault recovery mode. The automaton representing the set of enabled events in each step is represented in Figure 5.19.

Next we consider the enabled events along the executed path S_2. Like the previous scenario we can confirm maximally permissiveness of RLL-S supervisor by comparing it with the off-line solution. Observe that V_{1C} and V_{2C} are the only enabled events if both V_1 and V_2 are open. This happens because we assumed that the valves are fault free while they are open.

Lastly, we study the size of the automata which are used for online calculations (i.e. the number of states and transitions for K and G). This can be an indiction of computational complexity of RLL-S supervisor. A summary of this information is presented in Table 5.4

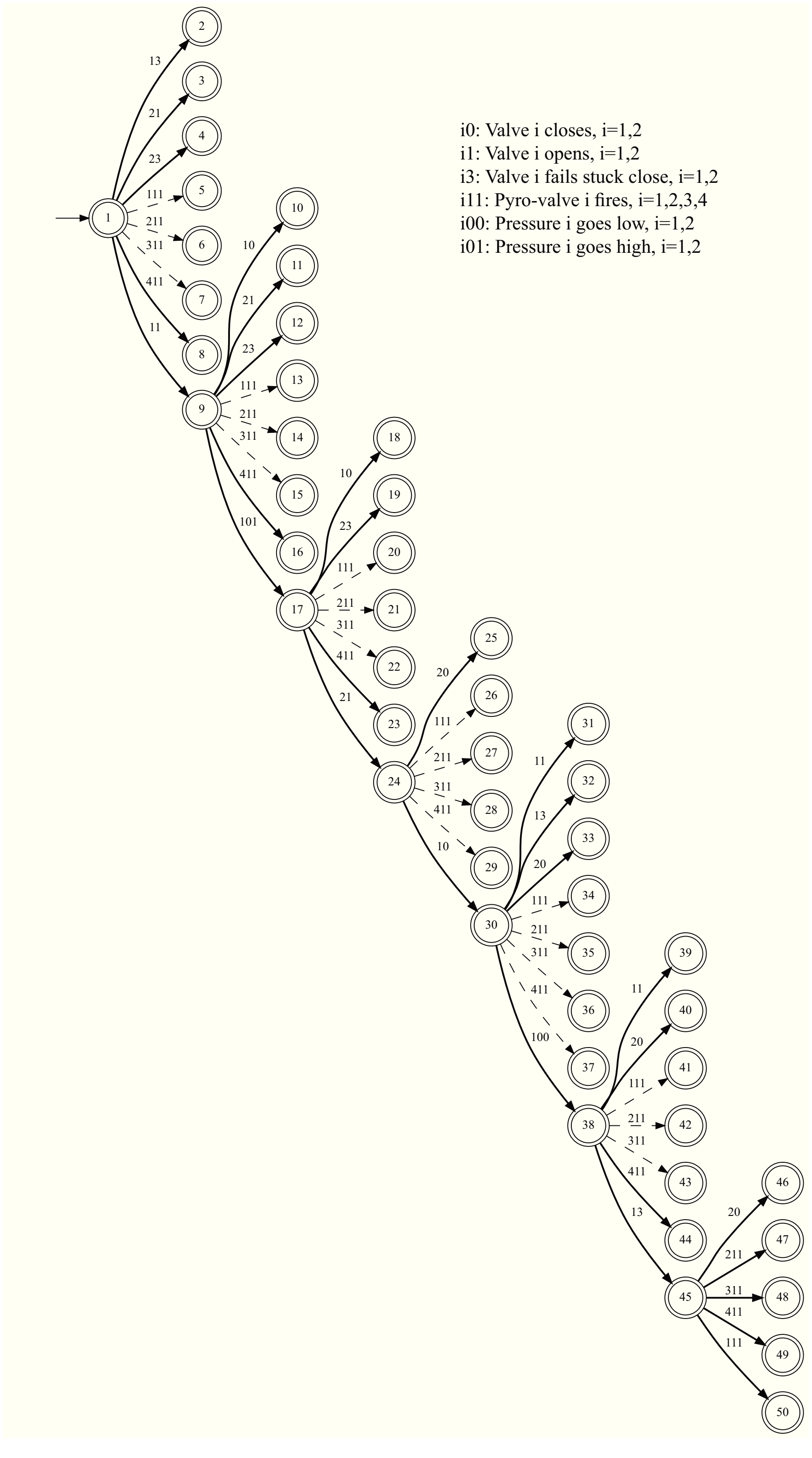
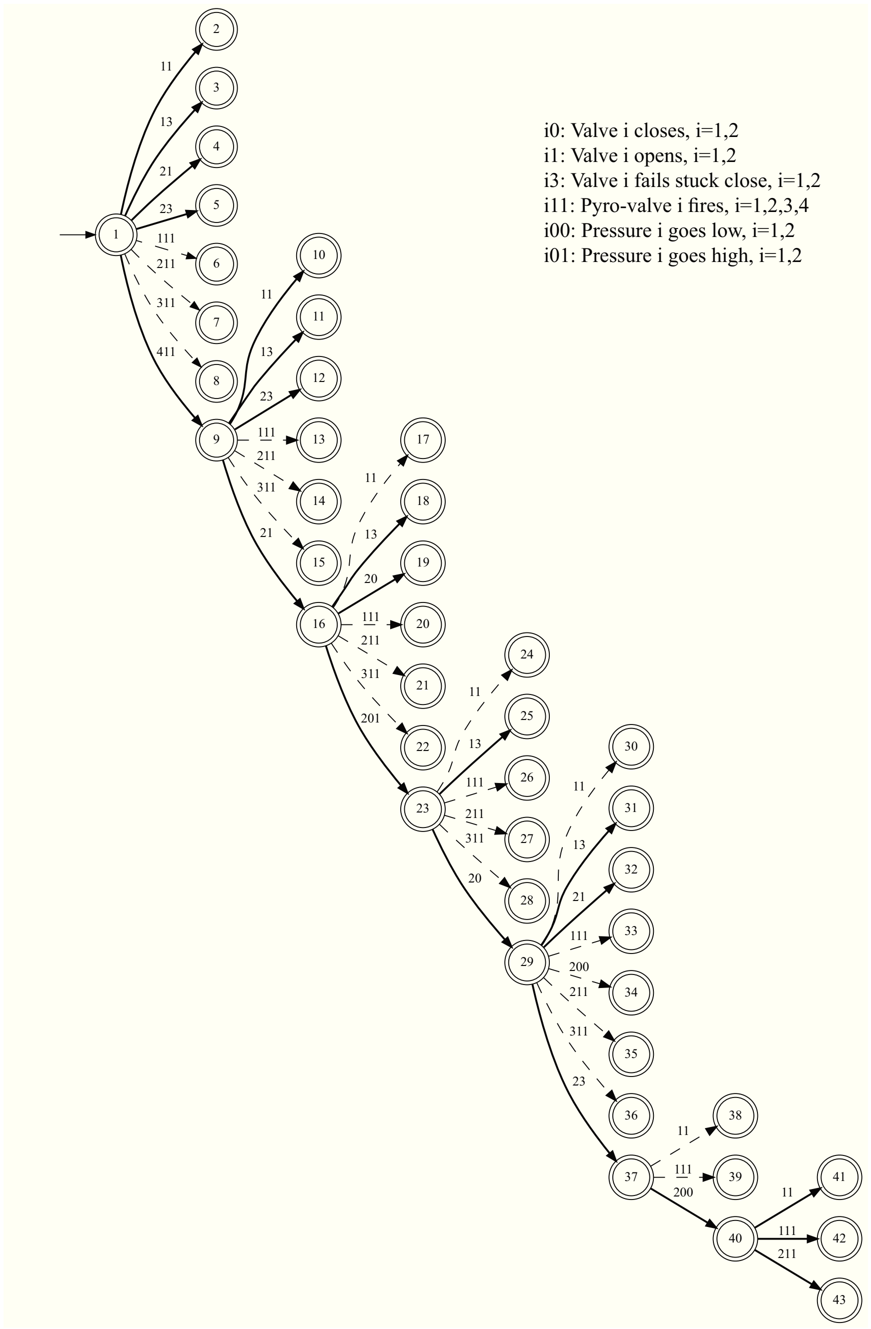

i0: Valve i closes, i=1,2
i1: Valve i opens, i=1,2
i3: Valve i fails stuck close, i=1,2
i11: Pyro-valve i fires, i=1,2,3,4
i00: Pressure i goes low, i=1,2
i01: Pressure i goes high, i=1,2

Figure 5.19: Enabled events by RLL-S supervisor along the path S_1



i0: Valve i closes, i=1,2
i1: Valve i opens, i=1,2
i3: Valve i fails stuck close, i=1,2
i11: Pyro-valve i fires, i=1,2,3,4
i00: Pressure i goes low, i=1,2
i01: Pressure i goes high, i=1,2

Figure 5.20: Enabled events by RLL-S supervisor along the path S_2

Table 5.4: Average sizes of expanded automata

Automata	# states	# transitions	# states in expansions			# transitions in expansions		
			min	avg	max	min	avg	max
G	348	1597	2	54	289	2	190	1184
K	292	1294	2	47	242	2	161	938

We observe that the online supervisor is capable of reducing computational complexity associated with off-line supervision. Specifically, the off-line problem with about 348 states and 1597 transitions has been reduced to an online problem with an average size 54 states and 190 transitions. We observe that at each state we can calculate the control action by looking at about %12 of the transitions and %15 of the states (on average) of the original plant and specification.

5.3 Conclusion

In this chapter, the proposed online RLL-S supervisor has been applied to robust supervision of the simplified versions of propulsion systems of Viking and Cassini spacecraft. The effectiveness of the RLL-S supervisor in reducing complexity associated with the computation of off-line supervisor has been evaluated.

Chapter 6

Conclusion

6.1 Summary

In this thesis we study the problem of robust nonblocking supervisory control of discrete event systems. We develop limited lookahead policies as an extension of previous work in literature [7]. The supervisor developed under this policy is named Robust Limited Lookahead (RLL) supervisor. We study the properties of RLL supervisor and obtain a set of sufficient conditions for maximal permissiveness of the supervision.

The RLL supervisor reduces the computational complexity of off-line supervisor as it only looks at the behavior of the system in the currently executed trace and over a limited horizon in the future. However, RLL supervisor may cause the behavior of the system to be more restrictive compared with the off-line maximally permissive supervisor. This happens if the length of lookahead widow is not large enough or the required length of window size is undefined.

In order to overcome this challenge, we formulate RNSCP with State information (RNSCP-S). Then RLL-S supervisor is proposed to solve the problem. In contrast with RLL supervisor, the required window size is always defined for RLL-S supervisor if the system is modeled with finite state automata.

Finally, we apply RLL-S supervision algorithm to simplified versions of Viking and Cassini main propulsion systems. The system is subject to failure and requires multiple set of marked states. This means that the problem should be formulated as a RNSCP. The procedures for solving the problem are developed in MATLAB environment based on Discrete Event Control Kit (DECK).

6.2 Future Work

In this thesis we assumed full observability of the system under supervision. Future research may include the extension of our results to the problem of robust nonblocking supervisory control under partial observation.

The implementation of the proposed algorithms can be optimized in term of computational complexity. For instance, if we setup a recursive algorithm, then it will be possible to use some of the information obtained in each step for the next step. Also the algorithm for the computation of supremal relative-closed, controllable, G_i-nonblocking sublanguage is computationally complex and should be optimized.

Two other lookahead policies are presented in literature for conventional supervisory control (1) optimistic policy, which assumes that all the pending traces are legal and marked, and (2) extension-based policy, which assumes that the behavior of the plant can be anything in Σ^* after a pending trace. These two policies can be generalized to the robust case. The required length of lookahead window, "N" for optimal supervisor can be obtained while adopting different policies. A good discussion is to compare "N" in different cases.

Variable Lookahead Policy (VLP) is also studied in literature for conventional supervisory control. VLP can also be applied to RNSCP. Finally, RLL supervision can be adopted in order to handel Timed Discrete Event Systems (TDES).

Only sufficient conditions are presented in this thesis and in the literature for optimality of online supervisors. Future research may include studying of the existence of necessary conditions for the optimality of online supervision. As we observed in the Cassini problem, the sufficient conditions may be restrict in some problem. Another suggestion is to develop an efficient algorithm to determine the minimum length of lookahead window (if exists) by comparing online supervisor with the predetermined off-line solution.

Bibliography

[1] Williams B. C., Nayak, P. P., “A Model-based Approach to Reactive Self-configuring Systems,”, *Proceedings of AAAI* , vol. 13, no. 2, pp. 971–978, 1996.

[2] Ramadge, P.J. and Wonham, W.M., “Supervisory control of a class of discrete-event systems,” *SIAM Journal of Control Optimization*, vol. 25, no. 1, pp. 206-230, 1987.

[3] Ramadge P. J. and Wonham W. M., “The Control of Discrete Event Systems,” *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[4] Ramadge P. J. and Wonham W. M., “On the Supremal Controllable Sublanguage of a given Language,” *SIAM Journal of Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.

[5] Chen, X. Y, Hashtrudi-Zad, S., “A direct approach to robust supervisory control of discrete-event systems,” *Proceedings of Canadian Conference on Electrical and Computer Engineering*, 2008, pp. 957–962, 2008.

[6] Chen, X. Y., “Online robust nonblocking supervisory control of discrete-event systems,” M.A.Sc. Thesis, Dept. Electrical and Computer Eng., Concordia Univ., Montreal, 2007.

[7] Chung, S.-L., Lafortune, S., Lin, F., “Limited lookahead policies in supervisory control of discrete event systems,” *IEEE Transactions on Automatic Control*, vol. 37 , no. 12, pp. 1921–1932, 1992.

[8] S. L. Chung, S. Lafortune and F. Lin, “Supervisory control using variable lookahead policies,” *Discrete Event Dynamic Systems: Theory and Applications*, vol. 4, no. 3, pp. 237–268, 1994.

[9] Hadj-Alouane, N.B., Lafortune, S., Feng Lin, “Variable lookahead supervisory control with state information,” *IEEE Transactions on Automatic Control*, vol. 39, no. 12, pp. 2398–2410, 1994.

[10] Kumar, R., Cheung, H.M. Marcus S.I., “Extension based Limited Lookahead Supervision of Discrete Event Systems,” *Automatica*, vol. 34, no. 11, pp. 1327–1344, 1998.

[11] Kumar, R., Cheung, H.M., Marcus, S.I., “Extension based limited lookahead control for discrete event systems,” *Proceedings of the 35th IEEE Decision and Control*, vol.2, pp. 2225–2230, 1996.

[12] Lin, F., “Robust and adaptive supervisory control of discrete event systems,” *IEEE Transactions on Automatic Control*, vol. 38, no.12, pp. 1848–1852, 1993.

[13] Takai, S., “Estimate based limited lookahead supervisory control for closed language specifications,” *Automatica*, vol. 33, no. 9, pp. 1739–1743, 1997.

[14] Takai, S., “Robust supervisory control of a class of timed discrete event systems under partial observation,” *Systems & Control Letters*, vol. 39, pp. 267–273, 2000.

[15] Takai, S., “Maximizing robustness of supervisors for partially observed discrete event systems,” *Automatica*, vol. 40, no. 3, pp. 531-535, 2004.

[16] Takai, S., “Maximizing robustness of supervisors for partially observed discrete event systems,” *Proceedings of the 10th IFAC Symposium on Large Scale Systems: Theory and Applications*, pp. 373-3780, 2004.

[17] Takai, S., “Synthesis of maximally permissive and robust supervisors for prefix-closed language specifications,” *IEEE Transactions on Automatic Control*, vol. 47, no. 1, pp. 132-136, 2002.

[18] Takai, S., “Maximally permissive robust supervisors for a class of specification languages,” *Proceedings of IFAC Conference on System Structure and Control*, pp. 445-450, 1998.

[19] Park, S.-J., Lim, S., “Robust and nonblocking supervisory control of nondeterministic discrete event systems using trajectory models,” *IEEE Transaction on Automatic Control*, vol. 47, pp. 655–658, 2002.

[20] Park, S.-J., Lim, S., “Non-blocking supervision for uncertain discrete event systems with internal unobservable transitions,” *IEE Proceedings Control Theory Applications*, vol. 152, pp. 165–170, 2005.

[21] Bourdon, S.E., Lawford, M., Wonham, W.M., “Robust nonblocking supervisory control of discrete-event systems,” *IEEE Transactions on Automatic Control*, vol.50, no.12, pp. 2015–2021, 2005.

[22] Saboori, A., Hashtrudi-Zad, S., “Fault recovery in discrete event systems,” *Computational Intelligence Methods and Applications, Congress on ICSC*, six pages, 2005.

[23] Saboori A. , Hashtrudi-Zad, S. , “Robust nonblocking supervisory control of discrete-event systems under partial observation,” *Systems & Control Letters*, vol. 55, no. 10, pp. 839–848, 2006.

[24] Winacott, C., Rudie, K., “Limited lookahead supervisory control of probabilistic discrete-event systems,” *47th Annual Allerton Conference on Communication, Control, and Computing*, pp. 660–667, 2009.

[25] Winacott, C., Behinaein, B., Rudie, K., “Methods for the estimation of the size of lookahead tree state-space,” *Journal of Discrete Event Dynamic Systems*, vol. 23, no. 2, pp. 135–155, 2013.

[26] Winacott, C., “Limited Lookahead Control of Discrete-Event Systems: Cost, Probability, and State Space,” M.A.Sc. Thesis, Dept. Electrical and Computer Eng., Queen’s Univ., Kingston, 2012.

[27] Chung, S. L., S. Lafortune and F. Lin, “Addendum to limited lookahead policies in supervisory control of discrete event systems: proofs of technical results,” Technical Report CGR- 92- 6, University of Michigan, Ann Arbor, Michigan, 1992.

[28] Fekri, M.Z., Hashtrudi-Zad, S., “Hierarchical robust supervisory control of discrete-event systems,” *Proceedings of American Control Conference*, pp.1178–1183, June 2008.

[29] Cury, J.E.R., Krogh, B.H., “Robustness of supervisors for discrete-event systems,” *IEEE Transactions on Automatic Control*, vol. 44, no. 2, pp. 376–379, 1999.

[30] Hill, R.C., Tilbury, D.M., Lafortune, S., “Covering-based supervisory control of partially observed discrete event systems for state avoidance,” *9th International Workshop on Discrete Event Systems*, pp. 28-30, 2008

[31] Lafortune, S., Chen, E., “The infimal closed controllable superlanguage and its application in supervisory control,” *IEEE Transactions on Automatic Control*, vol. 35, no. 4, pp.398–405, 1990.

[32] Zhong, H. and Wonham, W.M., “On the consistency of hierarchical supervision in discrete-event systems,” *IEEE Transactions on Automatic Control*, vol. 35, no. 10, pp. 1125-1134, 1990.

[33] Kurien, J., R-Moreno, M.D., “Costs and Benefits of Model-based Diagnosis,” *Proceedings of IEEE Aerospace Conference*, pp. 1–14, 2008.

[34] Williams, B. C., “Model-based autonomous systems in the new millennium”, *Proceedings of the 3rd International Conference on Artificial Intelligence Planning Systems*, AAAI Press, pp. 275-282 1996.

[35] Brown, C., “*Spacecraft Propulsion*,” AIAA, 1996.

[36] Dieter K. H., David H.H, “*Modern Engineering for Design Liquid-Propellant Rocket Engines*,” AIAA, 1992.

[37] Harland, D. M., Lorenz, R., “*Space Systems Failure*,” Springer, 2005.

[38] Bajwa A. and Sweet A., “The Livingstone Model of a Main Propulsion System”, *Proceedings of IEEE Aerospace Conference*, vol. 2, pp. 869–876, 2002.

[39] Morgan, P.S., “Cassini spacecraft’s in-flight fault protection redesign for unexpected regulator malfunction,” *Proceedings of IEEE Aerospace Conference*, pp.1–14, 2010.

[40] Wertz, J. R., Larson, W. J., “*Space Mission Analysis and Design*”, Klumer Academic Publishers/Microcosm press, 1999.

[41] Wonham, W.M., *Supervisory Control of DiscreteEvent Systems*, Systems Control Group, Edward S. Rogers Sr. Dept. of Electrical and Computer Engineering, University of Toronto, Canada, 2012; available at http://www.control.utoronto.ca/DES.

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

[43] Hashtrudi Zad, S., “Discrete Event Control Kit (DECK 1.2012.10) User Manual”, available at http://users.encs.concordia.ca/ shz/deck/DECK-manual.pdf

[44] Discrete Event Control Kit (DECK), Department of Electrical and Computer Engr., Concordia University. [Online]; available at http://www.ece.concordia.ca/ shz/deck.

[45] Ellson J., Gansner E. R., Koutsofios E., North S. C., Woodhull G., "*Graphviz and Dynagraph Static and Dynamic Graph Drawing Tools*,"; available at http://www.graphviz.org/Documentation/EGKNW03.pdf

Appendix A

Spacecraft Propulsion System Diagrams

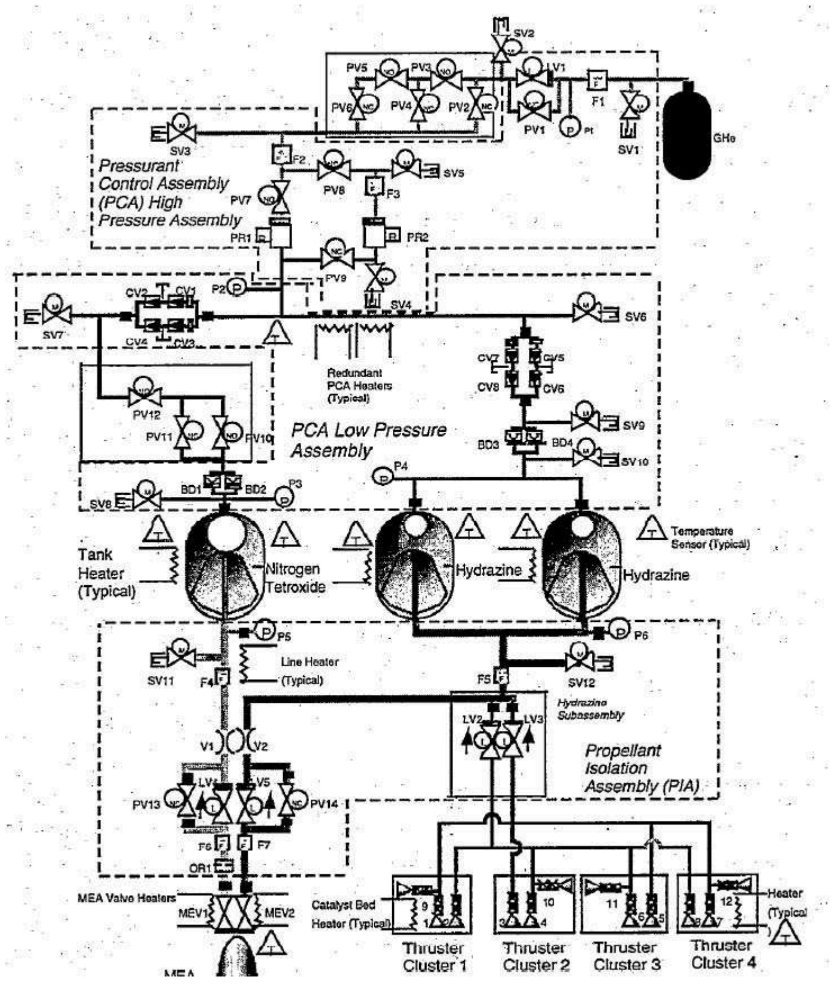


Figure A.1: Mars Global Surveyor [35].

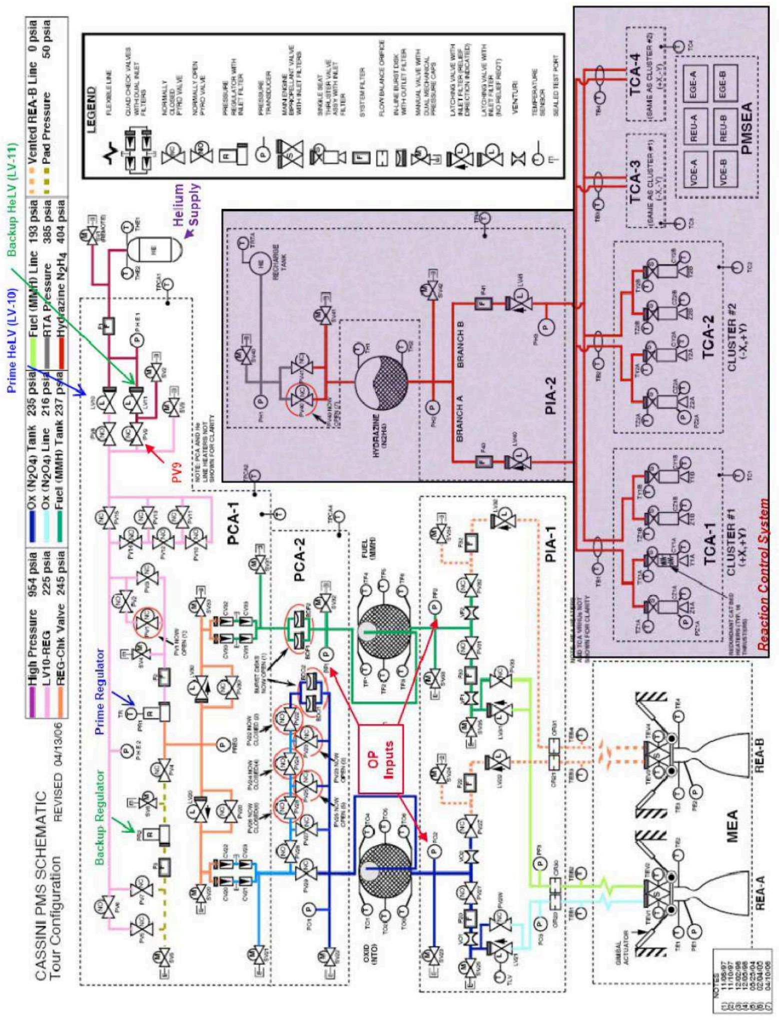


Figure A.2: Cassini Spacecraft Propulsion System [39].

Appendix B

Discrete Event Control Kit (DECK)

DECK is a toolbox written in MATLAB for the analysis and supervisory control of discrete event systems and it is equipped with the following functions [43], [44].

- Automaton: Creates an automaton model (automaton object) for use by DECK.
- Automatonchk: Verifies the validity of an automaton object.
- Complement: Returns complement of a deterministic automaton.
- Controllable: Determines if a language is controllable.
- Product: returns product of automata.
- Reach: Finds the reachable states of transition graph.
- Reachable: Finds reachable subautomaton.
- Selfloop: Adds selfoops to automaton.
- Supcon: Finds supremal controllable sublanguage.
- Sync: Returns synchronous product of automata.
- Trim: Finds the reachable and coreachable subautomaton.

Appendix C

MATLAB Code

This appendix complies the set of routines developed in the course of this thesis in the following four sections: (i) functions for manipulation of automata (ii) off-line control (iii) online control and (iv) examples.

C.1 Manipulation of automata

selfloopState: Adds selfloop to a specific state of an automaton.

```
function Gs=selfloopState(G,S,Es)
% SELFLOOPSTATE   Adds selfloops to state S of automaton G
%
% SYNTAX:   Gs=selfloopState(G,S,Es)
%
% INPUTS:   G      Input automaton
%           S      A state of input automaton
%           Es     List of events (vector)
%
% OUTPUTS:  Gs     Output automaton
%
% Empty input automaton
%
if G.N==0
  Gs=automaton(0,[],[]);
  return;
end
```

```
%
% Es empty
%
if isempty(Es)
  Gs=G;
  return;
end
%
Es=unique(Es);
if ¬isrow(Es)
  Es=Es';
end
%
% The event set of the input automaton and Es must be disjoint.
%
if ¬isempty(G.TL)
    Ind=G.TL(:,1)==S;
    E=unique(G.TL(Ind,2))';
else
  E=[];
end

if ¬isempty(intersect(E,Es))
  error('The active event set of G at S and Es in SELFLOOPSTATE ...
      must be disjoint.');
end
%
if ¬isempty(G.TL)
  TLs=[G.TL; zeros(length(Es),3)];
  j=size(G.TL,1);
else
  TLs=zeros(length(Es),3);
  j=0;
end
%
vi=zeros(length(Es),1);

vi(:)=S;
TLs(j+1:j+length(Es),:)=[vi,Es',vi];
%
```

```
Gs=automaton(G.N,TLs,G.Xm);
end
```

Closure: finds prefix closure of an automaton

```
function Gc=closure(G)

% CLOSURE  CLOSURE of an automaton
%
% SYNTAX:   Gco=complement(G)
%
% INPUTS:   G      Input automaton
%
% OUTPUTS:  Gco    Output deterministic automaton
%
G=trim(G); % Find all the states which are reachable to a marked ...
    state
G.Xm=1:G.N;
Gc=G;
end
```

Gunion: finds union of two automata

```
function G=Gunion(G1,G2)

% Gunion    Union of two automata
%
% SYNTAX:   G=Gunion(G1G2)
%
% INPUTS:   Gi     Input automaton i (i=1,2)
%
% OUTPUTS:  G         Output automaton
%

Ea=union(G1.TL(:,2),G2.TL(:,2));
Ea1=setdiff(Ea,G1.TL(:,2));
Ea2=setdiff(Ea,G2.TL(:,2));
%
G1com=complement(G1,Ea1');
```

```
G2com=complement(G2,Ea2');
%
G=product(G1com,G2com);
G=complement(product(G1com,G2com),setdiff(Ea,G.TL(:,2)));
G=trim(G);
end
```

isIncluded: tests if $L_m(G_1) \subseteq L_m(G_2)$

```
function E=isIncluded(G1,G2)

% ISINCLUDED  Checks if Lm(G1) \subset Lm(G2)
%
% SYNTAX:   E=isIncluded(G1,G2)
%
% INPUTS:   G1,G2       Input automatons
%
% OUTPUTS:  E    E is one if Lm(G1) \subset Lm(G2) and it is zero ...
    otherwise
if isempty(G2.TL)
    E=0;
    return
elseif isempty(G1.TL)
    E=1;
    return
end
%
Sigma=union(G1.TL(:,2),G2.TL(:,2));
Ea=setdiff(Sigma,G2.TL(:,2));
%
G2comp=complement(G2,Ea);
G=product(G1,G2comp);
%
if isequal(length(G.Xm), 0)
    E=1;
else
    E=0;
end
end
```

isNonblocking: tests if an automaton is G-nonblocking

```
function b=isNonblocking(K,varargin)

% ISNONBLOCKING  Checks if for all Gi, i=1,...,n
%                pr(K) [INTERSEC] L(Gi) \subset pr(K [INTERSECT] ...
   Lm(Gi))
%
% SYNTAX:   E=isNonblocking(K,G1,...,Gn)
%
% INPUTS:   K      Input automaton to bechecked
%           Gi     The set of plant models, i = 1,...,n
%
% OUTPUTS:  b      b is one if if K is Gi-nonblocking and zero ...
   otherwise
Km=closure(K);
B=[];
%
for i=1:length(varargin)
    G=varargin{i};
    Gm=G;
    Gm.Xm=1:Gm.N;
    LHS=product(Km,Gm);
    RHS=trim(closure(trim(product(K,G))));
    %
    if isIncluded(LHS,RHS)
       B=[B,1];
    else
       B=[B,0];
    end
end
%
if unique(B)==1
    b=1;
else
    b=0;
end
%
end
```

C.2 Offline Control

supNb: finds supremal G-nonblocking sublanguage

```
function H=supNb(K,Gi,Sigma)

% SUPNB  Supremal Gi-nonblocking Sublanguage
%
% SYNTAX:   K=supNb(K,Gi,Sigma)
%
% INPUTS:   K      Specification (deterministic) automaton
%           G      Plant (deterministic) automaton
%           Sigma  System's alphabet (union model)
%
% OUTPUTS:  K    Trim (deterministic) automaton marking supremal
%                Gi-nonblocking sublanguage.
GiM=Gi; % mark all the states of Gi, so L(GiM) == Lm(GiM) == L(Gi)
GiM.Xm=1:GiM.N;
%
%% pr(K) [INTERSECTION] Lm(GiM) == pr(K) [INTERSECTION] L(Gi)
%
Kbar=closure(K);
L1=product(Kbar,GiM);
%
%% (K [INTERSECTION] Lm(Gi))
%
L2=product(K,Gi);
L2=closure(L2);
%
%% Form L1-L2
%
if ¬isempty(L2.TL)
    Ea=setdiff(Sigma,L2.TL(:,2));
else
    Ea=Sigma;
end
%
L3=complement(L2,Ea');
L4=product(L1,L3);
```

```
%
%% Form L4 (concatination) Sigma^*
%
% First, remove all the outgoing transitions from L4.Xm
Ind=[];
if ¬isempty(L4.Xm)
    for i=1:size(L4.Xm)
       Ind=[Ind,find(L4.TL(:,1)≠L4.Xm(i))];
    end
end
L4.TL=L4.TL(Ind,:);
%
% Add selfloops of Sigma to all states in L4.Xm
%
TL=L4.TL;
Ea=Sigma;
for i=1:size(L4.Xm,2)
    Ei=L4.TL(L4.TL(:,1)==L4.Xm(i),2);
    L4=selfloopState(L4,L4.Xm(i),setdiff(Ea,unique(Ei)));
end
%
%% Find K-L4 = product(K,complement(L4))
%
if ¬isempty(L4.TL)
    Ea=setdiff(Sigma,L4.TL(:,2));
else
    Ea=Sigma;
end
L5=complement(L4,Ea');
%
H=trim(product(K,L5));
end
```

supR: finds supremal $L_m(G)$-closed sublanguage

```
function E=supR(K,G)

% SUPR  Supremal Relative-Closed Sublanguage
%
% SYNTAX:   K=supR(K,G)
```

```
%
% INPUTS:    K      Specification (deterministic) automaton
%            G      Plant (deterministic) automaton
%
% OUTPUTS:   E    Trim (deterministic) automaton marking supremal
%                 Lm(G)-closed sublanguage.
%
% Case 1:
%
if isempty(K.TL)
    E=automaton(1,[],[]);
    return
end
%
% Case 2:
%
Ea=setdiff(unique(G.TL(:,2)),unique(K.TL(:,2)));
Kcom=trim(complement(K,Ea'));
%
R=product(G,Kcom);
%
% Removing active events from marked states of R
%
Ind=[];
if ¬isempty(R.Xm)
   for i=1:size(R.Xm)
    Ind=[Ind,find(R.TL(:,1)≠R.Xm(i))];
   end
end
%
R.TL=R.TL(Ind,:);
%
% Adding self-loops to the marked states
%
TL=R.TL;
Ea=unique(G.TL(:,2));
%
for i=1:size(R.Xm,2)
    TLs=[R.Xm(i)⋆ones(size(Ea)),Ea,R.Xm(i)⋆ones(size(Ea))];
    TL=[TL;TLs];
```

```
end
%
R.TL=unique(TL,'rows');
%
% SupR(K)=K-R
%
Ea=setdiff(unique(G.TL(:,2)),unique(R.TL(:,2)));
Rcom=trim(complement(R,Ea'));
size(Rcom.TL)
size(K.TL)
E=trim(product(Rcom,K));

end
```

supRCN: finds supremal $L_m(G)$-closed, controllable, G-nonblocking sublanguage

```
function K=supRCN(E,G,Eu,varargin)

% SUPRCN  Supremal Relative-Closed, Controllable, Gi-nonblocking
%         Sublanguage of E
%
% SYNTAX:   K=supRCN(E,G,Eu,G1,...,Gn)
%
% INPUTS:   E      Specification (deterministic) automaton
%           G      Union model (deterministic) automaton
%           Eu     List of uncontrollable events (vector)
%           Gi     Plant models automaton
%
% OUTPUTS:  E    Trim (deterministic) automaton marking supremal
%                Lm(G)-closed, controllable, Gi-nonblocking ...
    sublanguage.

Sigma=unique(G.TL(:,2));
for i=1:length(varargin)
    Gi{i}=varargin{i};
end

gray=1;

while gray≠0
```

```
    if ¬isempty(E.TL)


    S=supcon(E,G,Eu);
    S=supR(S,G);

    for i=1:length(varargin)
    S=supNb(S,Gi{i},Sigma);
    end

    if isequal(size(S.TL),size(E.TL))
        if isequal(length(S.Xm),length(E.Xm))
           if S.TL==E.TL
               if S.Xm==E.Xm
        gray=0;
               end
           end
        end
    end
        K=S;
    else
        return
    end

end
end
```

GiveSpec: Returns overall specification of RNSCP

```
function K=GiveSpec(varargin)

% GIVESPEC  Gives overall specification for robust problem.
%
% SYNTAX:   K=GiveSpec(G1,...,Gn,K1,...,Kn)
%
% INPUTS:   Ki    Models' specifications, i = 1,...,n
%           Gi    Different models of robust problem
%
% OUTPUTS:  K     Overall specification of RNSCP
%
```

```
N=(length(varargin))/2; % find the number of plants
%
for i=1:N
    Ei{i}=varargin{N+i};
    Gi{i}=varargin{i};
end
%
G=Gi{1}; % union model
%
for i=2:N
    G=Gunion(G,Gi{i});
end
%
Sigma=unique(G.TL(:,2));
%
% Find (Sigma^*-Lm(Gi))
%
for i=1:N
    compGi{i}=complement(Gi{i},setdiff(Sigma,Gi{i}.TL(:,2))');  ...
        unique(compGi{i}.TL(:,2))
end
%
% Find (Ei (UNION) (Sigma^*-Lm(Gi))
%
for i=1:N
    U{i}=Gunion(Ei{i},compGi{i});unique(U{i}.TL(:,2))
end
%
% PU=product(Ui)
%
PU=U{1};
%
for i=2:length(U)
    PU=product(PU,U{i});
end
%
K= product(G,PU);
end
```

C.3 Online Control

expandLLSB: Expands G as a subautomaton up to N step

```
function Gexp=expandLLSB(G,S,N)

% EXPANDLLSB  Expands G as a subautomaton.
%
% SYNTAX:   Gexp=expandLLSB(G,S,N)
%
%
% INPUTS:   G     Input automaton to be expanded
%           S     An automaton which marks Currently Executed ...
    Trace (CET)
%           N     Number of steps expansion
%
% OUTPUTS:  Gexp  Expansion of input automaton G

%
% Change initial state to the state marked by S
%
[¬,states]=product(G,S);
%
r= states(:,2)==S.Xm;
SI=states(r,1);
%
Ind1= G.TL(:,1)==1;
Ind2= G.TL(:,3)==1;
Ind3= G.TL(:,1)==SI;
Ind4= G.TL(:,3)==SI;
%
G.TL(Ind1,1)=SI;
G.TL(Ind2,3)=SI;
G.TL(Ind3,1)=1;
G.TL(Ind4,3)=1;
%
Ind1= G.Xm==1;
Ind2= G.Xm==SI;
G.Xm(Ind1)=SI;
```

```
G.Xm(Ind2)=1;
%
% Expansion by using breath-first search algorithm
%
Gtemp=G;
Active_states=1;
TL=[];
%
for i=1:N
    Ind=ismember(Gtemp.TL(:,1),Active_states);
    TL=[TL;Gtemp.TL(Ind,:)];
    Active_states=unique(G.TL(Ind,3));
end
%
Gtemp.TL=unique(TL,'rows');
Gexp=reachable(Gtemp);

end
```

expandVLSB: Expands G as a subautomaton. Stops expansion at marked controllable states or noncoreachable states.

```
function [Gexp,L]=expandVLSB(G,S,Euc,N)
% EXPANDVLSB  Expands G as a subautomaton. Stops expansion at marked
%             controllable states or noncoreachable states.
%
% SYNTAX:   Gexp=expandVLSB(G,S,Euc,N)
%
% INPUTS:   G     Input automaton to be expanded
%           S     An automaton which marks Currently Executed ...
   Trace (CET)
%           Euc   List of uncontrollable events
%           N     Maximum number of steps expansion
%
% OUTPUTS:  Gexp  Expansion of input automaton G
%           L     The actual number of expansion steps
%

%
```

```
% Change initial states to the state marked by S
%
[¬,states]=product(Gm,S);

r= states(:,2)==S.Xm;
SI=states(r,1);

Ind1= G.TL(:,1)==1;
Ind2= G.TL(:,3)==1;

Ind3= G.TL(:,1)==SI;
Ind4= G.TL(:,3)==SI;

G.TL(Ind1,1)=SI;
G.TL(Ind2,3)=SI;
G.TL(Ind3,1)=1;
G.TL(Ind4,3)=1;

Ind1= G.Xm==1;
Ind2= G.Xm==SI;
G.Xm(Ind1)=SI;
G.Xm(Ind2)=1;
%
% remove outgoing transitions from marked controllable states
%
Gc=statesCTL(G,Euc);
Xmc=setdiff(Gc.Xm,1);

Ind=¬ismember(G.TL(:,1),Xmc);
G.TL=G.TL(Ind,:);
%
% remove outgoing transitions of noncoreachable states
%
nonCR=[]; % List of non-coreachable states

for i=1:G.N
    X=reach(G.TL,i);
    if isempty(intersect(X,Xmc))
        nonCR=[nonCR,i];
    end
```

```
end

Ind=¬ismember(G.TL(:,1),nonCR);
G.TL=G.TL(Ind,:);

G=trim(G);
Gtemp=G;

Active_states=1;
TL=[];
L=1;

for i=1:N
    Ind=ismember(Gtemp.TL(:,1),Active_states);
    TLtmp=unique(TL,'rows');
    TL=unique([TL;Gtemp.TL(Ind,:)],'rows');
    Active_states=unique(G.TL(Ind,3));
    %
    if size(TL,1)≠size(TLtmp,1)
       L=L+1;
    end
    %
end

Gtemp.TL=unique(TL,'rows');
Gexp=trim(Gtemp);

end
```

detKnb: determines controllable and uncontrollable nonblocking states.

```
function [Knbc,Knbu]=detKnb(K,Euc,varargin)

% DETKNB  Determins nonblocking states of specification automaton K
%
% SYNTAX:   Knbc=detKnb(K,Euc,G1,...Gn)
%           [Knbc,Knbu]=detKnb(K,Euc,G1,...Gn)
%
% INPUTS:   K     Specification automaton
%           Euc   List of uncontrollable events (vector)
%           Gi    List of possible plant models
%
% OUTPUTS:  Knbc  automaton with all controllable nonblocking ...
    states marked
%           Knbc  automaton with unontrollable nonblocking states ...
    marked

% Initialization

K.Xm=1:K.N;
Ktemp=K;
Ktemp.Xm=[];
XmNb=[];
N=0;

% Mark one state of K at a time and check whether the state is ...
    nonblocking

for i=1:length(K.Xm)
    tic
    Ktemp.Xm=K.Xm(i);
    Ktemp=closure(Ktemp);
    for j=1:length(varargin)
        if isNonblocking(Ktemp,varargin{j})
           N=N+1;
           if N==length(varargin) % if it is nonblocking for G1,...Gn
               XmNb=[XmNb,K.Xm(i)];
           end
        end
    end
    N=0;
    toc
end

Knb=K;
Knb.Xm=XmNb;

Knbc=statesCTL(Knb,Euc);
Knbu=Knb;
Knbu.Xm=setdiff(Knb.Xm,Knbc.Xm);
```

```
end
```

detNSB: determines N_B as defined before

```
function NB=detNSB(Knb,Euc)

% DETKNB    Determins NB which is required to determine length of ...
    window
%
% SYNTAX:   NB=detNSB(Knb,Euc)
%
% INPUTS:   Knb   Input automaton marking nonblocking states of SPEC
%           Euc   List of uncontrollable events (vector)
%
% OUTPUTS:  NB    NB as defined in the thesis

Xnb=[1,Knb.Xm];
NN=[];
NandS=[];
for i=1:length(Xnb)
     %
     Ktemp=Knb;
     Ktemp.Xm=Xnb(i);
     S=Ktemp;
     %
    [¬,N]=expandVLSB(Knb,S,Euc,Knb.N);
    NN=[NN,N];
    NandS=[NandS;N,S.Xm];
end

NB=max(NN);

end
```

RLL: implements RLL supervisor

```
function [CET,ONLINE,ONLINE_SYS]=RLL(E,G,Eu,N,varargin)

% RLL       Implements RLL supervisor to solve a RNSCP. It returns
%           the set of enabled events at each step, so the user ...
    can choose
%           among the different possibilities. It cotinues until the
%           execution is intrupted by the user. The program ...
    returns an
%           automaton which markes the executed trace.
%
% SYNTAX:   CET=RLL(E,G,Eu,G1,...,Gn)
%
%
% INPUTS:   E     The overall specification of RNSCP
%           G     Union model
%           Eu    Lis of uncontrollable events
%           N     Length of lookahead window
%           Gi    Set of plant models, i = 1,..., n
%
% OUTPUTS:  CET   Output automaton markes the currently executed ...
    trace.
%

for i=1:length(varargin)
    Gi{i}=varargin{i};
end

CET=automaton(1,[],[]); %currently executed trace
CET.Xm=CET.N; %the last state of the string should be marked
ONLINE=CET; %the events enabled by RLL through execution
ONLINE_SYS=CET;

gray=1;
T=[];
while gray
    tic
    for i=1:length(varargin)
        [temp,states]=product(CET,Gi{i});
        if isequal(CET.TL,temp.TL)
            In_Gi(i)=states(states(:,1) == CET.N,2);
        else
            In_Gi(i)=NaN;
        end

    end

    Giexp={};

    for i=1:length(varargin)
        if ¬isnan(In_Gi(i))
            Giexp{end+1}=expandLLSB(Gi{i},CET,N);
        end
    end
    %
    EexpN=expandLLSB(E,CET,N-1);
    Gexp=expandLLSB(G,CET,N);
    EexpN=product(EexpN,Gexp);
    Sexp=supRCN(EexpN,Gexp,Eu,Giexp{1:end});
    %
    if isempty(Sexp.TL)
       disp('epsilon')
       return
    else
    Enabled_Events=unique(Sexp.TL(Sexp.TL(:,1)==1,2));
    Active_Events=unique(Gexp.TL(Gexp.TL(:,1)==1,2));
    %Disabled_Events=setdiff(Active_Events,Enabled_Events);
    %
    %deck2gviz(Sexp);
    %winopen('Sexp.gv')
    fprintf('Enabeled events:%s     \n ',num2str(Enabled_Events'))

    reply = input('EXE?\n');

    if ¬isempty(reply)
                CET.N=CET.N+1;
        CET.TL=[CET.TL;CET.N-1,reply,CET.N];
        CET.Xm=CET.N;

        ONLINE.TL=[ONLINE.TL;ONLINE.N,reply,...
            ONLINE.N+length(Enabled_Events)];
        Enabled_Events=setdiff(Enabled_Events,reply);
        %
        ONLINE_SYS.TL=[ONLINE_SYS.TL;ONLINE_SYS.N,reply,...
```

```
            ONLINE_SYS.N+length(Active_Events)];
        Active_Events=setdiff(Active_Events,reply);
        %
        for i=1:length(Enabled_Events)
            ONLINE.TL=[ONLINE.TL;ONLINE.N,Enabled_Events(i),ONLINE.N+i];
        end
        %
        ONLINE.N=max(ONLINE.TL(:,3));
        ONLINE.Xm=1:ONLINE.N;
        %
                %
        for i=1:length(Active_Events)
            ONLINE_SYS.TL=[ONLINE_SYS.TL;ONLINE_SYS.N,...
                Active_Events(i),ONLINE_SYS.N+i];
        end
        %
        ONLINE_SYS.N=max(ONLINE_SYS.TL(:,3));
        ONLINE_SYS.Xm=1:ONLINE_SYS.N;
    else
        return
    end


   end
end
  T(end+1)=toc;
end
```

C.4 Examples

```
%% Offline solution to Cassini Main Propulsion System Problem

%% DES Normal/Normal-faulty of Models of V1 and V2
V1_NF=automaton(3,[1,11,2;2,10,1;1,13,3],1:3);
V2_NF=automaton(3,[1,21,2;2,20,1;1,23,3],1:3);
V1_N=automaton(2,[1,11,2;2,10,1],1:2);
V2_N=automaton(2,[1,21,2;2,20,1],1:2);
```

```

%% DES Models of pyro valves
PV1=automaton(2,[1,111,2],1:2);
PV2=automaton(2,[1,211,2],1:2);
PV3=automaton(2,[1,311,2],1:2);
PV4=automaton(2,[1,411,2],1:2);

%% DES Models of Pressure sensors
P1_M1=automaton(2,[1,101,2;2,100,1],2);
P2_M1=automaton(2,[1,201,2;2,200,1],1);
P_M1=sync(P1_M1,P2_M1);
%
P1_M2=automaton(2,[1,101,2;2,100,1],1);
P2_M2=automaton(2,[1,201,2;2,200,1],2);
P_M2=sync(P1_M2,P2_M2);

%% Interaction between valves status and sensor readings
[INT1_N,states]=sync(V1_N,PV1,PV3);

for i=1:size(states,1)
    if (states(i,1)==2 || states(i,2)==2) && states(i,3)==1
        INT1_N.TL=[INT1_N.TL;i,101,i];
    else
        INT1_N.TL=[INT1_N.TL;i,100,i];
    end
end

[INT1_NF,states]=sync(V1_NF,PV1,PV3);

for i=1:size(states,1)
    if (states(i,1)==2 || states(i,2)==2) && states(i,3)==1
        INT1_NF.TL=[INT1_NF.TL;i,101,i];
    else
        INT1_NF.TL=[INT1_NF.TL;i,100,i];
    end
end

[INT2_N,states]=sync(V2_N,PV2,PV4);

for i=1:size(states,1)
```

```
    if (states(i,1)==2 || states(i,2)==2) && states(i,3)==2
        INT2_N.TL=[INT2_N.TL;i,201,i];
    else
        INT2_N.TL=[INT2_N.TL;i,200,i];
    end
end

[INT2_NF,states]=sync(V2_NF,PV2,PV4);

for i=1:size(states,1)
    if (states(i,1)==2 || states(i,2)==2) && states(i,3)==2
        INT2_NF.TL=[INT2_NF.TL;i,201,i];
    else
        INT2_NF.TL=[INT2_NF.TL;i,200,i];
    end
end

%% Normal/Normal-faulty models of the plant and their specifications
[sys_M1_N,states_M1]=sync(P1_M1,P2_M1,V1_N,V2_N,PV1,PV2,PV3,PV4,...
    INT1_N,INT2_N);
%
K1=sys_M1_N;
Ind1=find(states_M1(:,1)==2 & states_M1(:,2)==2);
%
% Make specifications by removing illigal states.
%
for i=1:length(Ind1)
    K1.TL=K1.TL(K1.TL(:,1)≠Ind1(i),:);
    K1.TL=K1.TL(K1.TL(:,3)≠Ind1(i),:);
end
%
[sys_M2_N,states_M2]=sync(P1_M2,P2_M2,V1_N,V2_N,PV1,PV2,PV3,PV4,...
    INT1_N,INT2_N);
%
% Make specifications by removing illigal states.
%
K2=sys_M2_N;
Ind2=find(states_M2(:,1)==2 & states_M2(:,2)==2);
%
for i=1:length(Ind2)
```

```
    K2.TL=K2.TL(K2.TL(:,1)≠Ind2(i),:);
    K2.TL=K2.TL(K2.TL(:,3)≠Ind2(i),:);
end
%
[sys_NF1,states_NF1]=sync(P1_M2,P2_M2,V1_NF,V2_N,PV1,PV2,PV3,PV4,...
    INT1_NF,INT2_N);
%
% Make specifications by removing illigal states.
%
K3=sys_NF1;
Ind3=find(states_NF1(:,1)==2 & states_NF1(:,2)==2);
%
for i=1:length(Ind3)
    K3.TL=K3.TL(K3.TL(:,1)≠Ind3(i),:);
    K3.TL=K3.TL(K3.TL(:,3)≠Ind3(i),:);
end
%
[sys_NF2,states_NF2]=sync(P1_M1,P2_M1,V1_N,V2_NF,PV1,PV2,PV3,PV4,...
    INT1_N,INT2_NF);
%
% Make specifications by removing illigal states.
%
K4=sys_NF2;
Ind4=find(states_NF2(:,1)==2 & states_NF2(:,2)==2);
%
for i=1:length(Ind4)
    K4.TL=K4.TL(K4.TL(:,1)≠Ind4(i),:);
    K4.TL=K4.TL(K4.TL(:,3)≠Ind4(i),:);
end
%
sys=Gunion(sys_M1_N,Gunion(sys_M2_N,Gunion(sys_NF1,sys_NF2)));
K=Gunion(K1,Gunion(K2,Gunion(K3,K4)));
%
S=supRCN(K,sys,[101,100,201,200,13,23],sys_M1_N,sys_M2_N,...
    sys_NF1,sys_NF2);

%% Solving problem using conventional supervisory control.
%
SC=supcon(K,sys,[101,100,201,200,13,23]);
%
```

```
sys.Xm=intersect(sys_M1_N.Xm,intersect(sys_M2_N.Xm,...
                              intersect(sys_NF1.Xm,sys_NF2.Xm)));
%
KI=sys;
%
SI=supcon(KI,sys,[101,100,201,200,13,23]);
```

deck2gviz: constructs a .GV file of an automaton so it can be visualized by GraphViz

```

function deck2gviz(G, eventsLabels,statesLabels,fontSize)

% DECK2GVIZ   Reads an automaton object G and save a G.GV file which
%             constructs the automaton in DOT language to be ...
    visualized
%             in Graphviz
%
% SYNTAX:   deck2gviz(G)
%           deck2gviz(G,eventsLabels)
%           deck2gviz(G,eventsLabels,statesLabels)
%           deck2gviz(G,eventsLabels,statesLabels,fontsize)
%           deck2gviz(G,[],[],fontsize)
%           deck2gviz(G,eventsLabels,[],fontsize)
%
%
% INPUTS:   G                 input automaton object
%           eventsLabels      struct('event',{...},'label',{...})
%                             a structure containing events and labels
%           statesLabels      struct('state',{...},'label',{...})
%                             a structure containing states and labels
%           fontsize          labels font size
%
% OUTPUTS:  GraphViz .GV file
%


TL=G.TL;
Xm=G.Xm;
```

```

if nargin==1
    [TL_labeled,Xm_labeled]=num2label(TL,Xm);
    fontsize='14';
end
%
if nargin==2
    [TL_labeled,Xm_labeled]=num2label(TL,Xm,eventsLabels);
    fontsize='14';
end
%
if nargin==3
    [TL_labeled,Xm_labeled]=num2label(TL,Xm,eventsLabels,statesLabels);
    fontsize='14';
end
%
if nargin==4 && ¬isempty(eventsLabels) && ¬isempty(statesLabels)
    [TL_labeled,Xm_labeled]=num2label(TL,Xm,eventsLabels,statesLabels);
    fontsize=num2str(fontSize);
elseif nargin==4 && ¬isempty(eventsLabels) && isempty(statesLabels)
    [TL_labeled,Xm_labeled]=num2label(TL,Xm,eventsLabels);
    fontsize=num2str(fontSize);
elseif nargin==4 && isempty(eventsLabels) && isempty(statesLabels)
    [TL_labeled,Xm_labeled]=num2label(TL,Xm);
    fontsize=num2str(fontSize);
end
%
% open a file named "G.GV" for writing
%
filename=[inputname(1) '.GV'];
fid=fopen(filename,'w');

fprintf(fid,'digraph FSA{\n'); % name the graph as FSA

if ¬fontsize
    fprintf(fid,'node[fontsize=0];\n');
end

fprintf(fid,'rankdir=LR;\n'); % sets direction of graph layout
fprintf(fid,'node [shape = none,fontcolor=white];0;\n');
```

```

if ¬isempty(Xm) % displayes edges of graph layout
   for i=1:size(Xm,2)
    fprintf(fid,...
    'node [shape = ...
        doublecircle,fontcolor=black,fontsize=%s];%s;\n',...
        fontsize,num2str(Xm_labeled(i).state));
   end
end
%
% sets initial state's name
%
if nargin<3 || (nargin≥3 && ¬isstruct(statesLabels))
    IS=num2str(1);
else
    IS=statesLabels(1).label;
end
%
% marks initial state with an arrow
%
fprintf(fid,...
    'node [shape = circle,fontcolor=black,fontsize=%s];\n',fontsize);
fprintf(fid,'0 -> %s [labelsize = 0];\n',IS);
%
% adds transitions to graph layout
%
if ¬isempty(TL_labeled) && ¬isempty(TL)
    for i=1:size(TL_labeled,2)
    fprintf(fid,'%s -> %s [ label = ...
        "',num2str(TL_labeled(i).source)...
            ,num2str(TL_labeled(i).sink));
       for j=1:size(TL_labeled(i).event,2)-1
           fprintf(fid,'%s,',num2str(TL_labeled(i).event{j}));
       end
       fprintf(fid,...
           '%s ",fontsize=%s];\n',...
           num2str(TL_labeled(i).event{end}),fontsize);
   end
end
%
```

```
fprintf(fid,'}');
%
fclose(fid); % closes the file

end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function TL_merged=merge(TL) % marges parallel edges into a ...
    single edge
%
temp=struct('source',[],'event',[],'sink',[]);
%
i=1;
X=[];
while ¬isempty(TL)
%
 clear X;
%
 for j=1:size(TL,1)
     if isequal(TL(1,1),TL(j,1)) && isequal(TL(1,3),TL(j,3))
         X(j,:)=ones(1,3);
     else
         X(j,:)=zeros(1,3);
     end
 end
%
TL_temp=TL.*X;
TF = TL_temp==0;
TFrow = ¬all(TF,2);
TL_temp=TL_temp(TFrow,:);

if ¬isempty(TL_temp)
temp(i).source=TL_temp(1,1);
temp(i).sink=TL_temp(1,3);
temp(i).event=TL_temp(:,2);
end
%
TL_new=TL.*¬X;
%
TF = TL_new==0;
```

```
TFrow = ¬all(TF,2);
TL_new=TL_new(TFrow,:);
%
i=i+1;
TL=TL_new;
end
%
TL_merged=temp;
%
end
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [TL_labeled,Xm_labeled]=num2label(TL,Xm,varargin)
%
% If the event labels are specified then label the events ...
    accordingly;
% otherwise use the event numbers as the ...
    labelsTL_new=cell(size(TL'));
%
Xm_new=cell(size(Xm'));
%
if length(varargin)==2
   statesLabels=varargin{2};
   eventLabels=varargin{1};
elseif length(varargin)==1
        eventLabels=varargin{1};
elseif isempty(varargin)
        if ¬isempty(TL)
            events=unique(TL(:,2));
            eventLabels=struct('event',{},'label',{});
            %
            for i=1:size(events,1)
                eventLabels(i).event=(events(i));
                eventLabels(i).label=(events(i));
            end
        else
            eventLabels=struct('event',{},'label',{});
        end
end
%
```

```
% If the state labels are specified then label the states ...
    accordingly;
% otherwise use the state numbers as the labels
%
for i=1:size(TL',1)
    for j=1:size(TL',2)
        TL_new{i,j}=TL(j,i);
    end
end
%
for i=1:size(Xm',1)
    for j=1:size(Xm',2)
        Xm_new{i,j}=Xm(j,i);
    end
end
%
TL_strc=merge(TL);
%
if ¬isempty(Xm_new)
    Xm_strc=cell2struct(Xm_new','state');
else
    Xm_strc= struct('state',[]);
end
%
temp=TL_strc;
Xm_temp=Xm_strc;
%
for i=1:size(temp,2)
    temp(i).event={};
end
%
for i=1:size(eventLabels,2)
    for j=1:size(TL_strc,2)
        for k=1:size(TL_strc(j).event,1)
            if eventLabels(i).event==TL_strc(j).event(k)
                temp(j).event(k)={eventLabels(i).label};
            end
        end
    end
end
```

```
%
if length(varargin)==2
   for i=1:size(statesLabels,2)
       for j=1:size(TL_strc,1)
           if statesLabels(i).state==TL_strc(j).source
              temp(j).source=statesLabels(i).label;
           end%
           if statesLabels(i).state==TL_strc(j).sink
              temp(j).sink=statesLabels(i).label;
           end
       end
    end
%
for i=1:size(statesLabels,2)
    for j=1:size(Xm_strc,1)
        if statesLabels(i).state==Xm_strc(j).state
            Xm_temp(j).state=statesLabels(i).label;
        end

    end
end
%
for i=1:size(statesLabels,2)
    for j=1:size(TL_strc,2)
        %
        if statesLabels(i).state==TL_strc(j).source
            temp(j).source=statesLabels(i).label;
        end
        %
        if statesLabels(i).state==TL_strc(j).sink
            temp(j).sink=statesLabels(i).label;
        end
    end
end
end
%
TL_labeled=temp;
Xm_labeled=Xm_temp;
%
end
```