Online Robust Nonblocking Supervisory Control of Discrete-event Systems

Xiao Yong Chen

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science (Electrical Engineering) at
Concordia University
Montreal, Quebec, Canada

December, 2007

Your file Votre référence
ISBN: 978-0-494-34588-7
Our file Notre référence
ISBN: 978-0-494-34588-7

Abstract

Online Robust Nonblocking Supervisory Control of Discrete-event Systems

Xiao Yong Chen

In this thesis, robust nonblocking supervisory control problem (RNSCP) is studied in which the plant model belongs to a family of models. A drawback of the existing solutions in the literature is that the entire supervisor which is designed offline needs to be stored in computer memory for implementation.

In this thesis, we develop an online solution in the form of a variable lookahead policy (VLP). In this approach, the supervisory commands are computed online when the plant is in operation, based on the behavior of the plant models in the near future (lookahead window). In this way, only the plant models need to be stored in computer memory (which can be usually done more efficiently than that of a supervisor).

To derive our VLP algorithm, we develop a direct offline solution to RNSCP which is more transparent than the indirect method in the literature. As an illustrative example, we have used our direct method to solve supervisory control problem with multiple markings in which the plant under supervision is required to be able to reach any of the multiple sets of marked states, each corresponding to the completion of a different task.

In some problems, the VLP lookahead window may become unbounded and the VLP algorithm cannot be applied. To relieve the limitation, state-based RNSCP (RNSCP-S) is formulated and solved using state-based VLP (VLP-S). If a RNSCP cannot be solved using VLP, it can be solved using VLP-S as a corresponding RNSCP-S after automaton refinement.

Acknowledgements

I would like to take this opportunity to extend my profound gratitude to my supervisor, Dr. Shahin Hashtrudi Zad. Without his instruction and examination, I could not get my research and this thesis done with a desirable quality.

I also want to express my heartfelt thankfulness to my wife, Guo Hong Zhang. Without her encouragement and support, I could not pursue my master studies and devote to the research and thesis work.

I dedicate this thesis to my parents and my brother in China. Their unconditional support enabled me to be an educated and dependable person and thus be successful in academic, professional, and personal life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis studies robust nonblocking supervisory control of discrete event systems with model uncertainty, specifically, the exact system model is uncertain but could be one of a set of possible models. The solutions to the robust control problem in the literature are calculated at the design stage (offline) which requires a large amount of memory for implementation. To tackle this issue, an online solution is developed in this thesis which makes control decision for the current state at run time without storing supervisors.

In this chapter, we first introduce discrete event systems and language modeling in Section 1.1. Then conventional supervisory control with precise model is introduced in Section 1.2 followed by robust supervisory control with model uncertainty in Section 1.3. Fault recovery problem is introduced as a special application of robust supervisory control. After the literature review on supervisory control and robust supervisory control in Section 1.4, the objectives and contributions of this thesis are presented in Section 1.5. In the end, the organization of this thesis is described in Section 1.6.

1.1 Discrete Event Systems

Discrete-event systems (DES) [5] [38] are systems with discrete states. State transitions are driven by events. Since state transitions are driven by events, we can use the sequences of events to describe the system dynamic behaviors. A sequence that a system can conduct represents a sample path of the dynamics of the system.

Example 1.1: Take a simple propulsion system shown in Figure 1.1 as an example. The system consists of a fuel tank *T*, a valve *V*, and an engine *E*. We assume the tank always has sufficient fuel. The mechanism of the system is that the engine will be on if the valve is turned on, and engine will be off when the valve is turned off. The system can be modeled as a DES.

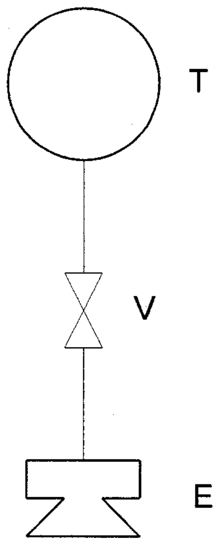

Figure 1. 1: A simplified propulsion system: Example 1.1

The states of the system are discrete. Suppose the tank always has sufficient fuel and thus there is only one state for the tank which is tank full. The valve has two states which are on and off. The engine also has two states on and off. The overall states of the system can be represented by the combination of component states. For example, (Tank full, Valve on, Engine off) is one of the overall system states.

If we associate events to state transitions, state transition of a system can be viewed as driven by a finite number of events. The events in our propulsion example include: *turn* *on valve*, *turn off valve*, *engine fires*, and *engine stops*. For example, if an event *engine fires* happens at the state (Tank full, Valve on, Engine off), then the system state transits to (Tank full, Valve on, Engine on) driven by the event *engine fires*.

A sample path of the system can be the following. At the initial state (Tank full, Valve off, Engine off), if the valve is turned on, then the engine will fire. After that, if the valve is turned off, then the engine will stop firing. The sample path can be represented as an event sequence: turn on valve, engine fires, turn off valve, and engine off. All of the sequences that the system can conduct will form a set of sequences which represents the overall dynamics of the system. □

If we treat the set of events as an alphabet, then a sequence will correspond to a word formed from the alphabet. Similarly, the set of all feasible sequences in a system can be treated as the language formed from the words that the system speaks. Different systems own different alphabets and different words. Hence, the dynamics of a system can be modeled exactly as the language the system speaks.

We mark all of the sequences that indicate that tasks are fulfilled. If any string of a system can end with a marked sequence, then some task can be always achieved. In this case, we call the system nonblocking. The above simple propulsion system is nonblocking if sequences with engine on are marked since we can always bring a system from any state to a marked state with engine on.

1.2 Supervisory Control of Discrete Event Systems

Some sequences of a system may be undesirable. For example, the engine has to be warmed up first before the fuel valve is opened in Example 1.1. Therefore, opening valve before engine is warm is undesirable. Hence, we want to design a controller which ensures that the valve is not turned on before the engine is warm.

Supervisory control is a feedback control to meet this kind of requirements shown in Figure 1.2. Supervisors observe sequences of a system and disable events if necessary. In the above example, when the event *engine stops* is observed by a supervisor, event *turn on the valve* is not allowed at the current state. Only when an event *warm up engine* is observed is the event *turn on the valve* enabled by the supervisor.

Supervisory control is passive feedback control in that it only disables some controllable events to prevent systems from generating illegal strings and being blocked without forcing any event to happen. In the above example, events *turn on the valve* and *turn off the valve* are controllable. However, *engine fires* and *engine stops* are uncontrollable.

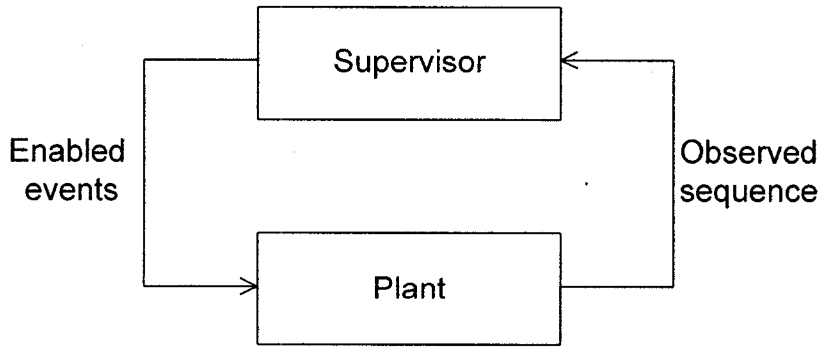

Figure 1. 2: The schematic of supervisory control

Supervisory Control [5] [38] aims to restrict the system behavior inside the safety specification. Nonblocking supervisory control ensures the closed-loop system is nonblocking in addition to ensuring safety specification. Offline supervisory control synthesizes supervisors in advance for an overall system. Offline supervisor is calculated once before system operation and stored as a lookup table for referring to at run time.

In linguistic supervision, the control domain is in terms of sequences. System states are used only for representation of system languages. As a dual procedure of linguistic supervision, in state-based supervision the specification is given in terms of desirable states (instead of sequences of events in the linguistic approach). A state-based problem can also be regarded as linguistic problem in which the specification automaton is a subautomaton of the plant automaton.

The main issue which impedes the application of supervisory control of discrete event systems is the difficulty of modeling, computational complexity due to the state explosion of system and specification automata, and difficulty of storing (offline) supervisors.

Online supervisory control [7] has been proposed to solve the problem. Online supervisory control examines the system's future behavior over a finite horizon without modeling the overall plant model and computes the control decision only for the current string at run time. When the system evolves one step ahead, the calculation is repeated for the new current state. As a result, the overall system model and specification model are unnecessary which avoids the state explosion problem of modeling. Moreover, no supervisor is required to be stored for looking up, which reduces space complexity. In addition, unlike offline supervisory control, online supervisory can be also applied for time-varying systems.

1.3 Robustness and Fault Recovery

The traditional supervisory control assumes the system model is certain. However, system environment is always changing, for example, due to reconfiguration of system modules, failure of some components, etc.

There are different ways to model system uncertainty. A natural way to model system uncertainty is to model each possible situation with a certain model and thus the system is represented with a set of possible models. In addition, there may be a unique specification for each possible model. The precise system model may or may not be certain at one time before a system runs. For example, we know the exact model when a certain configuration scheme is chosen in a reconfigurable system. However, we do not know if a failure will happen or not in a system.

In case that the precise model is known at the beginning of one operation, we may design a supervisor using the conventional supervisory control for each possible model and choose the corresponding supervisor before each operation. The disadvantage is that we have to design a supervisor for each possible system model and thus computationally expensive if there are several possible models. Therefore, we may wish to design a unique robust supervisor which works properly for all possible system configurations.

In case that we do not know the exact model at the start of an operation, traditional supervisory control will be inapplicable. From now on, we assume that the precise system model is uncertain before an operation and aim to design an optimal robust supervisor which meets safety requirement and ensures nonblocking property for all possible plant models.

Offline robust supervision in discrete event systems has been investigated by researchers [16] [3] [27]. In this approach, the union behavior of all possible models and the overall specification is required to be obtained before a supervisor can be synthesized and thus we refer to it as the *indirect approach.*

However, online robust supervisory control using lookahead policy has not yet been developed to the best of our knowledge. We develop a new offline robust supervisory control with a direct approach, which is more transparent and offers insight to robust supervision problems. We refer to this approach as the direct approach since it characterizes the solutions of the robust control problem in terms of sublanguages of each plant model (rather than those of a union model).

Fault recovery and fault tolerance supervision in discrete event systems becomes more important with the increase of the system complexity. We require that a system can conduct its full function and be nonblocking at the normal operation. In addition, if some components fail, the system shall fulfill all or part of functions and not block.

The safety specification may vary before and after failure happens. For example, the specification after failure may be less strict than the requirements when the system status is normal. The questions to investigate include: the existence of a robust supervisor for a given system with failures and methods for the synthesis of the supervisor.

Conventional supervisory control will not directly solve fault recovery problems. A supervisor designed for the normal model ceases working if a failure happens. As a result, the system may violate the specification or become blocking after failure. On the other hand, a supervisor designed for the entire system model with failure events may result in blocking in the normal mode.

Fault recovery problems can be solved as a special case of robust supervision problems [26]. We know the system normal model without failure and the normal-failure model if a specific failure happens; but we do not know if any failure will happen or which failure will happen. The system normal model and all normal-failure models form the set of possible system models. The following example illustrates a fault recovery problem in a simplified propulsion system.

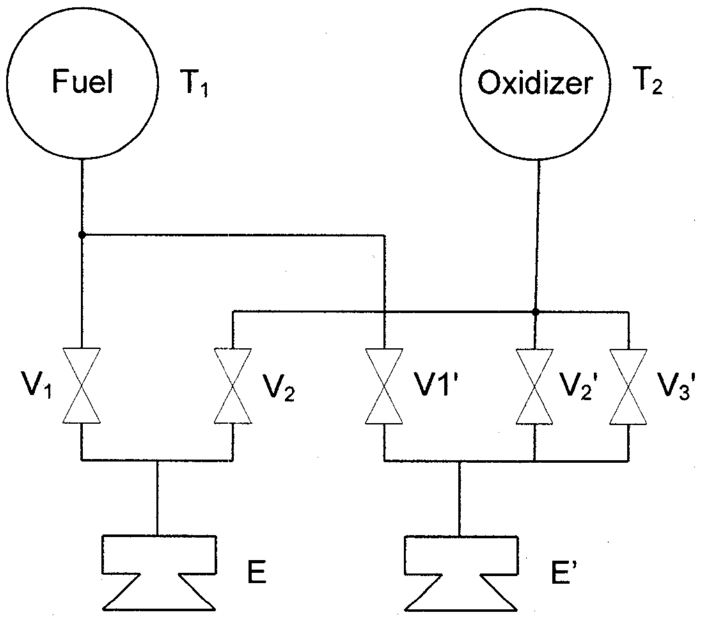

Figure 1. 3: A propulsion system: Example 1.2

Example 1.2: A propulsion system with one fuel tank T_1 one oxidizer tank T_2, five valves (V_1, V_2, V_1', V_2', V_3'), and two engines (E and E'). We assume the tanks always have sufficient fuel and oxidizer. Engine E will be on if V_1 and V_2 is on. Engine E' will be on if V_1' and V_2' (or V_3') are turned on. V_2' is used in the normal status and may become stuck-closed. In the normal operation, V_3' shall not be used. Assume that a tank can supply one engine at a time and thus V_1 and V_1' (V_2, V_2',and V_3') shall not be turned on at the same time. In addition, we assume that a valve can not be turned off until the engine connected to it fires. We mark the states where some engine is fired (i.e. we want to be able to start the engine using an appropriate sequence of events). The initial state is that all valves are turned off and both engines are off.

The intuitive supervision to the system will be as follows. In the normal mode, V_3' is disabled by a supervisor according to the specification, and V_1 and V_2 (resp. V_1' and V_2') are used to turn on and off the engine E (resp. E'). If V_2' fails stuck-closed at the state where engine E' is on, then E' will be off. In addition, V_1' is not allowed to be turned off when engine E' is off. In this case, V_1 is not allowed to be turned on since V_1' is on, and thus engine E will also not be fired. As a result, no engine will be on if the supervisor keeps disabling V_3' and thus the system becomes blocking in the failure mode. Therefore, V_3' shall be enabled after the failure to get the system recovered. □

1.4 Literature Review

Supervisory control and robust supervisory control have been studied by researchers. We briefly review the relevant results and give comments on them.

1.4.1 Supervisory Control

Supervisory control of discrete event systems is first developed in [23] [24]. The events of a system are partitioned into controllable and uncontrollable events. A supervisor is defined as a function which maps strings of a system to enabled events. A supervisor is constructed as a model which represents the desirable behavior. The interconnection of the system with the supervisor forms the closed-loop system which only generates the legal strings.

Since only controllable events can be disabled, a supervisor is admissible only if all uncontrollable events are enabled by it. Controllability theorem shows that a supervisor is admissible if and only if the closed-loop behavior is controllable with respect to the open-loop behavior.

In supervisory control problems, a string is marked in the closed-loop system as long as it is marked in the plant. Hence, if a marked string s is allowed in the closed-loop system and string s is also a marked string in the plant, then string s will be marked in the closed-loop system. Nonblocking can be guaranteed in the closed-loop system only if the marked language of a closed-loop system is relative-closed with respect to the open-loop marked language.

The existence problem for a supervisor is reduced to calculating the supremal controllable sublanguage of a given specification. The calculation is further investigated in [37] by the authors. The supremal controllable sublanguage is characterized as the largest fixed point of an operator. There is no closed-form expression to compute the result. Instead, the supremal controllable sublanguage is calculated recursively with an algorithm. It is proved that the algorithm converges in a finite number of iterations if the system and specification are modeled by finite-state automata.

Limited lookahead supervisory control (LLP) is first proposed in [7]. LLP predicts the plant behavior in certain steps and makes control decision only for the current string. The procedure is repeated after some enabled event happens.

Conservative or optimistic attitudes are taken to accommodate the uncertainty of the system behavior after the limited lookahead window. Conservative attitude may disable an event even if it can be enabled. As a result, the closed-loop behavior with conservative limited lookahead policy is never more permissive than the optimal offline solution. Similarly, an event that shall be disabled may be enabled if optimistic attitude is taken. This will cause error since illegal sequences or blocking may become unavoidable later. As a result, the closed-loop behavior with optimistic lookahead policy is never more conservative than the optimal offline solution.

The behavior of a system under supervision of a lookahead supervisor may be different than the behavior of the system under supervision of a conventional optimal supervisor. A larger lookahead window results in a better closed-loop behavior in that it is closer to the offline optimal solution. In some systems, the closed-loop behavior under supervision of an online supervisor will be equivalent to the offline supremal solution if a sufficiently large lookahead window is taken. We call such an online supervisor valid. Nevertheless, this is not always the case since sometimes the equivalence is never obtained no matter how long the size of the window is even when plant and specification are finite-state automata.

The lookahead window is represented by a finite tree. The computation of the supremal controllable sublanguage with respect to the lookahead window is transferred to an optimal problem in [9] in order to recursively use the results of the previous setup at each step and thus reduce computational complexity. Each state of the tree is assigned a value with either 0 or ∞, and the control decision for the current string is made according to the value of the current state and all of its next states. The values for the boundary states are first assigned and their values are propagated backwards to the root state.

In [10], the authors observe that values of some states in the lookahead window are not necessary to make a control decision for the current string. The authors propose a forward searching algorithm which only assigns values if necessary. As a result, it improves the efficiency of computation. This algorithm is called variable lookahead policy (VLP).

The above references take a linguistic approach which works on languages. Variable lookahead supervisory control with state information (VLP-S) [1] takes state-based approach with variable expansion. It requires that the specification be modeled as a subautomaton of the plant automaton. In contrast to the linguistic approach, the online supervisor with VLP-S is always valid.

Extension Based lookahead policy [14] estimates the behaviors of a plant after the N-step lookahead window. This approach is in fact similar to the conservative approach in the sense that the resulting supervisor is never more permissive than the offline optimal solution. Estimate-based lookahead policy for closed language specification is considered in [28]. Online supervisory control under partial observation is considered in [2] [12].

In this thesis, we take the same spirit of VLP to solve robust problems without transferring to optimal problems. We also investigate VLP-S in robust problems without assigning values to states.

1.4.2 Robust supervisory control

Lin [16] first considers systems with a set of possible models. This framework of modeling uncertainty is natural in the sense that a system with uncertainty may not be modeled by a single model. The specification is assumed to be a sublanguage of the marked languages of all possible plant models. Robust supervision in this formulation aims to find a unique supervisor which is suitable for all possible models. The performance of a solution is compared in terms of the closed-loop languages. The union of all possible behaviors is first synthesized. There exists a solution to the robust problems if and only if a sublanguage of the specification can be found that is controllable and observable with respect to the "union" plant behavior. Park and Lim [20] [21] generalize Lin's work to nondeterministic automata.

Takai [29] [30] relaxes the assumption that the specification is a sublanguage of each marked language. However, it only considers prefix-closed languages. An overall specification is given as a language and the specification for each possible plant model is obtained by taking the intersection of the overall specification with the possible plant model.

Bourdon [3] generalizes Takai's work to marked languages. Nonconflicting is an additional condition for the existence of a solution. In addition, a separate specification is considered for each possible plant model instead of being given as an overall specification for all possible plant models. The overall specification is synthesized from the individual specifications. The solution to the robust problem is to find a sublanguage of the synthesized overall specification that is controllable and relative-closed to the overall plant behavior as well as nonconflicting to the marked behavior of each possible plant model.

It turns out that the nonconflicting condition used in [3] to ensure nonblocking property of the closed-loop systems is only necessary and not sufficient. Saboori [27] generalizes Bourdon's work to partial observation and replaces nonconflicting with G-nonblocking condition to remove the above limitation, and shows that G-nonconflicting is a necessary and sufficient condition for the existence of a robust solution.

There are two other paradigms of robust supervisor control in the literature which are in terms of languages. Cury and Krogh [11] and Takai [31] [32] model the uncertainty of a system with ω-languages. Given a supervisor which works for a nominal plant, it aims to maximize the set of plants for which the supervisor is suitable. Park and Lim [19] [22] model the uncertainty of a system with internal unobservable transitions. This approach is also based on a nominal plant.

Our work models system uncertainty as done by Lin in [16] and formulates robust supervisory control problem as in Bourdon [3] and Saboori [27]. Without synthesizing the overall plant behavior and the overall specification, we work on the specification and plant pairs directly to solve the problem. A concept of *consistency* is defined to offer insight to the robustness property. In addition to solving robust supervisory control problem with direct approach offline, we also address the linguistic and state-based robust problem with direct approach online by generalizing traditional online supervisory control to accommodate robustness.

The objective in fault recovery problems is to ensure that the plant under supervision meets its specification in faulty modes (as well as normal modes). The specification for faulty behavior may not be as stringent as those of normal operation.

One way of fault recovery is to model the plant with faults as a finite state automaton. The states of a plant automaton are divided into normal states and failure states. The part of the automaton with all normal states forms the normal model. The normal states and all states after one kind of failures form a normal-failure model. There will be many normal-failure models if different failures exist.

Modular switching supervisor control [18] [34] designs a supervisor for the normal model and a supervisor for each normal-failure model. The supervisors can be configured to control the system in the normal mode and each failure mode. This approach simplifies supervisor design and implementation complexity. However, it may result in blocking when the individual supervisors work together as modular supervision. In addition, the modular supervisor designed is not guaranteed to be the most permissive supervisor.

Another approach [26] is to convert fault recovery problems to robust problems. In this approach, a single robust supervisor is designed systematically which is suitable for the normal model and all normal-failure models. The necessary and sufficient conditions for the existence of solutions to fault recovery problems are obtained in [26].

We use fault recovery as an example of robust nonblocking supervisory control to illustrate our results in state-based robust supervision problem. Hence, we take the second approach of solving fault recovery problems as robust problems.

1.5 Thesis Objectives and Contributions

The indirect approach to robust problem does not provide insight to the key issue of robust problems. This thesis develops an equivalent offline algorithm in Chapter 3, which does not need to synthesize the overall specification and plant. It directly deals with each plant model and specification pair and thus is referred to as the direct approach. A concept of consistency is defined which is the key issue to the robustness property. The necessary and sufficient conditions for the existence of solutions to robust problems are developed. Algorithms are proposed to compute optimal solutions to robust problems. As an illustrative example, nonblocking control problem with multiple markings is investigated.

The drawback of offline robust supervision is that if the possible models (such as failure modes in fault recovery problems) become numerous, the resulting supervisor may become too large to be stored in computer memory. One method to deal with this problem is to develop online methods that compute control commands online. This thesis aims to offer a transparent direct solution to both linguistic and state-based robust problem and implement it online.

In Chapter 4, linguistic online robust supervision with direct approach is developed and its validity is proved. We only generate control action for the current system state. When the system evolves, the algorithm is repeated to calculate the next control action. There is no need to store any supervisor and thus the supervision implementation is simple.

Traditional limited lookahead policy (LLP) expands each branch to a given step. In fact, the expansion for a branch can be terminated before the given step is reached in some circumstance. For example, the behavior after an illegal state or a marked state without uncontrollable active events has no influence on control decision for the current string.

We generalize LLP to robust supervision problems. Instead of using LLP directly, we modify LLP to VLP by only expanding the part of the system models that are necessary for making the control decision for the current state.

Linguistic online solution to robust problem works only if expansion always terminates. To solve the termination problem, in Chapter 5, a state-based robust problem is formulated and an algorithm is proposed that always finds an optimal supervisor. It is shown that any linguistic robust problem can be converted to a corresponding state-based robust control problem after automaton refinement.

We apply our results of state-based robust supervision to fault recovery problems of a simplified spacecraft propulsion system. A simplified propulsion system with failure is modeled as a discrete event system. The fault recovery problem is solved as a state-based robust problem manually. In addition, the procedure of solving the fault recovery problem with TTCT [33] is provided.

The main contributions of this thesis can be listed as follows.

- A direct approach to robust control problem that is more transparent than the existing indirect method.
- Application of the direct approach to supervisory control with multiple markings.
- A VLP for online implementation of the direct approach.
- A VLP-S for robust control to resolve the termination issue of VLP.

1.6 Thesis Organization

Supervisory control of DES and its online implementation using LLP is reviewed in Chapter 2. Also, robust control problem and its indirect solution are examined in this chapter. Application of RNSCP to fault recovery is also reviewed.

In Chapter 3, we propose a direct robust supervision procedure which offers insights to robust problems and relaxes the necessity of synthesizing the overall specification and plant. Nonblocking supervisory control problem with multiple markings is formulated and solved as a special case of robust problem using our direct approach.

Online synthesis procedure of the new direct approach is further investigated in Chapter 4. A variable lookahead expansion policy VLP is given for online calculations of robust control and its validity is proved.

In Chapter 5, RNSCP-S is formulated and an online algorithm VLP-S is proposed to solve it. A procedure for converting RNSCP to RNSCP-S is proposed. Our results are applied to a fault recovery problem of a simplified spacecraft propulsion system. A TTCT procedure for robust supervisor synthesis is also provided there.

Chapter 6 summaries our work and discusses possible directions for future research.

Chapter 2:

Background

The behavior of discrete-event systems can be represented using languages. However, it is difficult to analyze language models directly and synthesize controller based on languages. To assist system analysis and design, automata are usually used to represent languages.

Supervisory control aims to design a supervisor that restricts a system's behavior in order to satisfy design specifications by disabling controllable events. A supervisor can be synthesized offline or online.

In this chapter, we review some of the background material that will be later used in this thesis. In Section 2.1, modeling of discrete-event systems is introduced. Then, offline supervisory control is reviewed in Section 2.2. After that, online supervisory control is presented in Section 2.3. Finally, robust supervisory control is reviewed in Section 2.4.

2.1 Modeling of Discrete-event Systems

Discrete-event systems (DES) are systems that have discrete states and their state transitions are driven by events asynchronously. Most man-made systems such as telecommunication systems, manufacturing systems, and computer systems can be modeled as discrete-event systems. Physical continuous-variable systems can also be abstracted as discrete event systems.

2.1.1 Languages

An event of a discrete-event system takes the current system state to the next state immediately. We assume only one event happens at a time.

A **sequence** of events can be treated as a sample trajectory of system behavior and thus the dynamics of a discrete-event system can be represented as a set of sequences. A sequence is also called a **string**, a **trace**, or a **word**. The string without any event is called the empty string and denoted as ε. The basic operation on strings is catenation of two strings. For example, the catenation of two strings u and v forms a new string uv. We denote the length of a string s as $|s|$.

If we treat the set of events as an alphabet, then a sequence will be equivalent to a word formed from the alphabet. Similarly, a set of sequences will correspond to a **language** with words formed from the alphabet. The behavior of a discrete-event system can be interpreted as the language that it speaks.

Definition 2.1: Language Any set of finite-length strings formed from the events in Σ.□

We denote the language that includes the empty string and all possible finite strings from the event set Σ as Σ^*. A language is a set of sequences, and thus the basic set operations of two languages L_1 and L_2 such as intersection ($L_1 \cap L_2$), union ($L_1 \cup L_2$), complement (L_1^{co}), and relative complement ($L_1 - L_2$) apply to languages. In addition, other operations on languages such as prefix-closure, post-language, truncation, right quotient are also defined and will be discussed later.

Consider a string $s = uvt$ which is formed from catenation of three strings u, v, and t. In this case, u is called a **prefix** of s; v is called a **subtrace** of s; and t is called a **suffix** of s. The set of all prefixes of strings in a language L forms a new language denote as $\overline{L}$.

Definition 2.2: Prefix-closure Let $L \in \Sigma^*$, then $\overline{L} := \{s \in \Sigma^* : \exists t \in \Sigma^*, st \in L\}$. □

If a string s is a prefix of string t, then we write $s < t$. Since all strings in L belong to $\overline{L}$, we always have $L \subseteq \overline{L}$. If the inverse inclusion also holds $L \supseteq \overline{L}$, then we will have $L = \overline{L}$ in which case, we say L is prefix-closed.

The post-language of a language L after a string s consists of the set of the strings that the catenation of s with them belongs to L.

Definition 2.3: Post-language The post-language of $L \subseteq \Sigma^*$ after string $s \in \Sigma^*$ is denoted as L/s and defined as: $L/s := \{t \in \Sigma^* : st \in L\}$. □

The truncation of a language $L \subseteq \Sigma^*$ to a natural number N consists of strings of L that have a length less than or equal to N.

Definition 2.4: Truncation $L|_N := \{t \in L : |t| \leq N\}$. □

Definition 2.5: Right Quotient The right quotient of $L \subseteq \Sigma^*$ by $M \subseteq \Sigma^*$ is denoted as L/M and defined as $L/M := \{s \in \Sigma^* : \exists t \in M, st \in L\}$. □

The following lemmas will be used later in this thesis.

Lemma 2.1: [7] [8] Let $K \subseteq \Sigma^*$, then $\overline{K}/s = \overline{K/s}$ for any $s \in \Sigma^*$. □

Lemma 2.2: [7] [8] Let $A, B \subseteq \Sigma^*$, then $(A \cap B)/s = A/s \cap B/s$ for any $s \in \Sigma^*$. □

Lemma 2.3: [7] [8] Let $A, B \subseteq \Sigma^*$. If $s \in \overline{A}$, then $AB/s = (A/s)B$. □

Lemma 2.4: [27] Let $A, B \subseteq \Sigma^*$. Then, the following three statements are equivalent.

1) $\overline{A} \cap B = \Phi$

2) $\overline{A} \cap B\Sigma^* = \Phi$

3) $A \cap B\Sigma^* = \Phi$. □

It is difficult to study discrete-event systems using only languages. To solve this problem, other modeling methods such as automata and Petri nets have been proposed to represent languages. We only discuss automata here. An automaton is also called a generator.

2.1.2 Automata

State transition graph is perhaps the most direct way to define an automaton. Deterministic automata are automata that every event leads a state to only one state. The formal mathematical definition of a deterministic automaton is given below.

Definition 2.6: Deterministic Automaton A deterministic automaton G is a five–tuple:

$G = (X, \Sigma, \delta, x_0, X_m)$ where X is the set of states and Σ is the finite set of events. δ is the partial transition function from $X \times \Sigma$ to X. x_0 is the initial state and X_m is the set of marked states. It is the desirable subset of X. □

Note that at any state x, not all events may occur and therefore δ is a partial function. We denote the set of all events that are eligible to occur at the state of G reached by a string s, as $\Sigma_G(s)$ and call it the **active events** of string s.

The set of all possible sequences from the initial state to a state in the state transition graph forms a language, which is called the generated (the closed) language of the automaton. Similarly, the set of all sequences from the initial state to marked states forms the marked language.

Definition 2.7: Language Represented by an Automaton The language generated by $G = (X, \Sigma, \delta, x_0, X_m)$ is $L(G) := \{s \in \Sigma^* \mid \delta(x_0, s) \text{ is defined}\}$. The language marked by $G = (X, \Sigma, \delta, x_0, X_m)$ is $L_m(G) := \{s \in L(G) \mid \delta(x_0, s) \in X_m\}$. □

It is easier to operate on automata than on languages. The basic operations on automata are introduced below. We call a state unreachable if no string leads to it from the initial state. The reachable states of an automaton can be obtained by the following accessible part operation. The accessible part operation does not affect the languages generated and marked by an automaton.

Definition 2.8: Accessible Part: Let $G = (X, \Sigma, \delta, x_0, X_m)$. Then $Ac(G) := (X_{ac}, \Sigma, \delta_{ac}, x_0, X_{ac,m})$ where $X_{ac} = \{x \in X : \exists s \in \Sigma^*, \delta(x_0, s) = x\}$, $X_{ac,m} = X_m \cap X_{ac}$, and $\delta_{ac} = \delta \mid X_{ac} \times \Sigma \to X_{ac}$. □

The set of all states from which there exists a sequence to marked states are called coreachable (coaccessible). The following operation only keeps the coaccessible part of an automaton.

Definition 2.9: Coaccessible Part: Let $G = (X, \Sigma, \delta, x_0, X_m)$.

Then $CoAc(G) := (X_{coac}, \Sigma, \delta_{coac}, x_{0,coac}, X_m)$ where

$X_{coac} = \{x \in X : \exists s \in \Sigma^*, \delta(x,s) \in X_m\}$, $x_{0,coac} = \begin{cases} x_0, & \text{if } x_0 \in X_{coac} \\ \text{undefined}, & \text{otherwise} \end{cases}$, and

$\delta_{coac} = \delta \mid X_{coac} \times \Sigma \to X_{coac}$. □

Coaccessible part operation will result in the same marked language. The generated language of a nonblocking automaton will remain unchanged after coaccessible part operation. However, coaccessible part operation will end up with a smaller generated language if the original automaton is blocking.

Blocking happens in an automaton if there does not exist any trace leading from an unmarked reachable state to a marked state. Blocking could be in the form of deadlock or livelock. An automaton could reach an unmarked state with no active event. This is called deadlock because no further event can be executed at the unmarked state. The system has a livelock if there exists a strongly connected set of unmarked reachable states such that no event takes the system outside the set of unmarked states. The formal definition of blocking is given in terms of languages.

Definition 2.10: Blocking of Automaton An automaton G is called blocking if $\overline{L_m(G)} \subset L(G)$ and nonblocking if $\overline{L_m(G)} = L(G)$. □

Operation Trim is the combination of accessible part and coaccessible part operations. The automaton after trim operation will be both reachable and nonblocking. The accessible and coaccessible operations in the trim operation are commutative.

Definition 2.11: Trim: Let $G = (X, \Sigma, \delta, x_0, X_m)$.

Then $Trim(G) := CoAc[Ac(G)] = Ac[CoAc(G)]$. □

The complement operation of an automaton marks the complement language of the language marked by the original automaton.

Definition 2.12: Complement: If $G = (X, \Sigma, \delta, x_0, X_m)$, then G^{comp} marks $L_m(G^{comp}) = \Sigma^* - L_m(G)$ and generates Σ^*. □

The product operation $G_1 \times G_2$ represents the common language of the languages represented by the original automata. It represents the intersection of the languages of the original automata.

Definition 2.13: Product: Let $G_1 = (X_1, \Sigma_1, \delta_1, x_{01}, X_{m1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{02}, X_{m2})$.

Then $G_1 \times G_2 := Ac(X_1 \times X_2, \Sigma_1 \cap \Sigma_2, \delta, (x_{01}, x_{02}), X_{m1} \times X_{m2})$ where

$$\delta((x_1, x_2), \sigma) := \begin{cases} (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma)), & \text{if } \delta_1(x_1, \sigma)! \wedge \delta_2(x_2, \sigma)! \\ \text{undefined}, & \text{otherwise} \end{cases}. \quad \square$$

It follows that $L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2)$ and $L(G_1 \times G_2) = L(G_1) \cap L(G_2)$. The union operation of two languages can be realized by intersection and complement since $L_1 \cup L_2 = (L_1^{co} \cap L_2^{co})^{co}$.

The synchronous product $G_1 \| G_2$ represents the parallel behavior of systems. It can be used to obtain the overall system model from independent subsystems. If an event is a common event of G_1 and G_2, then it is defined at a state of $G_1 \| G_2$ only when it is active at the corresponding states of both automata. The next state will be the pair of the next states of the original automata. If an event is a private event of automaton G_1, it is defined at a state of $G_1 \| G_2$ as long as it is active in the corresponding state of G_1. The next state will be the pair of the next state of G_1 and the current state of G_2.

Definition 2.14: Synchronous Product (Parallel Composition):

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

Then $G_1 // G_2 := Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \delta, (x_{01}, x_{02}), X_{m1} \times X_{m2})$ where

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

In synchronous product, G_1 and G_2 have an equivalent role. A common event of G_1 and G_2 is not allowed in the synchronous product operation unless it is an active event of the current state of both automata. In contrast, one automaton of G_1 and G_2 in the biased synchronous product defined below acts as a leader in the state evolution in that state transition is permitted as long as an event is active at the current state of the leading automaton. As a result, the resulting automaton of biased synchronous product results has the same generated and marked languages as the leading automaton.

Definition 2.15: Biased Synchronous Product [15]: Given two automata $G_1 = (X_1, \Sigma_1, \delta_1, x_{01}, X_{m1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{02}, X_{m2})$, the multiple biased synchronous product of G_1 is defined as:

$G_1 \|_r G_2 := Ac(X_1 \times X_2, \Sigma_1, \delta, (x_{01}, x_{02}), X_{m1} \times X_2)$ where

$$\delta((x_1, x_2), \sigma) = \begin{cases} (x_1', x_2') & \text{if } \sigma \in \Sigma_{G_1}(x_1) \\ \text{undefined} & \text{otherwise} \end{cases} \text{ and } x_i' = \begin{cases} \delta_i(x_i, \sigma) & \text{if } \sigma \in \Sigma_{G_i}(x_i) \\ x_i & \text{if } \sigma \notin \Sigma_{G_i}(x_i) \end{cases}. \quad \square$$

If an automaton G_1 is a subgraph of another automaton G_2, then we say G_1 is a subautomaton of G_2. In this case, a state in G_1 corresponds to only one state in G_2. In other words, if two strings lead to the same state in G_1, they will lead to the same corresponding state in G_2.

Definition 2.16: Subautomaton: Let $G_1 = (X_1, \Sigma_1, \delta_1, x_{01}, X_{m1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, x_{02}, X_{m2})$. G_1 is called a subautomaton of G_2 denoted as $G_1 \sqsubseteq G_2$ if $\delta_1(x_{01}, s) = \delta_2(x_{02}, s)$ for all $s \in L(G_1)$. $\square$

Automata are very convenient for system modeling, analysis, and control. However, the number of states of the product of two automata may be the product of the number of the original automata. If there are n automata with m states, the product of them may have m^n states. This is called state explosion.

It should be noted that some languages can not be represented by automata with finite states. Only regular languages can be generated or marked by automata with finite states. The basic operations in this subsection on two finite-state automata will result in finite-state automata.

2.2 Offline Supervisory Control

Given an automaton $G = (X, \Sigma, \delta, x_0, X_m)$, it is assumed that the event set can be divided into two disjoint sets $\Sigma = \Sigma_c \,\&\, \Sigma_u$ where Σ_c is the set of controllable events and Σ_u is the set of uncontrollable events. We want to restrict the system language to a desirable sublanguage and have nonblocking property by disabling some controllable events. We call this kind of control, supervisory control. Figure 2.1 illustrates the configuration of supervisory control.

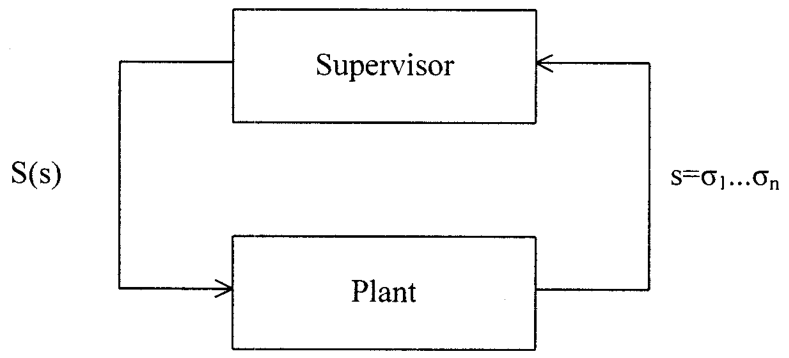

Figure 2. 1: Supervisory control

A supervisor S is a function $S : L(G) \to 2^{\Sigma}$. For $s \in L(G)$, $S(s)$ specifies the enabled events at some strings in $L(G)$. The control domain is based on strings and thus we call it **linguistic supervisory control**. The language generated and marked by the closed-loop system is denoted as $L(S/G)$ and $L_m(S/G) = L_m(G) \cap L(S/G)$.

Supervisory control of discrete event systems is a feedback control in that the supervisor observes the strings generated in the plant and decides which active events should be enabled at the current state. A supervisor shall not disable any uncontrollable event. A supervisor disabling only controllable events is called an admissible supervisor.

Definition 2.17: A supervisor S is called admissible if it does not disable any uncontrollable event in G.

2.2.1 Basic Supervisory Control

The supervisory control of prefix-closed languages is first introduced. The issue of blocking is not considered. The main concern it to limit the generated sequence in the plant to the desirable sequences and to prevent undesirable (unsafe) sequences.

We first introduce the languages which can be generated by the closed-loop systems under supervision of admissible supervisors. They are called controllable languages with respect to a certain plant and uncontrollable event set. For an language K, if all uncontrollable continuations $\overline{K}\Sigma_u$ is either outside $L(G)$ or remain in $\overline{K}$, then we say K is controllable with respect to $L(G)$ and Σ_u.

Definition 2.18: **Controllability** A language K is controllable with respect to another language M and uncontrollable event set Σ_u if $\overline{K}\Sigma_u \cap M \subseteq \overline{K}$. □

Obviously, Φ is always controllable. In addition, controllability is conserved under arbitrary unions. Therefore, if a language K itself is not controllable with respect to $L(G)$ and Σ_u, there exists a supremal controllable sublanguage of K denoted as $SupC(K,G)$.

Due to uncontrollable events, the closed-loop system behavior can not be any sublanguage of $L(G)$. The language of a system under supervision of an admissible supervisor shall be controllable.

Theorem 2.1: [5] Controllability Theorem Consider an automaton $G = (X, \Sigma, \delta, x_0)$ with uncontrollable event set $\Sigma_u \subseteq \Sigma$ and a nonempty language K with $\overline{K} \subseteq L(G)$. There exists an admissible supervisor S such that $L(S/G) = \overline{K}$ if and only if $\overline{K}$ is controllable with respect to $L(G)$ and Σ_u. □

An **optimal (minimally restrictive) supervisor** is a supervisor that only disables events if necessary.

BSCP: **Basic Supervisory Control Problem** Given DES G with event set Σ, uncontrollable event set $\Sigma_u \subseteq \Sigma$, and specification language $E = \overline{E} \subseteq L(G)$, synthesize an optimal supervisor S such that $L(S/G) \subseteq E$. □

Solution to BSCP: The above BSCP can be solved by simply calculating the supremal controllable sublanguage K of specification E with respect to $L(G)$ and Σ_u. An optimal supervisor S can then be realized by an automaton R with $L(R) = K = \overline{K}$. □

For the case of prefix-closed languages, closed-form expression exists for the calculation of the supremal controllable sublanguage. It is given as follows.

Computation of $SupC(K,G)$: Prefix-closed Case

If $K = \overline{K}$, then $SupC(K,G) = K - [(L(G) - K)/\Sigma_u^{*}]\Sigma^{*}$. □

2.2.2 Nonblocking Supervisory Control

In most cases, the nonblocking property of the closed-loop system becomes an important issue to be considered.

Definition 2.19: Relative-closed language ($L_m(G)$-closed) A language $K \subseteq L_m(G)$ is said relative-closed if $\overline{K} \cap L_m(G) = K$. □

Obviously, the empty language Φ is always relative-closed. In addition, the arbitrary union of relative-closed languages is still relative-closed. As a result, if a language K is not relative-closed, then there always exists a supremal relative-closed sublanguage of K denoted as $SupR(K,G)$. The supremal relative-closed sublanguage of K can be calculated with the formula $SupR(K,G) = K - (L_m(G) - K)\Sigma^*$.

The relative-closure property is conserved under the operation of supremal controllable sublanguage.

Lemma 2.5: [5] If a language K is relative-closed, its supremal controllable sublanguage $SupC(K,G)$ will remain relative-closed. □

Theorem 2.2: [5] Nonblocking Controllability Theorem (NCT) Consider DES $G = (X, \Sigma, \delta, x_0, X_m)$ where $\Sigma_u \subseteq \Sigma$ is the set of uncontrollable events and a nonempty language $K \subseteq L_m(G)$. There exists a nonblocking supervisor S such that $L_m(S/G) = K$ and $L(S/G) = \overline{K}$ if and only if the two following conditions hold:

1) Controllability: $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$.

2) Relative-closure: $\overline{K} \cap L_m(G) = K$. □

The nonblocking supervisory control problem is formally formed as follows.

BSCP-NB: Basic Supervisory Control Problem - Nonblocking Case Given DES G with event set Σ, uncontrollable event set $\Sigma_u \subseteq \Sigma$, and specification language $E \subseteq L_m(G)$, find an optimal (minimally restrictive) admissible nonblocking supervisor S such that $L_m(S/G) \subseteq E$. □

The solution to the robust nonblocking supervisory control problem is to find the supremal relative-closed and controllable sublanguage. Since the supremal controllable sublanguage of a relative-closed language remains relative-closed, the supremal controllable and relative-closed sublanguage can be computed by computing the supremal relative-closed sublanguage first and then obtaining the supremal controllable sublanguage.

Solution to BSCP-NB:

The solution to BSCP-NB is to first calculate $L = SupR(E,G)$ and then compute $K = SupC(L,G)$. An optimal admissible nonblocking supervisor S can be realized by an automaton R with $L(R) = L_m(R) = \overline{K}$. □

There is no closed-form expression for the calculation of the supremal controllable sublanguage in the general case. However, iterative procedures using the largest fixed point techniques are available for the computing the supremal controllable sublanguage $SupC(L,G)$ when the plant G is finite-state and the language L is regular.

2.2.3 State-based Supervisory Control

Let us consider a plant G. The set of all strings leading to a state x of a plant automaton G from the initial state is denoted by $[x]$. The post-languages of $L(G)$ of all strings in $[x]$ are the same denoted by $L(G)/[x]$. $L(G)/[x]$ can be generated by the accessible part of the automaton resulted from treating the state x of G as the initial state.

Suppose a trim automaton H marks the legal language E with $L_m(H) = E \subseteq L_m(G)$. If H is a subautomaton of G, the control policy for each string in $[x]$ will be the same [1]. Hence, the domain of the supervision map can be based on states rather than strings. A **state-based supervisor** is a function $S : X \to 2^{\Sigma}$.

2.3 Online Supervisory Control

Offline supervisory control synthesizes the overall supervisor in advance and stores the supervisor in memory for looking up at run time. If a system is too complicated, it will be impractical to synthesize or store offline supervisors.

Online supervisory control does not require modeling overall plant automaton and specification automaton. Instead, it predicts the system behavior and computes the control policy only for the current string at run time. As a result, there is no need for storing any supervisor. In addition, online supervision is suitable for control of time-varying discrete-event systems where system behavior may change when a system is in operation.

2.3.1 Limited Lookahead Policy

One of the online supervision methods is Limited Lookahead Policy (LLP). LLP expands the plant language as a tree generator at the current string and stops once a fixed length is reached for each branch. The automaton of a plant G and its two-step lookahead window at the empty string are shown in Figure 2.2. LLP takes a linguistic approach since each node of a tree corresponds to exactly one string.

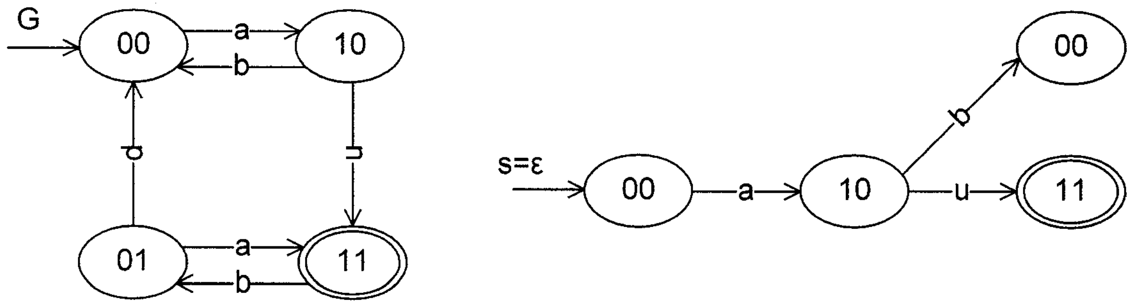

Figure 2. 2: Plant and two-step lookahead expansion

Let G denote a DES plant with event set Σ. We assume that all events are observable and G is nonblocking $\overline{L_m(G)} = L(G)$. Now we want to design a supervisor γ to restrict the plant G to certain specification $K \subseteq L_m(G)$ and keep the system under supervision nonblocking. In other words, the closed-loop system should satisfy $L_m(\gamma / G) \subseteq K$ and $\overline{L_m(\gamma / G)} = L(\gamma / G)$. We assume $K \neq \Phi$ and $L_m(G)-$closed $K = \overline{K} \cap L_m(G)$. The procedure of limited LLP is formulated below.

LLP supervisory control calculates the control action for the current string s in five stages. Firstly, the behavior of G in N steps beyond the current trace s is predicted as a N-step lookahead tree. Secondly, it determines which traces in the N-step tree are illegal according to the specification. The strings at the boundary of the tree which are not illegal are called pending strings due to uncertainty of the behavior beyond the lookahead window. At the third step, one of two different attitudes can be employed to deal with the ambiguity of the pending traces. The optimistic approach treats all pending traces legal and marked. On the contrary, the conservative approach views them as illegal. After that, the supremal controllable sublanguage of the resulting specification with respect to the lookahead window is calculated. Finally, the conservative (resp. optimistic) control decision $\gamma_{cons}^{N}(s)$ (resp. $\gamma_{optm}^{N}(s)$) at the current string s is made by allowing all uncontrollable active events and the first event of the sequences of the supremal controllable sublanguage. After an enabled event σ is executed, the new current string will be $s\sigma$ and the same procedure is repeated for the new current string.

There are three important issues in this framework. Firstly, what if the supermal controllable sublanguage in some lookahead window is Φ? Secondly, how do attitude and the length of lookahead windows affect the behavior of closed-loop systems? Finally, under what condition is the online control decision equivalent to the offline optimal (minimally restrictive) decision? The following definition deals with the first question.

Definition 2.20: Run time error (RTE) If the supremal controllable sublanguage is Φ at some current string $s \in L(G,\gamma)$, then we say there is a run time error at string s. □

If an RTE happens at some string s, then there will be no way to prevent the system from entering illegal states or becoming blocked. Hence, RTE must be avoided. If an RTE happens at the initial state, we call it a **starting error** (SE).

The second question is answered by the following properties. In the case of optimistic policy, $L(\gamma_{optm}^{N}/G)$ is bounded and monotone in that $\overline{SupC(K,G)} \subseteq L(\gamma_{optm}^{N}/G)$ and $L(\gamma_{optm}^{N}/G) \supseteq L(\gamma_{optm}^{N+1}/G)$. In the case that conservative policy is taken and $SupC(K,G) \neq \Phi$, $L(\gamma_{cons}^{N}/G)$ is also bounded and monotone in that $L(\gamma_{cons}^{N}/G) \subseteq \overline{SupC(K,G)}$ and $L(\gamma_{cons}^{N}/G) \subseteq L(\gamma_{cons}^{N+1}/G)$.

The validity defined below is concerned with the third question.

Definition 2.21: Validity An online supervisor γ is called valid if the behavior of the closed-loop system under its supervision is equivalent to the closed-loop behavior under offline optimal supervision. Namely, $L(\gamma/G) = \overline{SupC(K,G)}$. □

Although $L(\gamma_{optm}^{N}/G)$ and $L(\gamma_{cons}^{N}/G)$ are monotone and bounded, they do not converge to $\overline{SupC(K,G)}$ in general when N goes to infinity. However, a certain lookahead window size N may exist to make $L(\gamma/G) = \overline{SupC(K,G)}$ in some cases.

In order to obtain $L(\gamma/G) = \overline{SupC(K,G)}$, two conditions have to be satisfied:

1) Every uncontrollable subtrace that leads to the illegal region $(\overline{K})^{co}$ should be prevented before it is too late.

2) Such prevention should come as late as possible for validity.

The validity condition for both prefix-closed specifications and non-closed specifications are considered under both optimistic and conservative policies below.

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

The length of the longest uncontrollable subtrace of strings in a language K is denoted as $N_u(K)$ and defined below.

Definition 2.22:

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

The sufficient conditions for validity of LLP in the prefix-closed situation are given in the following theorems.

Theorem 2.3: [7] If $N \geq N_u(K) + 2$ or $N \geq N_u(L(G)) + 1$,

then $L(\gamma_{optm}^N / G) = SupC(K, G)$. □

Theorem 2.4: [7] If no SE happens in $L(\gamma_{cons}^N / G)$ and $N \geq N_u(K) + 2$, then $L(\gamma_{cons}^N / G) = SupC(K, G)$. □

Case B: Non-closed specification $K \subset \overline{K}$

Before giving the sufficient condition for validity, we define the following languages.

$$K_{mc} = \{s \in K : (\forall \sigma \in \Sigma_u) s\sigma \notin L(G)\} \qquad (2.1)$$

$$K_{fc} = ((L(G) - \overline{K}) / \Sigma_u) \cap \overline{K} \qquad (2.2)$$

K_{mc} is called the marked legal and controllable language. K_{fc} is the set of legal strings which is the catenation of a legal string with an uncontrollable event. K_{fc} can be regarded as the frontier of the legal language.

Definition 2.23:

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

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

N_{mcfc} is the length of the maximum subtrace which leads the empty string or the marked legal controllable strings to an illegal string without generating any marked legal controllable or illegal strings. N_{mcmc} is the length of the maximum subtrace which leads the empty string or the marked legal controllable strings to their neighbor next marked legal controllable string.

The following theorems give sufficient conditions for validity of an online supervision in optimistic and conservative attitudes respectively.

Theorem 2.5: **[7]** Assume $SupC(K,G) \neq \Phi$. If $N \geq N_{mcfc} + 1$ step lookahead is taken, then $L(\gamma_{optm}^{N} / G) = \overline{SupC(K,G)}$. $\square$

Theorem 2.6: **[7]** Assume no *SE* happens in $L(\gamma_{cons}^{N} / G)$ and $\overline{K} = \overline{K_{mc}}$. If $N \geq N_{mcmc} + 1$ step lookahead is taken, then $L(\gamma_{cons}^{N} / G) = \overline{SupC(K,G)}$. $\square$

2.3.2 Variable Lookahead Policy

The expansion of the N-step window in LLP may be unnecessary for each branch. For example, if a branch reaches an illegal string or a marked legal controllable string before N-step lookahead, further extension of the string is unnecessary. If a sufficiently large lookahead window size N will ensure validity of LLP, then all branches will terminate before N-step is reached. Another shortcoming of LLP is that the computation for the current string can not be reused when the system evolves.

Variable lookahead policy [10] was proposed to improve computational efficiency. The supervisory control problem is transferred to an optimization problem there. Strings are examined forward and in a depth-first approach. Values are assigned only to the nodes of the N-step expansion tree which are necessary to make control decision for the current string. In addition, the values assigned in the current expansion tree can be used when the system evolves one-step ahead. This further reduces computational complexity.

States are examined and values are assigned according to the following rules in VLP. If an illegal string is reached, ∞ is assigned to the node. If a marked legal controllable string is reached, value 0 is assigned. If a boundary string of the expansion tree is reached, ∞ is always assigned in the conservative attitude. Value 0 is assigned to the boundary strings in the optimistic approach except the illegal boundary strings. For other strings, all uncontrollable events are examined before all controllable events. If an uncontrollable σ_u event leads a string t to a string with ∞ value, ∞ is assigned to string t without further examining other active events. If a controllable event leads a string t to the next string with value 0, then string *t* is assigned value 0 without further examining other controllable events.

Validity may not be ensured in some cases no matter how many lookahead steps are taken in LLP and VLP. To solve this problem, variable lookahead policy with state information (VLP-S) [1] was proposed where control domain is states. The difference is that expansion window is a subgraph of the plant and values are assigned to states. The expansion at the empty string for the plant *G* in Figure 2.2 is shown in Figure 2.3. Expansion always terminates since the number of states is finite for finite-state plants and regular specifications. No attitude is required and validity is always guaranteed.

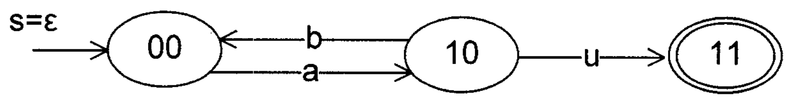

Figure 2. 3: VLP-S expansion

In this thesis, we take variable-lookahead policy with both linguistic (VLP) and state-based (VLP-S) approaches to synthesize valid online supervisors. VLP works only if expansion always terminates. To solve this problem, we formulate state-based supervisory control problem and propose an algorithm VLP-S to solve it. We make control decision by computing the supremal element using fixed point technique which is more transparent and easier to understand in comparison with the approach of transferring supervisory control problems to optimization problems.

2.4 Robust Supervisory Control

Conventional supervisory control assumes that a plant model is certain and can be represented by a known automaton. However, this assumption does not always hold and thus uncertainty has to be modeled. For example, a system may have different configurations for different tasks and thus we do not have a certain model for the system.

Robust supervisory control is one approach to handle model uncertainty. For example, we may design a unique supervisor which can meet requirements no matter how the system is configured.

In this thesis, we develop on-line solutions for a robust control problem. In this section, we review some of the results on robust control in the literature. There are different ways to model uncertainty and thus different formulations of robust supervisory control. Here, we only review the robust supervisory control with the assumption that the system model could be in a finite set of possible models though the exact model is uncertain.

2.4.1 Problem Formulation

Assume that a system model belongs to a set of possible known models G_i with $\Sigma_i = \Sigma_{c,i} \dot{\cup} \Sigma_{u,i}$ for all $i \in I = \{1,2,...,n\}$, and the specification for each possible mode G_i is $E_i \subseteq L_m(G_i)$. We also assume that an event is controllable (resp. uncontrollable) in all other plant models if it is controllable (resp. uncontrollable) in one plant model.

Now we want to design a robust supervisor $V : \bigcup_{i \in I} L(G_i) \rightarrow 2^{\Sigma}$ where $\Sigma = \bigcup_{i \in I} \Sigma_i$ such that the closed-loop system will meet the specifications and be nonblocking whatever the real plant model could be. When the robust supervisor V is applied to a certain plant model G_i, its function will be $V_i : L(G_i) \to 2^{\Sigma_i}$ with $V_i(s) = V(s) \cap \Sigma_i$. In other words, the closed-loop plant V / G_i denotes V_i / G_i for all $i \in I$.

Admissibility and nonblocking properties of a robust supervisor are required to be defined before we formally define the robust nonblocking supervisory control problem.

Definition 2.24: A robust supervisor $V : \bigcup_{i \in I} L(G_i) \to 2^{\Sigma}$ is said to be admissible if and only if V is admissible for all G_i where $i \in I$. $\square$

Definition 2.25: A robust supervisor $V : \bigcup_{i \in I} L(G_i) \to 2^{\Sigma}$ is said to be nonblocking if and only if $\overline{L_m(V / G_i)} = L(V / G_i)$ for all $i \in I$. $\square$

Definition 2.26: Let $\mathcal{V}^o$ be the collection of all admissible and nonblocking robust supervisors. A supervisor $\hat{V} \in \mathcal{V}^o$ is said to be the maximally permissive supervisor if $L(\hat{V} / G_i) \supseteq L(\tilde{V} / G_i)$ for any $\tilde{V} \in \mathcal{V}^o$. $\square$

The robust nonblocking supervisory control problem is described as below.

Robust Nonblocking Supervisory Control Problem (RNSCP):

Given a set of plants G_i with $i \in I = \{1,2,\ldots,n\}$, the marked language $L_m(G_i)$ is required to be restricted into $E_i \subseteq L_m(G_i)$ for all $i \in I$. Synthesize an admissible nonblocking robust supervisor, which solves the problem. If existing, synthesize a maximally permissive admissible nonblocking robust supervisor. $\square$

The next subsection introduces offline indirect solution in the literature to the above RNSCP. We will develop a robust direct solution later in this thesis.

2.4.2 An Offline Solution

The solution to RNSCP when G_i's are nonblocking is given in [3]. The solution in the general case is provided in [27]. We refer to this solution as the indirect solution. Before providing this solution, we need to review the concept of *G*-nonblocking languages.

Definition 2.27: *G*-nonblocking Let $G = (X, \Sigma, \delta, x_0, X_m)$. A language $K \subseteq \Sigma^*$ is called G-nonblocking if $\overline{K \cap L_m(G)} = \overline{K} \cap L(G)$. □

Let $N_b(K, G)$ denote the set of *G*-nonblocking sublanguages of *K*. The empty language Φ is G-nonblocking. In addition, G-nonblocking languages remain *G*-nonblocking under arbitrary number of union operations. As a result, there exists a supremal G-nonblocking sublanguage denoted as $SupN_b(K, G)$. Obviously, if K is G-nonblocking, then $SupN_b(K, G) = K$. $SupN_b(K, G)$ can be calculated using the following formula: $SupN_b(K, G) = K - (\overline{K} \cap L(G) - \overline{K \cap L_m(G)})\Sigma^*$.

Theorem 2.7: [27] Consider DES $G_i = (X_i, \Sigma_i, \delta_i, x_{0i}, X_{mi})$ with $i \in I = \{1,2,...,n\}$. Let $\Sigma_u \subseteq \Sigma$ be the set of uncontrollable events. Let G be an automaton with $L(G) = \bigcup_{i \in I} L(G_i)$ and $L_m(G) = \bigcup_{i \in I} L_m(G_i)$. Consider a nonempty language $K \subseteq L_m(G)$. There exists a supervisor S such that $L_m(S/G_i) = K \cap L_m(G_i)$ and $\overline{L_m(S/G_i)} = L(S/G_i)$ for all $i \in I$ if and only if the three following conditions hold:

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

2) $L_m(G)$-closure: $\overline{K} \cap L_m(G) = K$

3) G_i-nonblocking: $\overline{K \cap L_m(G_i)} = \overline{K} \cap L(G_i)$ for all $i \in I$. □

Definition 2.28: Let $G_i = (X_i, \Sigma_i, \delta_i, x_{0i}, X_{mi})$ where $\Sigma_u \subseteq \Sigma$ is the set of uncontrollable events. Synthesize G with $L(G) = \bigcup_{i=1}^{n} L(G_i)$ and $L_m(G) = \bigcup_{i=1}^{n} L_m(G_i)$. The set of sublanguages of $E \subseteq L_m(G)$ that is controllable with respect to G, $L_m(G)$-closed, and G_i-nonblocking for all $i \in I$ is denoted as $RN_bC(E,G)$. □

The empty language Φ is in $RN_bC(E,G)$. In addition, languages in $RN_bC(E,G)$ remain in $RNC(E,G)$ under arbitrary number of union operations. As a result, $RN_bC(E,G)$ has a supremal element denoted as $SupRN_bC(E,G)$. $SupRNC(E,G)$ provides the optimal solution to RNSCP.

Offline Solution to RNSCP:

The solution to the RNSCP is to synthesize an overall plant G with $L(G) = \bigcup_{i=1}^{n} L(G_i)$ and $L_m(G) = \bigcup_{i=1}^{n} L_m(G_i)$ and an overall specification $E = [\bigcap_{i \in I} (E_i \cup (\Sigma^* - L_m(G_i))] \cap L_m(G)$ first. Then, the supremal controllable, $L_m(G)$-closed, and G_i-nonblocking sublanguage of E is calculated. Finally, the supervisor $V : L(G) \to 2^{\Sigma}$ with $L(V/G) = \overline{SupRN_bC(E,G)}$ will solve the robust problem. □

There is no closed-form expression to compute $SupRN_bC(E,G)$. Nevertheless, it can be calculated if the following algorithm terminates.

Algorithm 2.1: SupRN$_b$C(E,G)

1) Initialization: let $E^0 = E$.

2) Update $E^1 = SupR(E^0, G)$.

3) Update $E^2 = SupC(E^1, G)$.

4) Update $E^3 = SupN_b(E^2, G_1)$, $E^4 = SupN_b(E^3, G_2)$,..., $E^{n+2} = SupN_b(E^{n+1}, G_n)$.

5) If $E^{n+2} \neq E^0$, then let $E^0 = E^{n+2}$ and go to step 2.

6) $SupRN_bC(E,G) = E^0$. □

2.4.3 Illustrative Example

We introduce a running example and solve it using the above indirect approach. The same problem will also be solved using direct approach offline and online developed in Chapter 3 and 4 respectively.

Example 2.1: A system model could be either G_1 or G_2 shown below. Assume that $\Sigma = \{\alpha, \beta, \gamma, \lambda, \mu, u, v, w\}$ with $\Sigma_c = \{\alpha, \beta, \gamma, \lambda, \mu\}$ and $\Sigma_u = \{u, v, w\}$. The specification languages for G_1 and G_2 are $E_1 = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$ and $E_2 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu\}$ respectively. Synthesize an optimal admissible nonblocking robust supervisor which solves the problem.

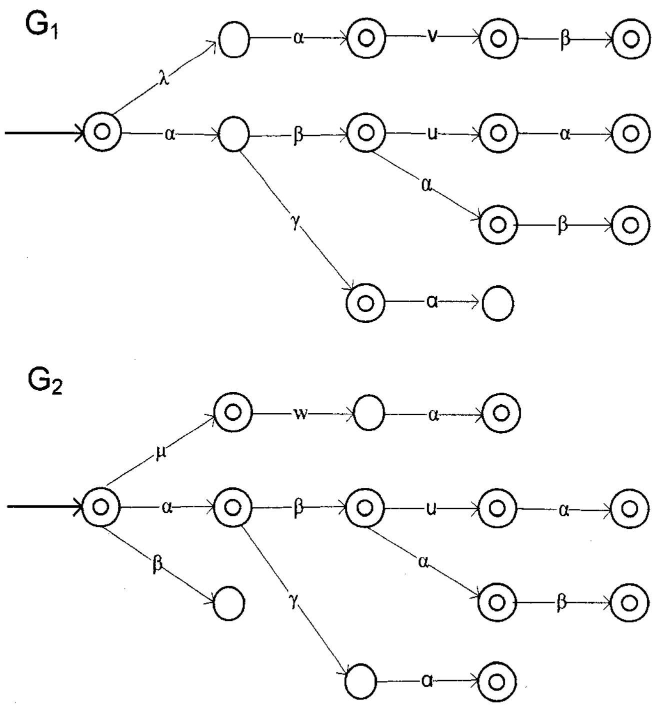

Figure 2. 4: Automata for Example 2.1

The language generated and marked by the plant models are:

$$L(G_1) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \lambda, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$$

$$L(G_2) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \mu, \mu w, \mu w\alpha, \beta\}$$

$$L_m(G_1) = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$$

$$L_m(G_2) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu, \mu w\alpha\}.$$

We then solve the problem using the existing offline indirect approach. First, we synthesize the overall plant G with $L(G) = L(G_1) \cup L(G_2)$ and $L_m(G) = L_m(G_1) \cup L_m(G_2)$. The automaton of G is shown in Figure 2.5.

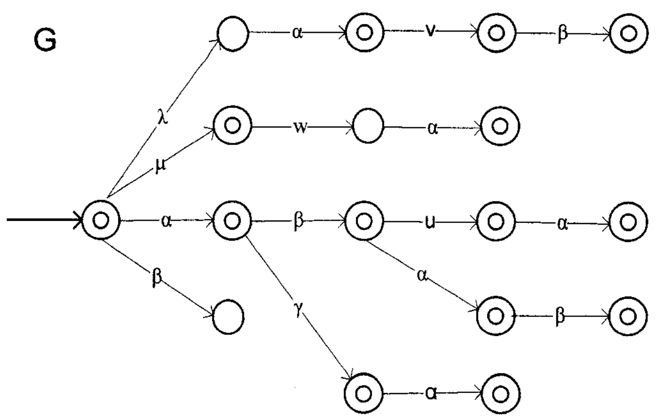

Figure 2. 5: The overall plant for Example 2.1

The generated and marked languages of G are:

$L(G) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \lambda, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta, \mu, \mu w, \mu w\alpha, \beta\}$

$L_m(G) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta, \mu, \mu w\alpha\}$.

The overall specification will be:

$E = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \alpha\gamma\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta, \mu\}$.

We then use Algorithm 2.1 to find $SupRN_bC(E,G)$.

Initialization: $E^0 = E$.

Iteration 1: E^0 is $L_m(G)$-closed and thus $E^1 = E^0$. E^1 is not controllable with respect to G. Thus, we calculate the supremal controllable sublanguage of E^1 to get E^2. E^2 is the language by removing string μ from E^1. E^2 is not G_1-nonblocking for all $i \in I$. For example, string $\alpha\gamma\alpha$ belongs to $\overline{E^2} \cap L(G_1)$ but it is not in $\overline{E^2 \cap L_m(G_1)}$. The supremal

G_1-nonblocking sublanguage E^3 of E^2 is $E^3 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$. E^3 is not G_2-nonblocking. For example, string $\alpha\gamma$ belongs to $\overline{E^3} \cap L(G_2)$ but it is not in $\overline{E^3 \cap L_m(G_2)}$. The supremal G_2-nonblocking sublanguage E^4 of E^3 will be $E^4 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$. Since $E^4 \neq E^0$, we let $E^0 = E^4$ and take the next iteration.

Iteration 2: E^0 does not change in this iteration and thus we stop with the optimal solution $E^0 = \sup RN_bC(E,G) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$. We can verify that E^0 will be controllable, $L_m(G)$-closed, and G_i-nonblocking. The supervisor can be realized by automaton R in Figure 2.6 with $L(R) = L_m(R) = \overline{E^0}$. □

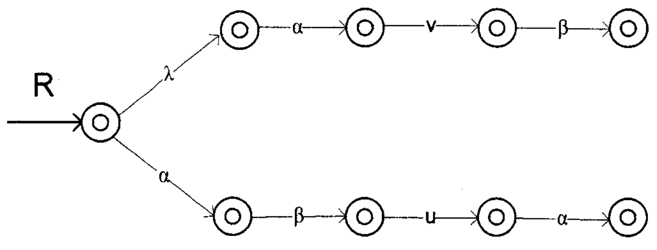

Figure 2. 6: Realization of Supervisor: Example 2.1

2.4.4 Example: Fault Recovery

One of the application of the theory of robust control is in fault recovery problems. In fact, fault recovery problems can be solved as a special case of robust problems. Suppose a DES G has p failure modes $F_1, \ldots, F_p$. Assuming single-failure scenario, G can be in $p+1$ modes: N, $F_1, \ldots, F_p$.

Assume the states of the overall plant model with failures can be partitioned into normal states X_N and different failure states X_{F_i} shown in Figure 2.7. The subautomaton containing only the normal states X_N is the normal model of the plant G_N. The subautomaton with the normal states X_N and one type of failure states, say X_{F_i}, is a normal-failure model referred to as G_{NF_i}. We want that each model under supervision meets its corresponding specification and be nonblocking.

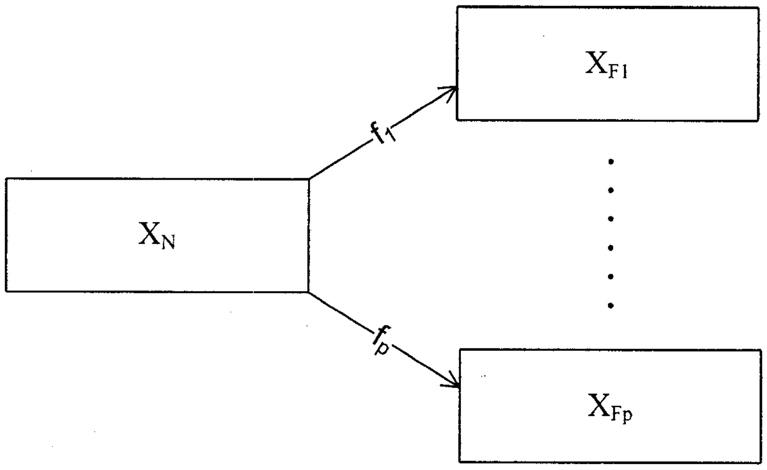

Figure 2. 7: Plant with *p* permanent failure modes

We assume the following: all events are observable; failures are permanent; only one failure happens at a time, and all modes agree on if an event is controllable or uncontrollable. Fault recovery problem can be described as follows.

Fault Recovery Problem (FRP):

Given a DES with a normal model G_N and normal-failure models G_{NF_i} for $i \in I = \{1,2,......p\}$ and their corresponding specifications $E_N \subseteq L_m(G_N)$ and $E_{NF_i} \subseteq L_m(G_{NF_i})$ for $i \in I$, synthesize a supervisor $S : \Sigma^* \to 2^\Sigma$ such that:

1) $L_m(S/G_N) \subseteq E_N$ and $\overline{L_m(S/G_N)} = L(S/G_N)$

2) $L_m(S/G_{NF_i}) \subseteq E_{NF_i}$ and $\overline{L_m(S/G_{NF_i})} = L(S/G_{NF_i})$ for all $i \in I$. □

Fault recovery problem FRP can be treated as a special case of robust supervisory control problem. Since a FRP can be treated as a RNSCP, solving the corresponding RNSCP will automatically solve the FRP.

Chapter 3:

Robust Nonblocking Supervisory Control with Direct Approach

The indirect approach to the solution of the robust control problem characterizes the solutions in terms of sublanguages of the legal behavior of the union of plant models. This chapter develops a new direct approach to RNSCP which describes the solutions in terms of sublanguages of the legal behavior of each plant. This characterization of solutions relies on the concept of consistency of languages introduced here. The resulting direct approach, in our opinion, is a more transparent approach especially for the purpose of the development of online solutions.

The organization of this chapter is as follows. Offline direct approach to RNSCP is first developed in Section 3.1. Algorithms are then proposed in Section 3.2 to calculate the optimal solution to the robust nonblocking supervisory control problem (RNSCP). The running mathematical problem in Example 2.1 is solved there using the offline direct approach. Section 3.3 discusses an application of RNSCP, namely, supervisory control with multiple sets of marked states.

3.1 Robust Nonblocking Supervisory Control: Direct Solution

We investigate robust nonblocking supervisory control problem assuming that the exact plant model is unknown but only could be one of a set of models. We review RNSCP before we derive a direct approach to solve it.

RNSCP—Robust Nonblocking Supervisory Control Problem:

Given a set of plants G_i with $i \in I$ (with $I = \{1,...,n\}$), the marked language $L_m(G_i)$ of plant G_i is required to be restricted into its specification $E_i \subseteq L_m(G_i)$ for all $i \in I$. Synthesize an admissible nonblocking supervisor, which solves the problem. □

3.1.1 Direct View of RNSCP

Suppose a supervisor V solves RNSCP and let us denote the set of marked closed-loop languages as a language n-tuple $(L_m(V/G_1),...,L_m(V/G_n))$. What shall be the necessary conditions that the language n-tuple $(L_m(V/G_1),...,L_m(V/G_n))$ has to satisfy? This question takes a direct view at RNSCP.

Like traditional supervisory control, controllability and relative-closure are necessary. The direct approach considers the language n-tuple $(L_m(V/G_1),...,L_m(V/G_n))$ and therefore it is useful to define controllability and relative-closure properties of language n-tuple $(M_1,...,M_n)$.

Definition 3.1: (Controllability) Language n-tuple $(M_1,...,M_n)$ is said to be controllable with respect to closed language n-tuple $(L(G_1),...,L(G_n))$ iff $\overline{M_i}\Sigma_{ui}\ \mathrm{I}\ L(G_i) \subseteq \overline{M_i}$ for all $i \in I$. □

Definition 3.2: (Relative-closure/L$_m$(G$_i$)-closure) Language n-tuple $(M_1,...,M_n)$ is said to be relative-closed with respect to language n-tuple $(L_m(G_1),...,L_m(G_n))$ iff $\overline{M_i}\ \mathrm{I}\ L_m(G_i) = M_i$ for all $i \in I$. □

If a string s can be generated by more than one model, a robust supervisor can either enable it in all those plants or disable it in all those plants. Namely, $L(V/G_i)\cap[L(G_j)-L(V/G_j)]=\Phi$ for all $i,j\in I$ has to be satisfied for a supervisor V to solve RNSCP. This means that if a string is enabled in V/G_i, then it is either enabled in V/G_j or cannot be generated by G_j at all. Since V has to be a nonblocking supervisor, it is necessary that $\overline{L_m(V/G_i)}=L(V/G_i)$ for all $i\in I$. Therefore, $\overline{L_m(V/G_i)}\cap[L(G_j)-\overline{L_m(V/G_j)}]=\Phi$ is required for all $i,j\in I$. This motivates the definition of consistency.

Definition 3.3: (Consistency) A Language n-tuple $(M_1,...,M_n)$ is said to be consistent with respect to a closed language n-tuple $(L(G_1),...,L(G_n))$ if and only if $\overline{M_i}\cap(L(G_j)-\overline{M_j})=\Phi$ for all $i,j\in I$. □

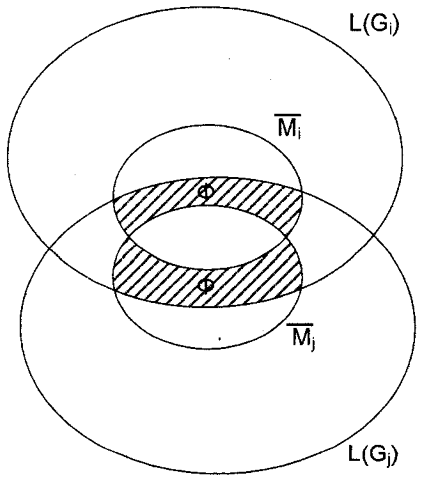

Figure 3. 1: Relation of consistent languages

The relation of any two languages M_i and M_j belonging to a consistent language n-tuple $(M_1,...,M_n)$ can be represented by the Venn diagram in Figure 3.1. There is no string inside the shaded areas which corresponds to those strings enabled in one closed-loop system and disabled in the other closed-loop system.

Consistency is a necessary condition for the existence of a robust supervisor from the above analysis. Here, we give a simple example to show that no robust supervisor could be synthesized if consistency of specifications is not satisfied. We also synthesize a supervisor for a consistent sublanguage n-tuple of the specifications.

Example 3.1: Two possible plant models G_1 and G_2 are shown in Figure 3.2. For simplicity, all events are assumed controllable. We synthesize a robust supervisor which restricts $L(G_1) = \{\varepsilon, a, b\}$ and $L(G_2) = \{\varepsilon, a, c\}$ inside specifications $E_1 = \{\varepsilon, a, b\}$ and $E_2 = \{\varepsilon, c\}$ respectively.

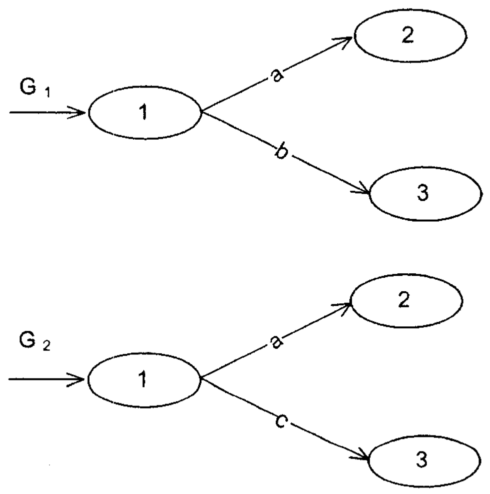

Figure 3. 2: Plant automata for Example 3.1

(E_1, E_2) is not consistent with respect to the closed behavior pair $(L(G_1), L(G_2))$ since $\overline{E_1} \cap (L(G_2) - \overline{E_2}) = \{a\} \neq \Phi$. It is impossible for a supervisor $V : L(G_1) \cup L(G_2) \to 2^{\Sigma}$ to satisfy $L(V/G_1) = \overline{E_1}$ and $L(V/G_2) = \overline{E_2}$ since event a is required to be enabled in G_1 but disabled in G_2 at the initial state. From the formulation of the problem, a robust supervisor is required to work for both plant models.

On the other hand, let $M_1 = \{\varepsilon, b\} \subseteq E_1$ and $M_2 = E_2 = \{\varepsilon, c\}$. Observe that (M_1, M_2) is consistent with respect to closed behavior pair $(L(G_1), L(G_2))$. A supervisor $V : L(G_1) \cup L(G_2) \to 2^{\Sigma}$ with $V(\varepsilon) = \{b, c\}$, $V(b) = \Phi$, and $V(c) = \Phi$ will guarantee that $L(V/G_1) = \overline{M_1} = \{\varepsilon, b\}$ and $L(V/G_2) = \overline{M_2} = \{\varepsilon, c\}$. □

3.1.2 Direct Solution to RNSCP

Theorem 3.1 provides necessary and sufficient conditions for the existence of an admissible nonblocking robust supervisor.

Theorem 3.1: DRNCT (Direct Robust Nonblocking Supervisory Control Theorem)

Consider a set of plants $G_i = (X_i, \Sigma_i, \delta_i, X_{0i})$ where $i \in I = \{1, 2, \ldots, n\}$. $\Sigma = \bigcup_{i \in I} \Sigma_i$ can be partitioned as two disjoint sets $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$ where Σ_c and Σ_u are the set of controllable and uncontrollable events respectively. Given a language n-tuple $(M_1, \ldots, M_n)$ where $M_i \neq \Phi$ and $M_i \subseteq L_m(G_i)$ for $i \in I$, there exists an admissible nonblocking robust supervisor $\hat{V}$ such that $L_m(\hat{V}/G_i) = M_i$ for all $i \in I$ if and only if:

1) Controllability: $(M_1,\ldots,M_n)$ is controllable with respect to $(L(G_1),\ldots,L(G_n))$.

2) Relative-Closure: $(M_1,\ldots,M_n)$ is relative-closed with respect to $(L_m(G_1),\ldots,L_m(G_n))$.

3) Consistency: $(M_1,\ldots,M_n)$ is consistent with respect to $(L(G_1),\ldots,L(G_n))$. □

We need the following lemmas to prove Theorem 3.1.

Lemma 3.1: Assume that $M_i \neq \Phi$ and $M_i \subseteq L_m(G_i)$ for all $i \in I$. Let language n-tuple $(M_1,\ldots,M_n)$ be controllable and consistent with respect to language n-tuple $(L(G_1),\ldots,L(G_n))$. If we define a set of admissible supervisors $V_i : \underset{i\in I}{\mathrm{Y}} L(G_i) \to 2^{\Sigma}$ with

$$V_i(s) = \begin{cases} \{\sigma \in \Sigma \mid s\sigma \in \overline{M_i}\} & \text{if } s \in L(G_i) \\ \Phi & \text{otherwise} \end{cases}$$

for all $i \in I$ and synthesize a *modular* supervisor $\widetilde{V} : \underset{i\in I}{\mathrm{Y}} L(G_i) \to 2^{\Sigma}$ with $\widetilde{V}(s) = \underset{i\in I}{\mathrm{Y}} V_i(s)$, then $L(\widetilde{V}/G_i) = \overline{M_i} = L(V_i/G_i)$ for all $i \in I$.

Proof: From controllability, we have $L(V_i/G_i) = \overline{M_i}$ for all $i \in I$.

To show $L(\widetilde{V}/G_i) = \overline{M_i}$ for all $i \in I$, it suffices to show $s \in L(\widetilde{V}/G_i) \Leftrightarrow s \in \overline{M_i}$ for all stings in the two languages. We prove this by induction on the length of all the strings.

Induction base: The base case is when the length of strings is zero. ε is the only string with zero length. We have $\varepsilon \in L(\tilde{V}/G_i)$ by definition. We also have $\varepsilon \in \overline{M_i}$ by assumption $\overline{M_i} \neq \Phi$. Therefore, the induction base holds.

Induction hypothesis: we assume that $s \in L(\tilde{V}/G_i)$ iff $s \in \overline{M_i}$ for all s with length less or equal to n ($|s| \leq n$).

Induction step: for all $\sigma \in \Sigma$.

$(\Leftarrow)$: Let $s\sigma \in \overline{M_i}$. It is obvious that $s\sigma \in L(V_i/G_i)$ and $s\sigma \in L(G_i)$ since $L(V_i/G_i) = \overline{M_i}$ and $\overline{M_i} \subseteq \overline{L_m(G_i)} \subseteq L(G_i)$. Therefore, we have $\sigma \in V_i(s)$, which implies $\sigma \in \tilde{V}(s)$ and thus $s\sigma \in L(\tilde{V}/G_i)$.

$(\Rightarrow)$: Let $s\sigma \in L(\tilde{V}/G_i)$. It is obvious that $s\sigma \in L(G_i)$ since $L(\tilde{V}/G_i) \subseteq L(G_i)$.

Suppose $\sigma \notin V_i(s)$, then we have:

$s\sigma \notin L(V_i/G_i) = \overline{M_i} \Rightarrow s\sigma \in L(G_i) - \overline{M_i}$

$\Rightarrow \forall j \in I : s\sigma \notin \overline{M_j}$ (By Consistency)

$\Rightarrow \forall j \in I : s\sigma \notin L(V_j/G_j)$ (By $L(V_j/G_j) = \overline{M_j}$)

$\Rightarrow \forall j \in I : \sigma \notin V_j(s)$ (By definition of V_j)

$\Rightarrow \sigma \notin \bigcup_{j\in I} V_j(s)$

$\Rightarrow \sigma \notin \tilde{V}(s)$

$\Rightarrow s\sigma \notin L(\tilde{V}/G_i)$.

This is contradictory with $s\sigma \in L(\widetilde{V}/G_i)$. Thus, we have $\sigma \in V_i(s)$ and $s\sigma \in L(V_i/G_i) = \overline{M_i}$. As a result, $s\sigma \in L(\widetilde{V}/G_i) \Leftrightarrow s\sigma \in \overline{M_i}$.

We conclude that $L(\widetilde{V}/G_i) = \overline{M_i} = L(V_i/G_i)$ for all $i \in I$. This completes the proof. □

Remark 3.1: Lemma 3.1 indicates that because of consistency of $(M_1,...,M_n)$, $\widetilde{V}$ does not allow any extra string in $L(\widetilde{V}/G_i)$ comparing with $L(V_i/G_i) = \overline{M_i}$. Without consistency, $L(\widetilde{V}/G_i) = \overline{M_i}$ may not necessarily be true. This clearly shows the importance of consistency. □

The following two examples show that the closed-loop behavior $L(\widetilde{V}/G_i)$ is exactly the same as the closed-loop behavior $L(V_i/G_i)$ for all $i \in \{1,2\}$ when consistency exists.

Example 3.2: Consider the consistent pair (M_1,M_2) in Example 3.1 where $M_1 = \{\varepsilon, b\}$ and $M_2 = \{\varepsilon, c\}$. We synthesize a modular supervisor $\widetilde{V}$ as shown in Lemma 3.1 such that $L(\widetilde{V}/G_i) = \overline{M_i} = L(V_i/G_i)$ for all $i \in \{1,2\}$.

The modular supervisor $\widetilde{V}(s) = V_1(s) \cup V_2(s)$ can be derived from the two supervisors

$V_1 : L(G_1) \cup L(G_2) \rightarrow 2^{\Sigma}$ with $V_1(s) = \begin{cases} \{\sigma \in \Sigma \mid s\sigma \in \overline{M_1}\} & \text{if } s \in L(G_1) \\ \Phi & \text{otherwise} \end{cases}$ and

$V_2 : L(G_1) \cup L(G_2) \rightarrow 2^{\Sigma}$ with $V_2(s) = \begin{cases} \{\sigma \in \Sigma \mid s\sigma \in \overline{M_2}\} & \text{if } s \in L(G_2) \\ \Phi & \text{otherwise} \end{cases}$.

For example, $V_1(\varepsilon)=\{b\}$ and $V_2(\varepsilon)=\{c\}$ at string $s=\varepsilon$, and thus $\widetilde{V}(\varepsilon)=V_1(\varepsilon)\cup V_2(\varepsilon)=\{b,c\}$. We can see that $L(\widetilde{V}/G_1)=\overline{M_1}=\{\varepsilon,b\}$ and $L(\widetilde{V}/G_2)=\overline{M_2}=\{\varepsilon,c\}$.

Obviously, $L(V_1/G_1)=\overline{M_1}=\{\varepsilon,b\}$ and $L(V_2/G_2)=\overline{M_2}=\{\varepsilon,c\}$. This shows that $L(\widetilde{V}/G_i)=\overline{M_i}=L(V_i/G_i)$ for all $i\in\{1,2\}$ due to consistency. □

Example 3.3: Let us consider the inconsistent languages $E_1=\{\varepsilon,a,b\}$ and $E_2=\{\varepsilon,c\}$ in Example 3.1. We synthesize a modular supervisor $\widetilde{V}$ as shown in Lemma 3.1 and show that $L(\widetilde{V}/G_i)=\overline{E_i}=L(V_i/G_i)$ does not hold for all $i\in\{1,2\}$.

The modular supervisor $\widetilde{V}(s)=V_1(s)\cup V_2(s)$ is synthesized the same way as Example 3.2. For example, $V_1(\varepsilon)=\{a,b\}$ and $V_2(\varepsilon)=\{c\}$ at string $s=\varepsilon$, and thus $\widetilde{V}(\varepsilon)=V_1(\varepsilon)\cup V_2(\varepsilon)=\{a,b,c\}$. We can see that $L(\widetilde{V}/G_2)=\{\varepsilon,a,c\}$.

Obviously, $L(V_2/G_2)=\overline{E_2}=\{\varepsilon,c\}$. This shows that $L(\widetilde{V}/G_2)\neq\overline{E_2}=L(V_2/G_2)$ due to inconsistency. □

Lemma 3.2: Consider the language n-tuple $(M_1,\ldots,M_n)$ and the modular supervisor $\widetilde{V}$ in Lemma 3.1. Assume that $(M_1,\ldots,M_n)$ is relative-closed with respect to language n-tuple $(L_m(G_1),\ldots,L_m(G_n))$. Then $L_m(\widetilde{V}/G_i)=M_i=L_m(V_i/G_i)$ for all $i\in I$.

Proof: M_i is relative-closed with respect to $L_m(G_i)$ and thus by Theorem 2.2 $L_m(V_i/G_i)=M_i$.

$$
\begin{aligned}
L_m(\tilde{V}/G_i) &= L(\tilde{V}/G_i) \cap L_m(G_i) \\
&= \overline{M_i} \cap L_m(G_i) && \text{(by Lemma 3.1)} \\
&= M_i && \text{(} M_i \text{ is relative-closed with respect to } L_m(G_i) \text{)}
\end{aligned}
$$

We conclude that $L_m(\tilde{V}/G_i) = M_i = L_m(V_i/G_i)$. This completes the proof. □

In Lemma 3.3, we show that consistency of $(M_1,\dots,M_n)$ is a necessary condition for existence of a nonblocking supervisor V such that $L_m(V/G_i) = M_i \neq \Phi$ for all $i \in I$. In other words, it is impossible to synthesize a nonblocking supervisor V to guarantee $L_m(V/G_i) = M_i$ for all $i \in I$ if consistency of $(M_1,\dots,M_n)$ is not satisfied.

Lemma 3.3: Let V be a robust nonblocking supervisor with $L_m(V/G_i) = M_i \neq \Phi$ ($i \in I$). Then $(M_1,\dots,M_n)$ must be consistent with respect to language n-tuple $(L(G_1),\dots,L(G_n))$.

Proof: (by Contradiction)

Suppose $\overline{M_i} \cap (L(G_j) - \overline{M_j}) \neq \Phi$ for some $i, j \in I$.

$$
\begin{aligned}
\overline{M_i} \cap (L(G_j) - \overline{M_j}) \neq \Phi &\Rightarrow \exists s \in \overline{M_i} \cap (L(G_j) - \overline{M_j}) \\
&\Rightarrow s \in \overline{M_i} \wedge s \in L(G_j) \wedge s \notin \overline{M_j} \\
&\Rightarrow s \in L(V/G_i) \wedge s \in L(G_j) \wedge s \notin L(V/G_j).
\end{aligned}
$$

It shows that s is enabled in G_i by V but disabled in G_j by V. But the robust supervisor V cannot distinguish G_i from G_j and thus we reach contradiction. Therefore,

$\overline{M_i} \cap (L(G_j) - \overline{M_j}) = \Phi$. $\overline{M_i} \cap (L(G_j) - \overline{M_j}) = \Phi$ means that $(M_1,...,M_n)$ is consistent with respect to language n-tuple $(L(G_1),...,L(G_n))$. □

We are ready to provide proof to Theorem 3.1 at this point.

Proof of the Theorem 3.1 (DRNCT):

(If): Given $(M_1,...,M_n)$ which meets the three conditions in Theorem 3.1, we need to synthesize an admissible nonblocking robust supervisor $\tilde{V}$ with $L_m(\tilde{V}/G_i) = M_i$ for all $i \in I$.

Define a set of admissible supervisors $V_i : \bigcup_{i \in I} L(G_i) \to 2^{\Sigma_i}$ with

$$V_i(s) = \begin{cases} \{\sigma \in \Sigma_i \mid s\sigma \in \overline{M_i}\} & \text{if } s \in L(G_i) \\ \Phi & \text{otherwise} \end{cases}$$

for all $i \in I$ and synthesize the modular supervisor $\tilde{V} : \bigcup_{i \in I} L(G_i) \to 2^{\Sigma}$ with $\tilde{V}(s) = \bigcup_{i \in I} V_i(s)$. We show that $\tilde{V}$ is an admissible nonblocking supervisor which meets the specification.

1) It follows from Lemma 3.1 that $L(\tilde{V}/G_i) = \overline{M_i}$ for all $i \in I$. Note that $\tilde{V}$ is admissible since $\overline{M_i}$ is controllable with respect to $L(G_i)$ for all $i \in I$.

2) It follows from Lemma 3.2 that $L_m(\tilde{V}/G_i) = M_i \subseteq E_i$ for all $i \in I$. It shows that $\tilde{V}$ meets safety specification.

3) $\overline{L_m(\widetilde{V}/G_i)} = L(\widetilde{V}/G_i) = \overline{M_i}$ for all $i \in I$ from Lemma 3.1 and 3.2. It shows that $\widetilde{V}$ is a nonblocking supervisor.

(Only If): Assume that there exists an admissible nonblocking supervisor V such that for every $i \in I : L_m(V/G_i) = M_i$. We need to show the language n-tuple $(M_1,...,M_n)$ meets the three conditions in Theorem 3.1.

1) (Controllability): M_i is controllable with respect to $L(G_i)$ (for all $i \in I$) since V is assumed to be admissible. Thus, $(M_1,...,M_n)$ is controllable with respect to $(L(G_1),...,L(G_n))$ by definition.

2) (Relative-closure): By assumption, V is a nonblocking supervisor and therefore, by Theorem 2.2 M_i is relative-closed with respect to $L_m(G_i)$ (for all $i \in I$). Thus, $(M_1,...,M_n)$ is relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$ by definition.

3) (Consistency): Follows from Lemma 3.3.

This completes the proof. □

If the specification language n-tuple $(E_1,...,E_n)$ in RNSCP does not meet the three conditions in DRNCT, there will not exist an admissible nonblocking supervisor V that ensures $L_m(V/G_i) = E_i$ for all $i \in I$. However, there may exist an admissible nonblocking supervisor V such that $L_m(V/G_i) = M_i \subset E_i$ for all $i \in I$ for sublanguages $M_i \subset E_i$ $(i \in I)$.

Since the direct approach is in terms of language n-tuple, it will be useful to consider the inclusion relation and the intersection and union operations for language n-tuples.

Definition 3.4: A language n-tuple $(M_1,...,M_n)$ is said to be included in another language n-tuple $(N_1,...,N_n)$, denoted as $(M_1,...,M_n) \subseteq (N_1,...,N_n)$, iff $M_i \subseteq N_i$ for all $i \in I$. □

Definition 3.5: The union of two language n-tuples $(M_1,...,M_n)$ and $(N_1,...,N_n)$ is defined as $(M_1,...,M_n) \cup (N_1,...,N_n) := (M_1 \cup N_1,...,M_n \cup N_n)$. □

Definition 3.6: The intersection of two language n-tuples $(M_1,...,M_n)$ and $(N_1,...,N_n)$ is defined as $(M_1,...,M_n) \cap (N_1,...,N_n) := (M_1 \cap N_1,...,M_n \cap N_n)$. □

It is obvious that $(Pwr(\Sigma^*))^n$ forms a complete lattice of language n-tuples with bottom element $(\Phi,...,\Phi)$ and top element $(\Sigma^*,...,\Sigma^*)$.

We are now able to solve RNSCP using DRNCT. Theorem 3.1 indicates that the solution to RNSCP is reduced to finding $(M_1,...,M_n) \subseteq (E_1,...,E_n)$ which meets controllability, relative-closure, and consistency conditions. Then, a supervisor could be synthesized in the form of the modular supervisor in Theorem 3.1.

Proposition 3.1: Modular Solution to RNSCP

If a sublanguage n-tuple $(M_1,...,M_n) \subseteq (E_1,...,E_n)$ meets the three conditions in Theorem 3.1, then the modular supervisor $\widetilde{V}$ synthesized in Theorem3.1 will solve RNSCP.

Proof: Follows from the proof of Theorem 3.1. □

Remark 3.2: The notation of consistency of an n-tuple $(M_1,...,M_n)$ was first introduced in [25] in the study of robust supervisory control. In [25], consistency was referred to as "blocking-invariance". There the solution of the robust control problem is provided when the *open-loop* language n-tuple $(L_m(G_1),...,L_m(G_n))$ is consistent with respect to $(L(G_1),...,L(G_n))$. Furthermore, it is shown that under certain assumption on the plants G_1, ... , G_n, if $(L_m(G_1),...,L_m(G_n))$ is not consistent, the original problem can be transferred to an equivalent robust control problem in which the open-loop marked language n-tuple is consistent and thus the robust control problem can be solved. In summary, [25] provides solution for a class of robust control problem. In this thesis, we show that the key to the solution of robust control problem is to consider the consistency of the closed-loop marked behaviors. Through this observation, we obtain all of the solutions of the robust control problem. Furthermore, in our direct approach, we characterize the solutions of the problem using consistent language n-tuples which provides a more transparent setup for the development of online solution (in Chapter 4).

□

Alternatively, a monolithic supervisor can be synthesized as shown in the following Theorem 3.2 for a given sublanguage n-tuple $(M_1,...,M_n) \subseteq (E_1,...,E_n)$ that meets the three conditions in Theorem 3.1. We then show it is equivalent to the modular supervisor.

Theorem 3.2: Alternative Solution to RNSCP:

If a sublanguage n-tuple $(M_1,...,M_n) \subseteq (E_1,...,E_n)$ meets the three conditions in Theorem 3.1, then the monolithic supervisor $\hat{V}: \bigcup_{i\in I} L(G_i) \to 2^{\Sigma}$ with $\hat{V}(s) = \{\sigma \in \Sigma \mid s\sigma \in \bigcup_{i\in I} \overline{M_i}\}$ solves RNSCP.

Proof: Since $\overline{M_i} \subseteq L(G_i)$, we can write $V_i(s) = \begin{cases} \{\sigma \in \Sigma_i \mid s\sigma \in \overline{M_i}\} & \text{if } s \in L(G_i) \\ \Phi & \text{otherwise} \end{cases}$

or simply $V_i(s) = \{\sigma \mid s\sigma \in \overline{M_i}\}$. Thus $\tilde{V}(s) = \bigcup_{i\in I} V_i(s) = \bigcup_{i\in I}\{\sigma \mid s\sigma \in \overline{M_i}\}$ = $\{\sigma \mid s\sigma \in \overline{M_1} \text{ or} \dots \text{or } s\sigma \in \overline{M_n}\} = \{\sigma \mid s\sigma \in \bigcup_{i\in I}\overline{M_i}\} = \hat{V}(s)$. □

Once a set of languages $(M_1,...,M_n) \subseteq (E_1,...,E_n)$ meeting conditions in Theorem 3.1 is obtained, a supervisor V_i can be synthesized for each G_i such that $L(V_i / G_i) = \overline{M_i}$ and the robust control law will be $\tilde{V}(s) = \bigcup_{i\in I} V_i(s)$ to solve RNSCP. Alternatively, a monolithic supervisor $\hat{V}$ can be implemented by constructing an automaton that marks and generates $\bigcup_{i\in I} \overline{M_i}$ to solve RNSCP. The monolithic supervisor $\hat{V}$ is identical to the modular supervisor $\tilde{V}$.

3.2 The Supremal Direct Solution to RNSCP

We investigate the existence of a supremal direct solution to RNSCP in this section. We first generalize the existence of the supremal controllable and the supremal relative-closed sublanguage to the robust situation. The existence of the supremal consistent sublanguage n-tuple is then proved. After that, we prove the existence of the supremal relative-closed, controllable, and consistent sublanguage n-tuple and propose algorithms for its computation.

3.2.1 The Supremal Sublanguages

First we show controllability and relative-closure are preserved under arbitrary unions. Controllable sublanguage n-tuples of $(E_1,...,E_n)$ form an upper semilattice since the arbitrary union of them remains an element in it.

Lemma 3.4: If $(M_1,...,M_n)$ and $(N_1,...,N_n)$ are controllable with respect to $(L(G_1),...,L(G_n))$, then $(M_1,...,M_n) \cup (N_1,...,N_n) = (M_1 \cup N_1,...,M_n \cup N_n)$ is controllable with respect to $(L(G_1),...,L(G_n))$.

Proof: For all $i \in I$, M_i and N_i are controllable with respect to $L(G_i)$ which implies that $M_i \cup N_i$ is controllable with respect to $L(G_i)$. Therefore, $(M_1 \cup N_1,...,M_n \cup N_n)$ is controllable with respect to $(L(G_1),...,L(G_n))$ by definition. □

Lemma 3.4 can be proved for arbitrary unions. Furthermore, $(\Phi,...,\Phi)$ is controllable with respect to $(L(G_1),...,L(G_n))$. As a result, if a language n-tuple $(E_1,...,E_n)$ is not controllable, there always exists a supremal sublanguage n-tuple of $(E_1,...,E_n)$ which is controllable. In other words, the supremal controllable sublanguage n-tuple denoted as

$SupC((E_1,...,E_n))$ is well defined and it is an element inside the upper semilattice of controllable sublanguage n-tuples.

Similarly, the set of relative-closed sublanguage n-tuples of $(E_1,...,E_n)$ forms an upper semilattice. The supremal relative-closed element $SupR((E_1,...,E_n))$ is well defined and it is an element inside the upper semilattice of relative-closed sublanguage n-tuples.

Lemma 3.5: If $(M_1,...,M_n)$ and $(N_1,...,N_n)$ are relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$, then $(M_1,...,M_n) \cup (N_1,...,N_n) = (M_1 \cup N_1,...,M_n \cup N_n)$ is relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$.

Proof: For all $i \in I$, M_i and N_i are relative-closed with respect to $L_m(G_i)$ which implies that $M_i \cup N_i$ is relative-closed with respect to $L_m(G_i)$. Therefore, $(M_1 \cup N_1,...,M_n \cup N_n)$ is relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$ by definition. □

Lemma 3.5 can be proved for arbitrary unions. Furthermore, $(\Phi,...,\Phi)$ is relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$. As a result, if a language n-tuple $(E_1,...,E_n)$ is not relative-closed, there always exists a supremal sublanguage n-tuple which is relative-closed. In other words, the supremal relative-closed sublanguage n-tuple denoted as $SupR((E_1,...,E_n))$ is well defined. It is inside the upper semilattice of relative-closed sublanguage n-tuples.

We now show that consistent sublanguage n-tuples also form an upper semilattice. The following lemma shows that consistency is also preserved under union operation. In addition, there always exists a supremal consistent sublanguage n-tuple of $(E_1,...,E_n)$ denoted as $SupS((E_1,...,E_n))$.

Lemma 3.6: If $(M_1,...,M_n)$ and $(N_1,...,N_n)$ are consistent with respect to $(L(G_1),...,L(G_n))$, then $(M_1,...,M_n) \cup (N_1,...,N_n) = (M_1 \cup N_1,...,M_n \cup N_n)$ is consistent with respect to $(L(G_1),...,L(G_n))$.

Proof: We only need to show $\overline{M_i \mathrm{Y} N_i} \mathrm{I}\ (L(G_j) - \overline{M_j \mathrm{Y} N_j}) = \Phi$ for all $i, j \in I$.

$$
\begin{aligned}
\overline{M_i \mathrm{Y} N_i} \mathrm{I}\ (L(G_j) - \overline{M_j \mathrm{Y} N_j}) &= (\overline{M_i} \mathrm{Y} \overline{N_i}) \mathrm{I}\ (L(G_j) - \overline{M_j} \mathrm{Y} \overline{N_j}) \\
&= [\overline{M_i} \mathrm{I}\ (L(G_j) - \overline{M_j} \mathrm{Y} \overline{N_j})] \mathrm{Y} [\overline{N_i} \mathrm{I}\ (L(G_j) - \overline{M_j} \mathrm{Y} \overline{N_j})] \\
&\subseteq [\overline{M_i} \mathrm{I}\ (L(G_j) - \overline{M_j})] \mathrm{Y} [\overline{N_i} \mathrm{I}\ (L(G_j) - \overline{N_j})] \\
&= \Phi
\end{aligned}
$$

Hence, we infer that $\overline{M_i \mathrm{Y} N_i} \mathrm{I}\ (L(G_j) - \overline{M_j \mathrm{Y} N_j}) = \Phi$ for all $i, j \in I$. □

Definition 3.7: The class of consistent sublanguage n-tuples of a language n-tuple $(E_1,...,E_n)$ is denoted as $S((E_1,...,E_n))$ and defined as:

$$S((E_1,...,E_n)) := \{(K_1,...,K_n) \subseteq (E_1,...,E_n) \mid \forall i, j \in I, \overline{K_i} \cap (L(G_j) - \overline{K_j}) = \Phi\}. \quad \square$$

Obviously, $(\Phi,...,\Phi)$ is an element of $S((E_1,...,E_n))$ and Lemma 3.6 can be generalized to arbitrary unions. We take the union of all the elements of $S((E_1,...,E_n))$ and denote it as $SupS((E_1,...,E_n))$, which belongs to $S((E_1,...,E_n))$, and is the supremal consistent sublanguage n-tuple of $(E_1,...,E_n)$.

Lemma 3.7: The supremal consistent sublanguage n-tuple of $(E_1,...,E_n)$ denoted as $SupS((E_1,...,E_n))$ exists. □

We next define an operator Ω and prove that a sublanguage n-tuple of $(E_1,...,E_n)$ is consistent if and only if it is a fixed point of Ω. The implementation of operator Ω in TTCT is given by Procedure 3.1 in Appendix 3.1.

Definition 3.8: The operator $\Omega : (Pwr(E_1),...,Pwr(E_n)) \rightarrow (Pwr(E_1),...,Pwr(E_n))$ is defined on sublanguage n-tuples of $(E_1,...,E_n)$ according to $\Omega((K_1,...,K_n)) = (K_1 - D\Sigma^*,...,K_n - D\Sigma^*)$ where $D = \bigcup_{i\in I}(L(G_i) - \overline{K_i})$. □

Theorem 3.3: Given $(E_1,...,E_n) \subseteq (L(G_1),...,L(G_n))$, a sublanguage n-tuple of $(E_1,...,E_n)$ is consistent if and only if it is a fixed point of the operator Ω. □

We need the following lemmas to prove Theorem 3.3.

Lemma 3.8: If $(K_1,...,K_n)$ is a fixed point of Ω, then $s \in \overline{K_i} \cap L(G_j)$ implies $s \in \overline{K_j}$.

Proof: (By Contradiction) It follows from the assumption that $K_i = K_i - D\Sigma^*$ and thus $K_i \cap D\Sigma^* = \Phi$ which implies $\overline{K_i} \cap D\Sigma^* = \Phi$ by Lemma 2.4. Let $s \in \overline{K_i} \cap L(G_j)$. Suppose $s \notin \overline{K_j}$.

$$\begin{aligned} s \in L(G_j) \wedge s \notin \overline{K_j} &\Rightarrow s \in L(G_j) - \overline{K_j} \\ &\Rightarrow s \in D \\ &\Rightarrow s \in D\Sigma^* \end{aligned}$$

$$\Rightarrow s \notin \overline{K_i}$$

It is in contradiction with $s \in \overline{K_i}$. As a result, we have $s \in \overline{K_j}$. □

Lemma 3.9: If $(K_1,...,K_n)$ is a fixed point of Ω, then it is consistent.

Proof: We need to show that $\overline{K_i} \cap (L(G_j) - \overline{K_j}) = \Phi$ for all $i, j \in I$. If $\overline{K_i} = \Phi$, then $\overline{K_i} \cap (L(G_j) - \overline{K_j}) = \Phi$. If $\overline{K_i} \neq \Phi$, then let $s \in \overline{K_i}$. We have only two cases for any string s. If $s \notin L(G_j)$, it implies that $s \notin L(G_j) - \overline{K_j}$. If $s \in L(G_j)$, it implies $s \in \overline{K_j}$ by Lemma 3.8 and thus $s \notin L(G_j) - \overline{K_j}$. As a result, $\overline{K_i} \cap (L(G_j) - \overline{K_j}) = \Phi$. □

Lemma 3.10 shows that the reverse direction of the above lemma holds.

Lemma 3.10: If $(K_1,...,K_n)$ is a consistent sublanguage n-tuple of $(E_1,...,E_n)$, then it is a fixed point of Ω.

Proof: For every $i, j \in I$:

$$\overline{K_i} \cap (L(G_j) - \overline{K_j}) = \Phi \Rightarrow \overline{K_i} \cap [\bigcup_{j \in I} (L(G_j) - \overline{K_j})] = \Phi$$

$$\Rightarrow \overline{K_i} \cap D = \Phi$$

$$\Rightarrow K_i \cap D\Sigma^* = \Phi \qquad \text{(by Lemma 2.4)}$$

$$\Rightarrow K_i = K_i - D\Sigma^*.$$

Therefore, $(K_1,...,K_n)$ is a fixed point of Ω. □

Proof of Theorem 3.3: Follows from Lemma 3.9 and Lemma 3.10. □

Obviously, $(\Phi,...,\Phi)$ is a fixed point of Ω. Furthermore, if $(K_1,...,K_n)$ and $(K_1',...,K_n')$ are fixed points of Ω, then $(K_1,...,K_n) \cup (K_1',...,K_n')$ is a fixed point of Ω. Therefore, there exists a largest fixed point of Ω denoted by $(K_1^*,...,K_n^*)$.

Corollary 3.1: Given $(E_1,...,E_n) \subseteq (L(G_1),...,L(G_n))$, $SupS((E_1,...,E_n))$ is the largest fixed point $(K_1^*,...,K_n^*)$ of the operator Ω.

Proof: We have to show that $(K_1^*,...,K_n^*) = SupS((E_1,...,E_n))$. First, the largest fixed point $(K_1^*,...,K_n^*)$ is a fixed point and thus consistent with respect to $(L(G_1),...,L(G_n))$ by Lemma 3.9. Therefore, $(K_1^*,...,K_n^*) \subseteq SupS((E_1,...,E_n))$. In addition, $SupS((E_1,...,E_n))$ is a consistent sublanguage n-tuple of $(E_1,...,E_n)$ and thus a fixed point of Ω by lemma 3.10. Therefore, $SupS((E_1,...,E_n)) \subseteq (K_1^*,...,K_n^*)$. As a result, we conclude that $(K_1^*,...,K_n^*) = SupS((E_1,...,E_n))$. □

We have proved that the supremal consistent sublanguage exists and is the largest fixed point of operator Ω. We need to propose an algorithm to find the largest fixed point and thus equivalently the supremal consistent sublanguage. The following algorithm computes the supremal consistent sublanguage n-tuple if it terminates in a finite number of steps.

Algorithm 3.1:

1) Initialization: $(K_1^0,...,K_n^0)=(E_1,...,E_n)$.

2) Update $(K_1^{j+1},...,K_n^{j+1})=\Omega((K_1^j,...,K_n^j)), j\geq 0$.

3) If $K_i^{j+1}\neq K_i^j$ for some $1\leq i\leq n$, then go back to step 2. □

Remark 3.3: If $(E_1,...,E_n)$ or $(L(G_1),...,L(G_n))$ has a finite number of strings, then Algorithm 3.1 will converge in a finite number of steps. For example, the finite expansion trees of VLP algorithm in Chapter 4 have a finite number of strings. □

Theorem 3.4 indicates that the above algorithm will compute the largest fixed point $(K_1^*,...,K_n^*)$ of Ω for $(E_1,...,E_n)$ and thus $SupS((E_1,...,E_n))$ if it terminates. The implementation of Algorithm 3.1 in TTCT is given by Procedure 3.3 in Appendix 3.1.

Theorem 3.4: If Algorithm 3.1 terminates in a finite number of steps, say m, then $(K_1^m,...,K_n^m)=SupS((E_1,...,E_n))$. □

Lemma 3.11 shows that Ω is a monotone operator, which is used for the proof of Theorem 3.4.

Lemma 3.11: If $(M_1,...,M_n)\subseteq(N_1,...,N_n)$, then $\Omega((M_1,...,M_n))\subseteq\Omega((N_1,...,N_n))$.

Proof: $\Omega((M_1,...,M_n))=(M_1-D_m\Sigma^*,...,M_n-D_m\Sigma^*)$ where $D_m=\bigcup_{i\in I}(L(G_i)-\overline{M_i})$.

$\Omega((N_1,...,N_n))=(N_1-D_m\Sigma^*,...,N_n-D_m\Sigma^*)$ where $D_n=\bigcup_{i\in I}(L(G_i)-\overline{N_i})$.

$$\begin{aligned} \forall i \in I, M_i \subseteq N_i &\Rightarrow \overline{M_i} \subseteq \overline{N_i} \\ &\Rightarrow \forall i \in I, L(G_i) - \overline{M_i} \supseteq L(G_i) - \overline{N_i} \\ &\Rightarrow D_m \supseteq D_n \\ &\Rightarrow D_m\Sigma^* \supseteq D_n\Sigma^*. \end{aligned}$$

$$\begin{aligned} \forall i \in I, M_i - D_m\Sigma^* &\subseteq N_i - D_m\Sigma^* \\ &\subseteq N_i - D_n\Sigma^*. \end{aligned}$$

Therefore, we have $\Omega((M_1,...,M_n)) \subseteq \Omega((N_1,...,N_n))$. □

Proof of Theorem 3.4: 1) first, we show that $(K_1^m,...,K_n^m)$ is a fixed point of Ω. If the algorithm terminates in a finite number of steps, then $(K_1^m,...,K_n^m)$ satisfies $(K_1^m,...,K_n^m) = \Omega((K_1^m,...,K_n^m))$ and thus $(K_1^m,...,K_n^m)$ is indeed a fixed point of Ω. Therefore, $(K_1^m,...,K_n^m)$ is a consistent sublanguage n-tuple of $(E_1,...,E_n)$ by Lemma 3.9, and thus $(K_1^m,...,K_n^m) \subseteq SupS((E_1,...,E_n))$.

2) Secondly, we prove $SupS((E_1,...,E_n)) \subseteq (K_1^m,...,K_n^m)$ by induction. Initially, it is obvious that $SupS((E_1,...,E_n)) \subseteq (E_1,...,E_n) = (K_1^0,...,K_n^0)$. The induction base holds. Then, we assume that $SupS((E_1,...,E_n)) \subseteq (K_1^j,...,K_n^j)$. We need to show that $SupS((E_1,...,E_n)) \subseteq (K_1^{j+1},...,K_n^{j+1})$. From Lemma 3.10 and 3.11, We have $SupS((E_1,...,E_n)) = \Omega(SupS((E_1,...,E_n)) \subseteq \Omega((K_1^j,...,K_n^j)) = (K_1^{j+1},...,K_n^{j+1})$. Therefore, we have $SupS((E_1,...,E_n)) \subseteq (K_1^m,...,K_n^m)$.

From 1) and 2), we conclude that $(K_1^m,...,K_n^m) = SupS((E_1,...,E_n))$. □

Algorithm 3.1 is used to calculate the supremal consistent sublanguage of the problem in Example 2.1. The following example illustrates the procedure of Algorithm 3.1.

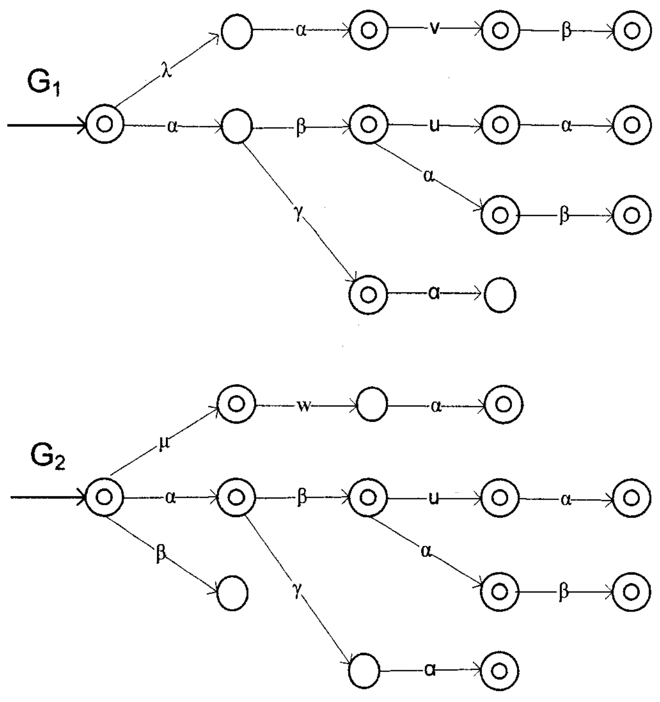

Figure 3. 3: Plant automata for Example 3.4

Example 3.4: A system model could be either G_1 and G_2 shown in Figure 3.3. Assume that all events $\Sigma = \{\alpha, \beta, \gamma, \lambda, \mu, u, v, w\}$ are controllable. We show how to compute the supremal consistent sublanguage of given specifications.

The language generated and marked by the plant models are:

$L(G_1) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \lambda, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$

$L(G_2) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \mu, \mu w, \mu w\alpha, \beta\}$

$L_m(G_1) = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$

$L_m(G_2) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu, \mu w\alpha\}$.

Assume that the specification languages are:

$E_1 = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$

$E_2 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu\}$.

We can verify the specifications are not consistent with each other. For example, $\alpha\gamma\alpha \in \overline{E_2} \cap (L(G_1) - \overline{E_1})$ which implies $\overline{E_2} \cap (L(G_1) - \overline{E_1}) \neq \Phi$. We now employ Algorithm 3.1 to compute the supremal consistent sublanguage of the specifications.

Initialization:

$K_1^0 = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$K_2^0 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu\}$.

Iteration 1:

$D = \{\alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \beta, \mu w, \mu w\alpha\}$.

$K_1^1 = K_1^0 - D\Sigma^* = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$K_2^1 = K_2^0 - D\Sigma^* = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \mu\}$.

K_1^1 does not change from K_1^0, but some strings are removed from K_2^0 in comparison with K_1^0. Thus, we proceed to the next iteration.

Iteration 2:

$D = \{\alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \beta, \mu w, \mu w\alpha\}$

$K_1^2 = K_1^1 - D\Sigma^* = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$

$K_2^2 = K_2^1 - D\Sigma^* = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \mu\}$

Both K_1^2 and K_2^2 change after this iteration, thus the next iteration has to be taken.

Iteration 3:

$D = \{\alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \beta, \mu w, \mu w\alpha\}$.

$K_1^3 = K_1^2 - D\Sigma^* = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$K_2^3 = K_2^2 - D\Sigma^* = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \mu\}$.

Neither K_1^3 nor K_1^3 changes thus we stop here. The resulting supremal consistent sublanguages are $K_1^* = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$ and $K_2^* = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \mu\}$. □

We have defined the supremal controllable sublanguage, the supremal relative-closed sublanguage, and the supremal consistent sublanguage in this subsection. In the next subsection, they are used to compute the supremal relative-closed, controllable, and consistent sublanguage, which characterizes the optimal solution to RNSCP.

3.2.2 The Optimal Solution to RNSCP

If a language n-tuple $(E_1, ..., E_n)$ is not relative-closed, controllable, and consistent, there may exist many sublanguage n-tuples which are relative-closed, controllable, and consistent. The collection of all relative-closed, controllable, and consistent sublanguage n-tuples is denoted as $RCS((E_1,...,E_n))$. Any direct solution to RNSCP is characterized by an element of $RCS((E_1,...,E_n))$.

Definition 3.9: $RCS((E_1,...,E_n))$ represents the collection of all sublanguage n-tuples of $(E_1,...,E_n)$ that are controllable and consistent with respect to $(L(G_1),...,L(G_n))$ and relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$. □

We show that $RCS((E_1,...,E_n))$ forms an upper semilattice. Lemma 3.12 shows that the union of two language n-tuples in $RCS((E_1,...,E_n))$ still belongs to $RCS((E_1,...,E_n))$.

Lemma 3.12: If $(M_1,...,M_n)$ and $(N_1,...,N_n)$ belong to $RCS((E_1,...,E_n))$, then $(M_1,...,M_n) \cup (N_1,...,N_n)$ will belong to $RCS((E_1,...,E_n))$.

Proof: Follows from Lemma 3.4, 3.5, and 3.6 and definition of $RCS((E_1,...,E_n))$. □

Lemma 3.12 can be easily generalized to arbitrary union. Furthermore, $(\Phi,...,\Phi)$ is the bottom element of $RCS((E_1,...,E_n))$. Therefore, the supremal relative-closed, controllable, and consistent sublanguage denoted as $SupRCS((E_1,...,E_n))$ is well defined and can be obtained from the union of all elements in $RCS((E_1,...,E_n))$. Thus, we have the following lemma.

Lemma 3.13: 1) $SupRCS((E_1,...,E_n))$ exists.

2) $SupRCS((E_1,...,E_n)) = \Upsilon\{\underline{K} \mid \underline{K} \in RCS((E_1,...,E_n))\}$.

3) $SupRCS((E_1,...,E_n)) \in RCS((E_1,...,E_n))$. □

Remark 3.4: We expect that we can derive the indirect optimal solution $K = SupRN_bC(E)$ from the direct optimal solution $(K_1,...,K_n) = SupRCS((E_1,...,E_n))$, and vice versa. They are both optimal solutions to RNSCP from different approaches. It is not difficult to prove that $K = \bigcup_{i \in I} K_i$ and $K_i = K \cap L_m(G_i)$. □

We then define operator $\underline{\Omega}$ and show that a sublanguage n-tuple of $(E_1,...,E_n)$ is in $RCS((E_1,...,E_n))$ if and only if it is a fixed point of $\underline{\Omega}$.

Definition 3.10: The operator $\underline{\Omega} : (Pwr(E_1),...,Pwr(E_n)) \to (Pwr(E_1),...,Pwr(E_n))$ is defined on sublanguage n-tuples of $(E_1,...,E_n)$ according to $\underline{\Omega}((K_1,...,K_n)) = SupS(SupC(SupR((K_1,...,K_n))))$. □

Theorem 3.5: A language n-tuple is in $RCS((E_1,...,E_n))$ if and only if it is a fixed point of $\underline{\Omega}$. □

We need the following lemmas to prove Theorem 3.5.

Lemma 3.14 : If $(K_1,...,K_n)$ is a fixed point of $\underline{\Omega}$, then it is in $RCS((E_1,...,E_n))$.

Proof: $SupS(SupC(SupR((K_1,...,K_n)))) = (K_1,...,K_n)$ since $(K_1,...,K_n)$ is a fixed point of $\underline{\Omega}$. Obviously, $SupR((K_1,...,K_n)) \subseteq (K_1,...,K_n)$. In addition,

$$\begin{aligned}(K_1,...,K_n) &= SupS(SupC(SupR((K_1,...,K_n))))\\ &\subseteq SupC(SupR((K_1,...,K_n)))\end{aligned}$$

$\subseteq SupR((K_1,...,K_n))$.

Thus, $SupR((K_1,...,K_n)) = (K_1,...,K_n)$. Similarly, we have $SupC((K_1,...,K_n)) = (K_1,...,K_n)$ and $SupS((K_1,...,K_n)) = (K_1,...,K_n)$.

Therefore, $(K_1,...,K_n)$ is controllable and consistent with respect to $(L(G_1),...,L(G_n))$, and relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$. As a result, $(K_1,...,K_n)$ is an element in $RCS((E_1,...,E_n))$. □

Lemma 3.15 : If $(K_1,...,K_n) \in RCS((E_1,...,E_n))$, then it is a fixed point of $\underline{\Omega}$.

Proof: $(K_1,...,K_n)$ is controllable, consistent, and relative-closed. Therefore, $\underline{\Omega}((K_1,...,K_n)) = SupS(SupC(SupR((K_1,...,K_n)))) = (K_1,...,K_n)$, which shows that $(K_1,...,K_n)$ is a fixed point of $\underline{\Omega}$. □

Proof of Theorem 3.5: Follows from Lemma 3.14 and 3.15. □

Corollary 3.2: The largest fixed point $(K_1^*,...,K_n^*)$ of $\underline{\Omega}$ exists and is equal to $SupRCS((E_1,...,E_n))$.

Proof: $(\Phi,...,\Phi)$ is a fixed point of $\underline{\Omega}$. Furthermore, if $(K_1,...,K_n)$ and $(K_1',...,K_n')$ are fixed points of $\underline{\Omega}$, then so is $(K_1,...,K_n) \cup (K_1',...,K_n')$. Therefore, the largest fixed point of $\underline{\Omega}$ exists.We have to show that $(K_1^*,...,K_n^*) = SupRCS((E_1,...,E_n))$. First, the largest fixed point $(K_1^*,...,K_n^*)$ is a fixed point and thus an element in $RCS((E_1,...,E_n))$ by Lemma 3.14. Therefore, $(K_1^*,...,K_n^*) \subseteq SupRCS((E_1,...,E_n))$. In addition,

$SupRCS((E_1,...,E_n))$ is an element in $RCS((E_1,...,E_n))$ and thus a fixed point of Ω by Lemma 3.15. Therefore, $SupRCS((E_1,...,E_n)) \subseteq (K_1^*,...,K_n^*)$. As a result, we conclude that $(K_1^*,...,K_n^*) = SupRCS((E_1,...,E_n))$. □

Up to now, we have shown the union of solutions to RNSCP is still a solution, and there exists a supremal solution to RNSCP. Optimal solution to robust nonblocking supervisory control problem reduces to the calculation of $SupRCS((E_1,...,E_n))$. We next propose an algorithm to compute $SupRCS((E_1,...,E_n))$.

Algorithm 3.2:

1) Initialization: $(K_1^0,...,K_n^0) = (E_1,...,E_n)$.

2) Update $(K_1^{j+1},...,K_n^{j+1}) = \underline{\Omega}((K_1^j,...,K_n^j)), j \geq 0$.

3) If $K_i^{j+1} \neq K_i^j$ for some $1 \leq i \leq n$, then go to step 2. □

Theorem 3.6 indicates that the above algorithm indeed computes the supremal sublanguage if it terminates with $(K_1^m,...,K_n^m)$ at the m^{th} step. Namely, $(K_1^m,...,K_n^m) = SupRCS((E_1,...,E_n))$. We first prove $\underline{\Omega}$ is a monotone operator in Lemma 3.16. The monotone property is the key issue of the proof of Theorem 3.6.

Lemma 3.16: If $(M_1,...,M_n) \subseteq (N_1,...,N_n)$, then $\underline{\Omega}((M_1,...,M_n)) \subseteq \underline{\Omega}((N_1,...,N_n))$.

Proof: $\underline{\Omega}((M_1,...,M_n)) = SupS(SupC(SupR((M_1,...,M_n))))$

$$\subseteq SupS(SupC(SupR((N_1,...,N_n))))$$

$= \underline{\Omega}((N_1,...,N_n))$.

Therefore, we have $\underline{\Omega}((M_1,...,M_n)) \subseteq \underline{\Omega}((N_1,...,N_n))$. □

Theorem 3.6: If Algorithm 3.2 terminates in m steps to $(K_1^m,...,K_n^m)$, then $(K_1^m,...,K_n^m) = SupRCS((E_1,...,E_n))$.

Proof: a) First, we show that $(K_1^m,...,K_n^m)$ is a fixed point of $\underline{\Omega}$. If the algorithm terminates in m iterations, then its result $(K_1^m,...,K_n^m)$ satisfies $(K_1^m,...,K_n^m) = \underline{\Omega}((K_1^m,...,K_n^m))$ and thus $(K_1^m,...,K_n^m)$ is a fixed point of $\underline{\Omega}$. Hence, $(K_1^m,...,K_n^m)$ is in $RCS((E_1,...,E_n))$ by Theorem 3.5, which implies that $(K_1^m,...,K_n^m) \subseteq SupRCS((E_1,...,E_n))$.

b) Secondly, we prove that $SupRCS((E_1,...,E_n)) \subseteq (K_1^m,...,K_n^m)$ by induction. The induction base holds since initially $SupRCS((E_1,...,E_n)) \subseteq (E_1,...,E_n) = (K_1^0,...,K_n^0)$. Now we suppose $SupRCS((E_1,...,E_n)) \subseteq (K_1^j,...,K_n^j)$ and show that $SupRCS((E_1,...,E_n)) \subseteq (K_1^{j+1},...,K_n^{j+1})$. This is true since $SupRCS((E_1,...,E_n)) = \underline{\Omega}(SupRCS((E_1,...,E_n))) \subseteq \underline{\Omega}((K_1^j,...,K_n^j)) = (K_1^{j+1},...,K_n^{j+1})$ (The first equality results from Theorem 3.5 and the inequality from Lemma 3.16). Therefore, we have $SupRCS((E_1,...,E_n)) \subseteq (K_1^m,...,K_n^m)$.

From a) and b), we conclude that $(K_1^*,...,K_n^*) = SupRCS((E_1,...,E_n))$. □

3.2.3 Alterative Algorithm

We next discuss some properties of relative-closed language n-tuples which enable us to modify Algorithm 3.2. It is straightforward to generalize traditional results that the supremal controllable sublangauge of a relative-closed language remains relative-closed to the case of robust control.

Lemma 3.17: If a language n-tuple $(K_1,...,K_n)$ is relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$, then its supremal controllable sublanguage n-tuple $SupR((K_1,...,K_n))$ will remain relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$.

Proof: Follows from Lemma 2.5 and the definition of relative-closure (Definition 3.2). □

Similarly, Lemma 3.18 shows that the supremal consistent sublanguage n-tuple preserves the relative-closure property.

Lemma 3.18: If a language n-tuple $(K_1,...,K_n)$ is relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$, then its supremal consistent sublanguage n-tuple $SupS((K_1,...,K_n))$ denoted as $(K_1^*,...,K_n^*)$ will remain relative-closed with respect to $(L_m(G_1),...,L_m(G_n))$.

Proof: Denote $\overline{K_i^*} \cap L_m(G_i)$ as $K_i^'$. We have to show that $(K_1^*,...,K_n^*) = (K_1^',...,K_n^')$. It is immediate that $(K_1^*,...,K_n^*) \subseteq (K_1^',...,K_n^')$. We then show that $(K_1^',...,K_n^')$ is consistent and $(K_1^',...,K_n^') \subseteq (K_1,...,K_n)$, which implies $(K_1^',...,K_n^') \subseteq (K_1^*,...,K_n^*)$. We have

$\overline{K_i^*} = \overline{K_i'}$ since $\overline{K_i^*} \subseteq \overline{\overline{K_i^*} \cap L_m(G_i)} = \overline{K_i'}$ and $\overline{K_i'} = \overline{\overline{K_i^*} \cap L_m(G_i)} \subseteq \overline{\overline{K_i^*}} = \overline{K_i^*}$. Hence, $(K_1',...,K_n')$ is consistent since $(K_1^*,...,K_n^*)$ is consistent. In addition, $K_i' = \overline{K_i^*} \cap L_m(G_i) \subseteq \overline{K_i} \cap L_m(G_i) = K_i$ for all $i \in I$. Thus, $(K_1',...,K_n') \subseteq (K_1^*,...,K_n^*)$. As a result, $(K_1^*,...,K_n^*) = (K_1',...,K_n')$. □

If $(K_1,...,K_n)$ is relative-closed, $\underline{\Omega}((K_1,...,K_n))$ $= SupS(SupC(SupR((K_1,...,K_n))))$ reduces to $\underline{\Omega}((K_1,...,K_n))$ $= SupS(SupC((K_1,...,K_n)))$. From the above lemmas, we observe that $\underline{\Omega}((K_1,...,K_n))$ $= SupS(SupC((K_1,...,K_n)))$ remains relative-closed if $(K_1,...,K_n)$ is relative-closed. Therefore, the supremal relative-closure operation only needs to be conducted at the beginning of Algorithm 3.2.

Based on the above observation, Algorithm 3.2 can be modified to Algorithm 3.2A. Theorem 3.7 shows the correctness of Algorithm 3.2A. The implementation of Algorithm 3.2A is provided by Procedure 3.5 in Appendix 3.1.

Algorithm 3.2A:

1) Initialization: $(K_1^0,...,K_n^0) = SupR((E_1,...,E_n))$.

2) Update $(K_1^{j+1},...,K_n^{j+1}) = SupS(SupC((K_1^j,...,K_n^j))), j \geq 0$.

3) If $K_i^{j+1} \neq K_i^j$ for some $1 \leq i \leq n$, then go to step 2. □

Theorem 3.7: If Algorithm 3.2A terminates, it will calculate $SupRCS((E_1,...,E_n))$.

Proof: We only need to show the relative-closure property is always satisfied during the iteration by induction. The induction base holds since $(K_1^0,...,K_n^0) = SupR((E_1,...,E_n))$ is relative-closed. We now assume $(K_1^j,...,K_n^j)$ is relative-closed. We have to show $(K_1^{j+1},...,K_n^{j+1})$ remains relative closed. $SupC((K_1^j,...,K_n^j))$ is relative-closed by Lemma 3.17. In addition, $SupS(SupC((K_1^j,...,K_n^j)))$ is still relative-closed by Lemma 3.18. As a result, $(K_1^{j+1},...,K_n^{j+1})$ maintains relative-closed. □

Consistency has clear physical meaning and offers insight to robustness in RNSCP. In addition, we do not synthesize the overall specification and plant language in the direct approach.

Offline direct approach Algorithm 3.2A is used to solve the same problem in Example 2.1. The following example illustrates the procedure of offline direct approach. The result is exactly the same as the solution of offline indirect approach in Example 2.1.

Example 3.5: A system model could be either G_1 or G_2 shown below. Assume that $\Sigma = \{\alpha,\beta,\gamma,\lambda,\mu,u,v,w\}$ with $\Sigma_c = \{\alpha,\beta,\gamma,\lambda,\mu\}$ and $\Sigma_u = \{u,v,w\}$. We synthesize the optimal admissible nonblocking robust supervisor for given specifications.

The language generated and marked by the plant models are:

$L(G_1) = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma,\alpha\gamma\alpha,\lambda,\lambda\alpha,\lambda\alpha v,\lambda\alpha v\beta\}$

$L(G_2) = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma,\alpha\gamma\alpha,\mu,\mu w,\mu w\alpha,\beta\}$

$L_m(G_1) = \{\varepsilon,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma,\lambda\alpha,\lambda\alpha v,\lambda\alpha v\beta\}$

$L_m(G_2) = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma\alpha,\mu,\mu w\alpha\}$.

Assume that the specification languages are:

$E_1 = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$E_2 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu\}$.

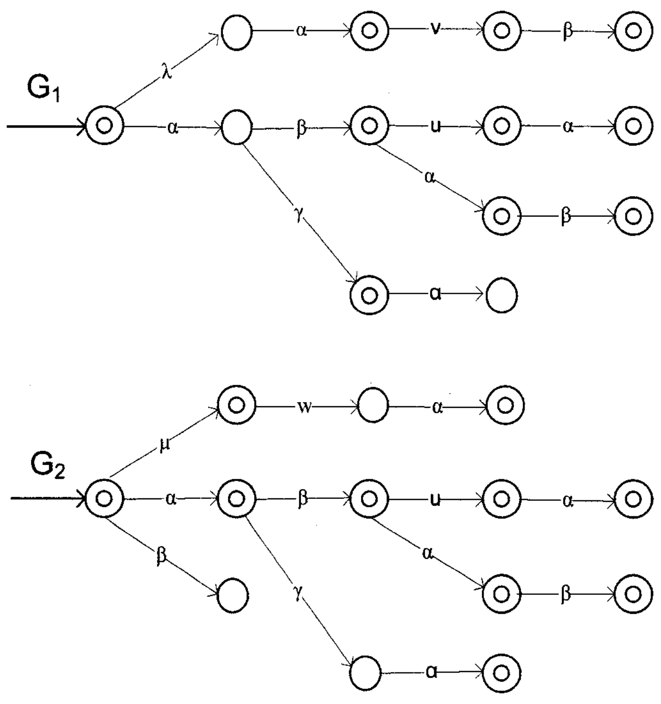

Figure 3. 4: Plant automata for Example 3.5

We now employ Algorithm 3.2A to compute the supremal solution to the robust problem.

Initialization:

$K_1^0 = SupR(E_1) = E_1$.

$K_2^0 = SupR(E_2) = E_2$.

Iteration 1:

K_1^0 is controllable, however, K_2^0 is not controllable since $\mu \in \overline{K_2^0}$ and $\mu w \in \overline{K_2^0}\Sigma_u \cap L(G_2)$ but $\mu w \notin \overline{K_2^0}$. The supermal controllable sublanguages are first calculated in this iteration.

$SupC(K_1^0) = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$SupC(K_2^0) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha\}$.

Obviously, $(SupC(K_1^0), SupC(K_2^0))$ is not consistent. For example, $\alpha\beta\alpha$ and $\alpha\gamma\alpha$ are disabled in G_1 but enabled in G_2. We next calculate the supremal consistent sublanguage in this iteration.

$K_1^1 = SupS(SupC(K_1^0)) = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$K_2^1 = SupS(SupC(K_2^0)) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha\}$.

After this iteration, K_1^1 and K_2^1 are different with K_1^0 and K_2^0 respectively and thus next iteration has to be preformed.

Iteration 2:

$SupC(K_1^1) = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$SupC(K_2^1) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha\}$.

$K_1^2 = SupS(SupC(K_1^1)) = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$.

$K_2^2 = SupS(SupC(K_2^1)) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha\}$.

Neither K_1^2 nor K_2^2 changes from K_1^1 and K_2^1 respectively. Hence, we stop here. The optimal direct solution to this problem is $(K_1, K_2) = (K_1^2, K_2^2)$. We can verify that $K_1 \cup K_2 = K = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$ where K is the indirect solution in Example 2.1. □

Once Algorithm 3.2A terminates, the three conditions in DRNCT are satisfied and the supremal solution is obtained. Then, a modular or a monolithic robust supervisor can be realized to ensure the specifications and the nonblocking requirement.

In the next section, we consider an application of robust supervisory control in the solution of supervisory control with multiple sets of marked states.

3.3 Supervisory Control with Multiple Sets of Marked States

The dynamics of a discrete event system can be modeled with automaton. The prefix-closed language generated by the automaton represents all the possible sequences that can be generated by the system. Among them, some sequences of an automaton are marked. The marked sequences indicate completion of tasks.

In different situations, we may choose, instead of a single, multiple sets of sequences to mark since we would like different targets to be reached. Traditional supervisory control has to design a supervisor for each set of marked states. For example, in a propulsion system, we only mark the sequences which lead to states in which the engine is on in the case that we want a propulsion system to be turned on. In other situation, we only mark the sequences which take the system to states with engine off if the target is to cut off the propulsion system.

Instead of designing a supervisor for each set of marked states, we would like to design a supervisor for different set of marked states for simplicity. Traditional supervisory control cannot satisfy this requirement. However, the problem can be solved as a special case of robust nonblocking supervisory control problems. We form the nonblocking supervisory control problem with multiple sets of marked states and convert it to a corresponding robust supervisory control problem. We only investigate the solution to the corresponding robust supervisory control problem since solving it will automatically address the original problem with multiple sets of marked states.

3.3.1 Problem Formulation

We consider the nonblocking supervisory control problem for a system with multiple sets of marked states. The definition of such a system and its corresponding languages are first given.

Definition 3.11: A plant with multiple sets of marked states can be represented as an automaton $G = (X, \Sigma, \delta, x_0, X_{m_1}, ..., X_{m_n})$. $L_m^{(i)}(G)$ represents the marked behavior with only states in X_{m_i} marked. □

We assume that all events are observable. The supervisory control problem with multiple sets of marked states is formally described as follows. From now on, we refer to multiple sets of marked states as multiple markings instead for brevity.

Nonblocking Supervisory Control Problem with Multiple Markings (NSCP-MM)

Given a plant with multiple marking $G = (X, \Sigma, \delta, x_0, X_{m_1}, ..., X_{m_n})$ and specifications $E_i \subseteq L_m^{(i)}(G)$ for all $i \in I = \{1,2,...,n\}$, synthesize a supervisor S such that $L_m^{(i)}(S/G) = L(S/G) \cap L_m^{(i)}(G) \subseteq E_i$ and $\overline{L_m^{(i)}(S/G)} = L(S/G)$ for all $i \in I$. □

The above problem is not equivalent to designing a supervisor for $GA = (X, \Sigma, \delta, x_0, X_{m_1} \cup \cdots \cup X_{m_n})$ with specification $E = \bigcup_{i \in I} E_i$ or designing a supervisor for $GB = (X, \Sigma, \delta, x_0, X_{m_1} \cap \cdots \cap X_{m_n})$ with specification $E = \bigcap_{i \in I} E_i$. The following examples illustrate these.

Example 3.6: A system has event set $\Sigma = \{a, b\}$ where event a represents turning on the system and event b means turning off the system. Both a and b are controllable. The system model G is given in Figure 3.5. State 0 represents that the system is ready. States 1 and 2 stand for system on and off respectively. In some situation, we choose states 0 and 1 marked as X_{m_1}. In another, states 0 and 2 are marked as X_{m_2}. Assume all strings are legal. We design a supervisor which is nonblocking in both marking situations X_{m_1} and X_{m_2}.

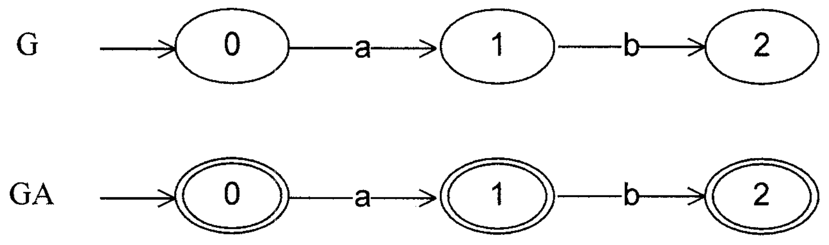

Figure 3. 5: Plant automaton and union of markings: Example 3.6

The plant with the union of marking is GA shown in Figure 3.5. The conventional supervisor $\gamma : L(G) \to \Sigma$ will be $\gamma(\varepsilon) = \{a\}, \gamma(a) = \{b\}, \gamma(ab) = \Phi$. This supervisor γ will be blocking when marking is X_{m_1}. □

Example 3.7: A plant G with event set $\Sigma = \{a,b,c,d\}$ has two sets of marking $X_{m_1} = \{0,1,2\}$ and $X_{m_2} = \{1,3\}$. Only event b is uncontrollable. Assuming all strings are legal, we design a supervisor which is nonblocking in both marking situations.

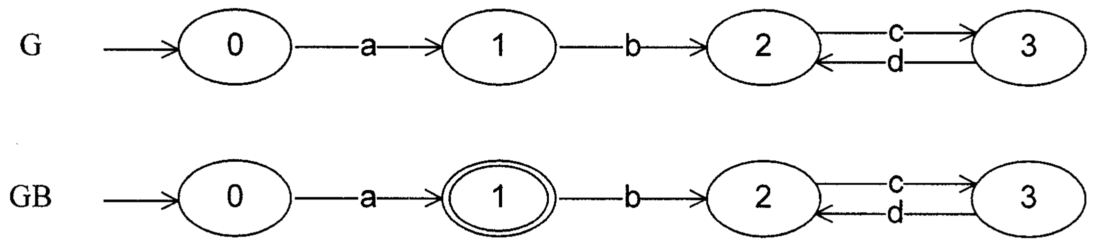

Figure 3. 6: Plant automata and intersection of markings: Example 3.7

The plant with the intersection of marking is GB shown in Figure 3.6 and the intersection of specifications is $\{\varepsilon\}$. There is no conventional supremal supervisor for GB. However, supervisor γ which enables all active events at all states will be a solution to the multiple marking problem. □

There is only one generated languages $L(G)$ in a system with multiple markings. However, the marked languages are multiple and denoted as $L_m^{(i)}(G)$ with $i \in I$. NSCP-MM can be treated as a special case of the general robust nonblocking supervisory control problem. The model uncertainty is due to the different markings in this situation.

We can convert a multiple marking supervision problem to an equivalent robust nonblocking supervisory control problem by defining $G_i = (X, \Sigma, \delta, x_0, X_{m_i})$ for all $i \in I$ which form the set of possible plant models. Obviously, we have $L(G) = L(G_i)$, $L_m^{(i)}(G) = L_m(G_i)$, $L(S/G) = L(S/G_i)$, and $L_m^{(i)}(S/G) = L_m(S/G_i)$ for all $i \in I$. The equivalent robust problem is formally described as follows.

Robust Nonblocking Supervisory Control Problem with Multiple Marking (RNSCP-MM)

Given a set of plants $G_i = (X, \Sigma, \delta, x_0, X_{mi})$ where $i \in I = \{1,2,...,n\}$ and specifications $E_i \subseteq L_m(G_i)$ for all $i \in I$, synthesize a supervisor all S such that $L_m(S/G_i) = L(S/G_i) \cap L_m(G_i) \subseteq E_i$ and $\overline{L_m(S/G_i)} = L(S/G_i)$ for all $i \in I$. □

We would like to explore the specialty of RNSCP-MM in comparison to the general RNSCP. Taking a plant with two possible marking as an example, the robust supervisor S solving the above problem has to satisfy that $L_m(S/G_1) \subseteq E_1$ and $\overline{L_m(S/G_1)} = L(S/G_1)$ as well as $L_m(S/G_2) \subseteq E_2$ and $\overline{L_m(S/G_2)} = L(S/G_2)$. We know $L(S/G_1) = L(S/G_2)$ since G_1 and G_2 are exactly the same plant with the same generated behavior. The difference is only due to the marking. Therefore, we know the equivalence $\overline{L_m(S/G_1)} = \overline{L_m(S/G_2)}$ has to be satisfied for S to be a solution to the problem.

Solving RNSCP-MM will automatically solve NSCP-MM. We only investigate solution to RNSCP-MM in the next subsection.

3.3.2 Solution to RNSCP-MM

We take direct robust supervision approach to solve the RNSCP-MM. The following theorem provides necessary and sufficient conditions for existence of a solution to RNSCP-MM.

Theorem 3.8: Given a language n-tuple $(K_1,...,K_n)$ where $K_i \neq \Phi$ and $K_i \subseteq L_m(G_i)$ for all $i \in I$ in RNSCP-MM, the necessary and sufficient conditions for the existence of an admissible nonblocking robust supervisor V such that $L_m(V/G_i) = K_i$ for all $i \in I$ are:

1) K_i is controllable with respect to G_i. (Controllability)

2) K_i is $L_m(G_i)$-closed. (Relative-closure)

3) $\overline{K_i} = \overline{K_j}$ for all $i, j \in I$. (Equiclosure) □

To prove Theorem 3.8, we only need to show that the equiclosure condition is equivalent to the consistency condition. In this special formulation, we have one more assumption which is $L(G_i) = L(G_j)$ for all $i, j \in I$. Theorem 3.9 provides the proof that the consistency reduces to equiclosure in the special case RNSCP-MM.

Theorem 3.9: Let $K_i \subseteq L_m(G_i)$ and $K_j \subseteq L_m(G_j)$ in RNSCP-MM. Then, $\overline{K_i} = \overline{K_j}$ iff K_i and K_j are consistent.

Proof: We need to show $\overline{K_i} = \overline{K_j} \Leftrightarrow [(L(G_i) - \overline{K_i}) \cap \overline{K_j} = \Phi \wedge (L(G_j) - \overline{K_j}) \cap \overline{K_i} = \Phi]$ for all $i, j \in I$. In RNSCP-MM, we have $L(G_i) = L(G_j)$ for all $i, j \in I$.

$(\Rightarrow)$: $(L(G_i) - \overline{K_i}) \cap \overline{K_j} = (L(G_i) - \overline{K_i}) \cap \overline{K_i} = \Phi$. Similarly, we can show $(L(G_j) - \overline{K_j}) \mathrm{I}\ \overline{K_i} = \Phi$.

$$(\Leftarrow): (L(G_i)-\overline{K_i})\cap\overline{K_j}=\Phi \Rightarrow L(G_j)\cap\overline{K_i}^{co}\cap\overline{K_j}=\Phi$$

$$\Rightarrow \overline{K_i}^{co}\cap\overline{K_j}=\Phi$$

$$\Rightarrow \overline{K_j}\subseteq\overline{K_i}$$

Similarly, we can show $\overline{K_i}\subseteq\overline{K_j}$. As a result, we conclude that $\overline{K_i}=\overline{K_j}$. □

Next we show that the operation $\Omega(K_i)=K_i-D\Sigma^*$ in the computation of $SupS(\cdot)$ reduces to $\hat{\Omega}(K_i)=K_i\cap H$ where $H=\bigcap_{j\in I}\overline{K_j}$. First, we prove the following lemmas.

Lemma 3.19: Assume that $L(G_i)=L(G_j)=L$ for all $i,j\in I$ and $K_i\subseteq L$ for all $i\in I$. Then $L\cap D\Sigma^*=D$ where $D=\bigcup_{k\in I}(L-\overline{K_k})$.

Proof: $(\supseteq)$: It is straightforward.

$(\subseteq)$: (By contradiction) Suppose $s\in L\cap D\Sigma^*$ but $s\notin D$. We have $s\notin L-\overline{K_i}$ for all $i\in I$ and thus $s\in\overline{K_i}$ for all $i\in I$. We have $\overline{K_i}\cap(L-\overline{K_i})\Sigma^*=\Phi$ for all $i\in I$ by Lemma 2.4 since $\overline{K_i}\cap(L-\overline{K_i})=\Phi$. Hence, we infer that $s\notin(L-\overline{K_i})\Sigma^*$ for all $i\in I$, which means $s\notin\bigcup_{k\in I}((L-\overline{K_k})\Sigma^*)$. Therefore, we have $s\notin(\bigcup_{k\in I}(L-\overline{K_k}))\Sigma^*=D\Sigma^*$, which is contradictory with $s\in D\Sigma^*$. Thus, we have $s\in D$.

As a result, we conclude that $L\cap D\Sigma^*=D$. □

Lemma 3.20: Assume that $L(G_i) = L(G_j) = L$ for all $i, j \in I$ and $K_i \subseteq L$ for all $i \in I$. Then, $\overline{K_i} \cap D = \overline{K_i} \cap D\Sigma^*$.

Proof: $\overline{K_i} \cap D\Sigma^* = \overline{K_i} \cap L \cap D\Sigma^*$

$= \overline{K_i} \cap D$ (by Lemma 3.19) □

Lemma 3.21: Assume that $L(G_i) = L(G_j) = L$ for all $i, j \in I$ and $K_i \subseteq L$ for all $i \in I$. Then, $K_i - \bigcup_{j\in I}(\Sigma^* - \overline{K_j}) = K_i - \bigcup_{j\in I}(L - \overline{K_j})$ for all $i \in I$.

Proof: ($\subseteq$): $\bigcup_{j\in I}(\Sigma^* - \overline{K_j}) \supseteq \bigcup_{j\in I}(L - \overline{K_j})$ since $\Sigma^* - \overline{K_j} \supseteq L - \overline{K_j}$ for all $j \in I$. Hence, we have $K_i - \bigcup_{j\in I}(\Sigma^* - \overline{K_j}) \subseteq K_i - \bigcup_{j\in I}(L - \overline{K_j})$.

($\supseteq$): $s \in K_i - \bigcup_{j\in I}(L - \overline{K_j}) \Rightarrow s \in K_i \wedge s \notin \bigcup_{j\in I}(L - \overline{K_j})$

$\Rightarrow s \in K_i \wedge s \in L \wedge (\forall j \in I : s \notin (L - \overline{K_j}))$

$\Rightarrow s \in K_i \wedge s \in L \wedge (\forall j \in I : s \in \overline{K_j}))$

$\Rightarrow s \in K_i \wedge (\forall j \in I : s \notin \Sigma^* - \overline{K_j})$

$\Rightarrow s \in K_i \wedge s \notin \bigcup_{j\in I}(\Sigma^* - \overline{K_j})$

$\Rightarrow s \in K_i - (\bigcup_{j\in I}(\Sigma^* - \overline{K_j}))$.

Hence, we infer that $K_i - \bigcup_{j\in I}(\Sigma^* - \overline{K_j}) \supseteq K_i - \bigcup_{j\in I}(L - \overline{K_j})$. As a result, we conclude that

$K_i - \bigcup_{j\in I}(\Sigma^* - \overline{K_j}) = K_i - \bigcup_{j\in I}(L - \overline{K_j})$. □

We are ready to prove the equivalence of operator Ω and operator $\hat{\Omega}$ in RNSCP-MM. The above lemmas are used in the proof of Theorem 3.10.

Theorem 3.10: Assume that $L(G_i) = L(G_j) = L$ for all $i, j \in I$ and $K_i \subseteq L$ for all $i \in I$. Then, $\hat{\Omega}(K_i) = \Omega(K_i)$.

Proof: $\hat{\Omega}(K_i) = K_i \cap H$

$$
\begin{aligned}
&= K_i \cap (\bigcap_{j\in I} \overline{K_j}) \\
&= K_i - (\bigcap_{j\in I} \overline{K_j})^{co} \\
&= K_i - (\bigcup_{j\in I} \overline{K_j}^{co}) \\
&= K_i - (\bigcup_{j\in I} (\Sigma^* - \overline{K_j})) \\
&= K_i - (\bigcup_{j\in I} (L - \overline{K_j})) && \text{(by Lemma 3.21)} \\
&= K_i - D \\
&= K_i - \overline{K_i} \cap D \\
&= K_i - \overline{K_i} \cap D\Sigma^* && \text{(by Lemma 3.20)} \\
&= K_i - D\Sigma^* \\
&= \Omega(K_i). && \square
\end{aligned}
$$

Denote the collections of all equiclosed sublanguage n-tuples of $(E_1,...,E_n)$ as $E((E_1,...,E_n))$. $(\Phi,...,\Phi) \subseteq (E_1,...,E_n)$ and $E((E_1,...,E_n))$ is closed with respect to union operation. Therefore, the supremal equiclosed sublanguage denoted as $SupE((E_1,...,E_n))$ is well-defined.

The supremal consistent sublanguage n-tuple $SupS((E_1,...,E_n))$ reduces to the supremal equiclosed sublanguage n-tuple $SupE((E_1,...,E_n))$ in RNSCP-MM. In the light of the above observation, we assert that $SupE((E_1,...,E_n))$ is equal to the largest fixed-point of the operator $\hat{\Omega}$.

The supremal equiclosed sublanguage n-tuple $SupE((E_1,...,E_n))$ can be calculated using the following algorithm if it terminates. The algorithm can be regarded as a reduced version of Algorithm 3.1.

Algorithm 3.3:

1) Initial language n-tuple $(K_1^0,...,K_n^0)=(E_1,...,E_n)$.

2) Update $(K_1^{j+1},...,K_n^{j+1})=\hat{\Omega}((K_1^j,...,K_n^j)), j \geq 0$.

3) If $K_i^{j+1} \neq K_i^j$ for some $1 \leq i \leq n$, go back to step 2. □

We next show that there exists a supremal solution to RNSCP-MM. Any language n-tuple including $(\Phi,...,\Phi)$ that meets the three conditions in Theorem 3.8 will be a solution to RNSCP-MM. In addition, the arbitrary union of solutions to RNSCP-MM remains a solution. Therefore, the supremal solution to RNSCP-MM exists. It is the supremal relative-closed, controllable, and equiclosed sublanguage n-tuple of the specifications. This supremal element is denoted by $SupRCE((E_1,...,E_n))$.

The Supremal Solution to RNSCP-MM: The solution to RNSCP-MM is to calculate the supremal sublanguage n-tuple $SupRCE((E_1,...,E_n))$. An optimal modular or monolithic supervisor can then be realized to solve the problem. □

$SupRCE((E_1,...,E_n))$ is the largest fixed point of operator $\underline{\hat{\Omega}}$ defined as $\underline{\hat{\Omega}}((K_1,...,K_n)) = SupE(SupC(SupR((K_1,...,K_n))))$. In addition, we infer that the supermal equiclosed sublanguage of a relative-closed language n-tuple remains relative-closed.

The algorithm below calculates the supremal solution to the above problem RNSCP-MM if it terminates. It first calculates the supremal solution as the reduced form of Algorithm 3.2A and then synthesizes supervisor monolithically or modularly.

Algorithm 3.4:

1) Initialization: $(K_1^0,...,K_n^0) = SupR((E_1,...,E_n))$.

2) Update $(K_1^{j+1},...,K_n^{j+1}) = SupE(SupC((K_1^j,...,K_n^j))), j \geq 0$.

3) If $K_i^{j+1} \neq K_i^j$ for some $1 \leq i \leq n$, then go to step 2.

4) Synthesize the monolithic supervisor $\hat{V}(s) = (\underset{i \in I}{\mathrm{Y}} \overline{K_i}) / \dot{s} \cap \Sigma$ or the modular supervisor $\widetilde{V}(s) = \underset{i \in I}{\mathrm{Y}} V_i(s)$ where $V_i(s) = \overline{K_i} / s \cap \Sigma$. □

Assume the above algorithm terminates with $(K_1^m,...,K_n^m)$ at the m^{th} iteration. Following from the general RNSCP and the equivalence of $SupE((E_1,...,E_n))$ and

$SupS((E_1,...,E_n))$, we have $(K_1^m,...,K_n^m) = SupRCE((E_1,...,E_n))$. The resulting nonblocking supervisors $\hat{V}$ and $\tilde{V}$ will solve RNSCP-MM with $L_m(\hat{V}/G_i) = K_i$ and $L_m(\tilde{V}/G_i) = K_i$.

Example 3.8: A plant with two different markings is shown in Figure 3.7. Assume that all events are controllable. We synthesize a supervisor which restricts each plant model into its specification and ensures the closed-loop behavior of each model is nonblocking.

The generated behaviors of G_1 and G_2 are exactly the same while the marked languages of G_1 and G_2 are different. Therefore, it can be solved using Algorithm 3.4. The plant languages are $L(G_1) = L(G_2) = \{\varepsilon, a, ab, abc, abd\}$, $L_m(G_1) = \{\varepsilon, a, abc\}$, and $L_m(G_2) = \{\varepsilon, a, abd\}$. Assume the specifications are $E_1 = L_m(G_1) = \{\varepsilon, a, abc\}$ and $E_2 = L_m(G_2) = \{\varepsilon, a, abd\}$.

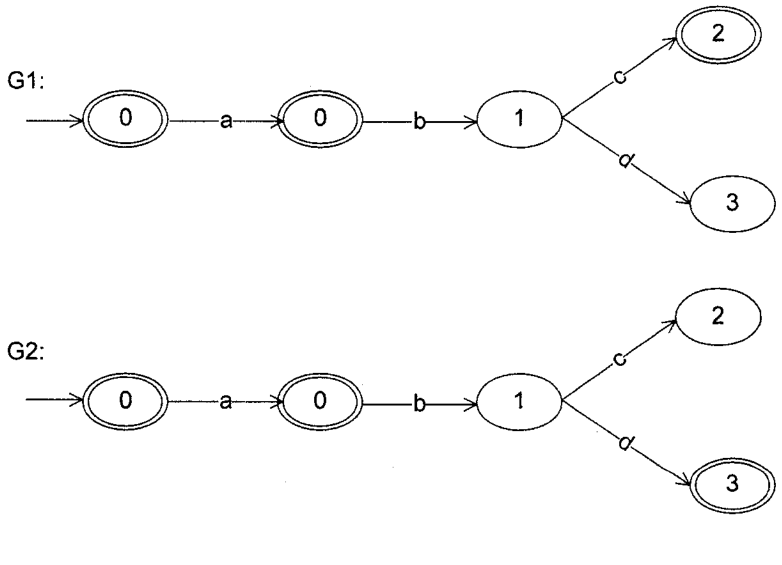

Figure 3. 7: Automata for Example 3.8

The procedure that solves the problem using Algorithm 3.4 is illustrated below:

Initialization: Calculate $(K_1^0, K_2^0) = SupR((E_1, E_2))$. The result is $K_1^0 = \{\varepsilon, a, abc\}$ and $K_2^0 = \{\varepsilon, a, abd\}$.

Iteration 1: Compute $(K_1^1, K_2^1) = SupE(SupC((K_1^0, K_2^0))$. The result is $K_1^1 = \{\varepsilon, a\}$ and $K_2^1 = \{\varepsilon, a\}$. Since both K_1^1 and K_2^1 are different from K_1^0 and K_2^0, the next iteration is taken.

Iteration 2: Compute $(K_1^2, K_2^2) = SupE(SupC((K_1^1, K_2^2))$. The result is $K_1^2 = \{\varepsilon, a\}$ and $K_2^2 = \{\varepsilon, a\}$. Since neither K_1^2 nor K_2^2 is different from K_1^1 and K_2^1 respectively, we stop here.

Solution: The solution will be $L(\hat{V} / G_1) = \overline{K_1^2} = \{\varepsilon, a\}$ and $L(\hat{V} / G_2) = \overline{K_2^2} = \{\varepsilon, a\}$. □

Supervisory control problem of a discrete event system with multiple marking can be solved as a special case of robust nonblocking supervisory control problem. There exists a unique property $L(G_i) = L(G_j)$ for all $i, j \in I$, which can be used to make the solution easier. The consistency condition in the general robust nonblocking supervisory control problem is replaced by the equiclosure condition in robust nonblocking supervisory control problem with multiple markings.

Appendix 3.1: TTCT Procedures

Consider a system with a set of n possible models $(G_1,...G_n)$ and corresponding specifications $(E_1,...E_n)$. Assume $(E_1,...E_n)$ is marked by trim automata $(H_1,...H_n)$. The procedure of computing $\Omega((E_1,...,E_n))$ in TTCT is given in the following.

Procedure 3.1: $Omega((H_1,...,H_n))$

1) Find out strings disabled in G_1:

Obtain HM_1 by marking all states of H_1 . Then, compute $HMCOM_1 = Complement(HM_1)$. Obtain GM_1 by marking all states of G_1 . The automaton $D_1 = Meet(HMCOM_1, GM_1)$ marks the language $L(G_1) - \overline{E_1}$. Obtain DST_1 by deleting all active events at the marked states and then adding selfloop of Σ at the marked states of D_1. The resulting automaton DST_1 marks the language $(L(G_1) - \overline{E_1})\Sigma^*$.

2) Find the disabled strings DST_2 to DST_n in G_2 to G_n similarly:

3) Derive the overall disabled strings.

$DCOM_1 = Complement(DST_1)$;...; $DCOM_n = Complement(DST_n)$.

$DA_2 = Meet(DCOM_1, DCOM_2)$; $DA_3 = Meet(DA_2, DCOM_3)$; ...;

$DA_n = Meet(DA_{n-1}, DCOM_n)$.

$D = Complement(DA_n)$. The automaton D marks $\bigcup_{i\in I}[(L(G_i)-\overline{E_i})\Sigma^*]$ which is the same as $[\bigcup_{i\in I}(L(G_i)-\overline{E_i})]\Sigma^*$.

4) Remove the disabled strings from $(E_1,...E_n)$.

$DCOM = Complement(D)$. Obviously, $DCOM$ and DA_n mark the same language.

Finally, we have $Omega((H_1,...,H_n)) = (Meet(H_1, DCOM),..., Meet(H_n, DCOM))$. □

We next derive the procedure *SupSis* to compute $SupS((E_1,...,E_n))$ in TTCT. We have proved that the supremal consistent sublanguage is the largest fixed point of operator Ω, which can be calculated by applying operator Ω on the specification n-tuple repeatedly until a fixed point is reached. Hence, testing equivalence of two languages is required. We first develop Equivalence procedure *Eq* in TTCT before procedure *SupSis* in TTCT is given. The following procedure *Eq* checks if two languages E_1 and E_2 marked by R_1 and R_2 are equivalent in TTCT.

Procedure 3.2: $Eq((R_1, R_2))$

1) First, check if $E_1 \subseteq E_2$.

$RCOM_2 = Complement(R_2)$. $RA = Meet(R_1, RCOM_2)$.

2) If there is some marked state in RA, then return False.

3) Then, Check if $E_2 \subseteq E_1$.

$RCOM_1 = Complement(R_1)$. $RB = Meet(R_2, RCOM_1)$

4) If there is some marked state in RB, then return False.

5) Return True. □

We are now ready to develop the procedure of calculating $SupS((E_1,...,E_n))$ in TTCT. Although operator Ω in the computation of the supremal consistent sublanguages can be computed using the procedure *Omega* in TTCT, there is no closed-form formula for calculating the supremal consistent sublanguages. An iterative procedure has to be taken until consistency is satisfied. Assume $(E_1,...,E_n)$ is marked by trim generators $(H_1,...,H_n)$. The procedure of calculating the supremal consistent sublanguages in TTCT is illustrated below.

Procedure 3.3: $SupSis((H_1,...,H_n))$

1) Initialization: $(E_1^0,...,E_n^0) = (E_1,...,E_n)$.

2) Compute $(E_1^{j+1},...,E_n^{j+1}) = \Omega((E_1^j,...,E_n^j))$ where $j \geq 0$ using Procedure 3.1 *Omega*.

3) Check if $E_i^{j+1} = E_i^j$ using procedure *Eq* for all $1 \leq i \leq n$. If $E_i^{j+1} \neq E_i^j$ for some $1 \leq i \leq n$, then $j = j+1$ and go to step 2. □

We finally develop TTCT procedure for implementing Algorithm 3.2A. Operations *SupR*, *SupC*, and *SupS* are required in Algorithm 3.2A. In addition, testing equivalence of two languages is also required. All of the operations shall be implemented by existing commands or their combination in TTCT in order to obtain the SupRCS using Algorithm 3.2A in TTCT.

The existing command *SupCon* in TTCT calculates the supremal controllable sublanguage. The supremal consistent sublanguages can be computed using Procedure 3.3 *SupSis* in TTCT. Testing equivalence of two languages can be fulfilled using Procedure 3.2 *Eq* in TTCT.

There is no command to compute the supremal relative-closed sublanguage in TTCT. However, a closed-form expression exists for the calculation of supremal relative-closed sublanguages. All operations in the closed-form expression can be computed using the existing commands in TTCT.

We present a procedure *SupRa* for calculating the supremal relative-closed sublanguage in TTCT before we propose the procedure *SupRSCa* that implements Algorithm 3.2A using TTCT. Assuming a language K is marked by automaton H, the following procedure computes $SupR(K)$ in TTCT.

Procedure 3.4: $SupRa(H)$

1) Compute $HCOM = Complement(H)$. The automaton $R = Meet(HCOM, G)$ marks the language $L_m(G) - K$.

2) Obtain RST by deleting all active events at the marked states and then adding self loop of Σ at the marked states of R. The resulting automaton RST marks the language $(L_m(G) - K)\Sigma^*$.

3) Obtain $RSTCOM = Complement(RST)$ and $RA = Meet(RSTCOM, H)$.

The automaton RA will mark the language $SupR(K)$. □

Up to now, all procedures in TTCT required for calculating $SupRCS$ are developed. We can use them to implement Algorithm 3.2A using TTCT. The procedure $SupRCSa$ that computes $SupRCS$ using TTCT is illustrated below.

Procedure 3.5: $SupRCSa((H_1,...,H_n))$

1) Compute $(K_1^0,...,K_n^0) = SupR((E_1,...,E_n))$ using Procedure 3.4 $SupRa$.

2) Compute $(K_1^{j+1},...,K_n^{j+1}) = SupS(SupC((K_1^j,...,K_n^j))), j \geq 0$ where $j \geq 0$ using the existing command procedure $SupCon$ and newly developed Procedure 3.3 $SupSis$.

3) Check if $K_i^{j+1} = K_i^j$ using Procedure 3.2 Eq for all $1 \leq i \leq n$. If $K_i^{j+1} \neq K_i^j$ for some $1 \leq i \leq n$, then go to step 2. □

Chapter 4

Online Robust Nonblocking Supervisory Control

Robust nonblocking supervisory control problem was solved offline using direct approach in Chapter 3. However, it is difficult to implement when the plant model is very complex.

In this chapter, offline direct robust supervisory control is implemented online to avoid storing supervisor explicitly. We first develop an algorithm to solve RNSCP online and then prove its validity. The running problem in Example 2.1 is solved online using the algorithm.

4.1 Variable Lookahead Policy (VLP)

A variable lookahead policy is introduced to solve RNSCP. We review RNSCP before we derive an online algorithm with direct approach to solve it.

RNSCP—Robust Nonblocking Supervisory Control Problem:

Given a set of plants G_i with $i \in I = \{1,2,...,n\}$, the marked language $L_m(G_i)$ of plant G_i is required to be restricted into its legal behavior $E_i \subseteq L_m(G_i)$ for all $i \in I$. Synthesize an optimal admissible robust nonblocking supervisor, which solves the problem. □

Offline solution with direct approach was investigated in the previous chapter. The direct approach to RNSCP can also be implemented online without exploring the entire plant models. We only expand the part of the models that is sufficient for valid control decision making. We assume without loss of generality that E_i is $L_m(G_i)$-closed for all $i \in I$.

Conventional limited lookahead policy [7] expands a plant as a tree generator to a fixed length, takes conservative or optimistic attitude to the pending traces, and then makes control decision after computing the supremal element. Run-time error may happen due to uncertainty of the unexpanded system behavior.

Here, we apply certain expansion stop rules to the tree generators instead of a fixed expansion length. Each string stops only when the future behavior of the system does not affect the valid control decision for the current string. Hence, no attitude is required and no run-time error happens. As a result, the correct decision for the current string will be obtained if the expansion for the current string stops. In addition, the overall online supervision will be valid if the expansion stops for all strings executed in the closed-loop system.

We first investigate how expansion shall be done for online robust supervision. Obviously, expansion of all possible models shall be conducted simultaneously since the control decision is made for the same string in all possible models. In addition, all events following the current string shall be expanded since we want to decide which active events of the current string can be enabled.

Next we would find the length of expansion that can ensure validity. In other words, we want to explore under what conditions the expansion of a branch of an expansion tree can be terminated without uncertainty for valid online control decision. Generally speaking, if there is no uncertainty about control decision for the current string at all boundary strings, then the expansion will ensure the online control decision for the current string is the same as the offline optimal supervision.

In order to offer formal description of the expansion stop rules, we define illegal language, legal marked language, legal marked controllable language, legal marked uncontrollable language, and legal transient language. They form a partition of the entire possible strings of a system. The definitions of these languages in robust problems are different with those in conventional (non-robust) problems. For example, a string is illegal in the specification of one model may be legal in the specification of another model. A string marked in one possible model may not be marked in another possible plant model. A string leading to a marked state with only controllable active events may lead to a marked state with uncontrollable continuity in another model. The definitions of illegal language, legal marked language, legal marked controllable language, legal marked uncontrollable language, and legal transient language are given below.

If a string is illegal in one model, it shall be removed from all possible models. Hence, it is unnecessary to expand its following strings. Such a string is called an illegal string. Note that an illegal string is not necessarily illegal in all models.

Definition 4.1: The language denoted by K_{fc} consists of strings that are illegal in some plant model: $K_{fc} := \{s \in \bigcup_{j\in I} L(G_j) \mid \exists i \in I : s \in (L(G_i) - \overline{E_i})\}$. □

The **illegal language** $K_{illegal}$ will be $K_{fc}\Sigma^* \cap (\bigcup_{j\in I} L(G_j))$. The **legal language** denoted as K_{legal} will be $(\bigcup_{j\in I} L(G_j)) - K_{illegal}$. All active uncontrollable events of a legal string shall be expanded since we do not know if they will lead to illegal strings or result in blocking.

If a legal string is marked in all plant models, none of its following controllable events needs to be expanded. This will not affect the control decision for the current string. All legal strings that are marked in all plant models form the legal marked language. The following example illustrates that no active controllable events of a legal marked string is required to be expanded for correct online decision.

Example 4.1: A system with two possible plant models is shown in Figure 4.1. The marked states are denoted by a double circle. Assume all events are controllable and all strings are legal. We expand the system for the initial state such that correct decision can be made.

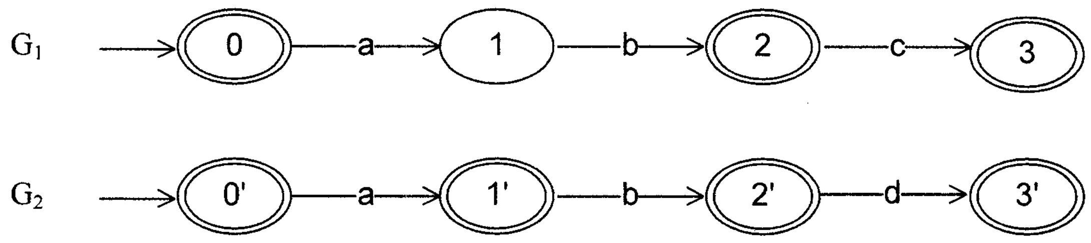

Figure 4. 1: Plant automata for Example 4.1

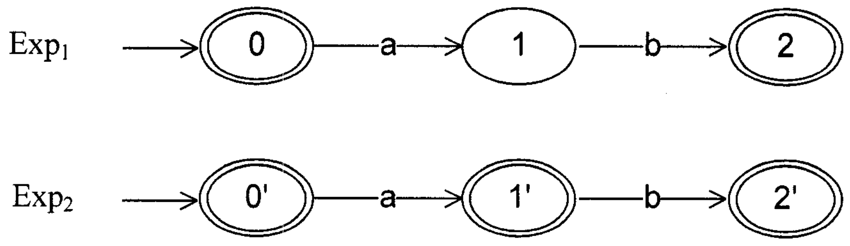

Figure 4. 2: Expansion for the initial state: Example 4.1

The controllable events c and d are not expanded at string ab that is marked in both models. The expansion for the initial state is shown in Figure 4.2. The supremal solution for the expansion windows will be $(\{\varepsilon, ab\}, \{\varepsilon, a, b\})$. Hence, event a will be enabled by the online supervisor at the initial state. This agrees with the offline supervision. □

Definition 4.2: The **legal marked language** denoted as K_m consists of legal strings that are marked in all plant models which contain the strings:

$$K_m := \{s \in \bigcup_{j \in I} L(G_i) \mid \forall i \in I : s \in L(G_i) \Rightarrow s \in \overline{E_i} \cap L_m(G_i) = E_i\}. \qquad \square$$

The legal marked language can be further divided into legal marked controllable language and legal marked uncontrollable language.

Definition 4.3: The **legal marked controllable language** K_{mc} consists of strings that are in K_m and followed by no event or only controllable events in all plant models that contain the strings: $K_{mc} := \{s \in K_m \mid \forall i \in I : s \in L(G_i) \Rightarrow \Sigma_{L(Gi)}(s) \subseteq \Sigma_c\}$. □

Definition 4.4: The **legal marked uncontrollable language** K_{mu} consists of strings that are in K_m and followed by uncontrollable events in some plant model: $K_{mu} := \{s \in K_m \mid \exists i \in I \wedge s \in L(G_i) : \Sigma_{G_i}(s) \cap \Sigma_u \neq \Phi\}$. □

Obviously, K_{mc} and K_{mu} form a partition of K_m and thus $K_m = K_{mc} \dot{\cup} K_{mu}$.

Expansion should not stop at a legal string that is unmarked in all models since we do not know if there is blocking in the future. Hence, its following controllable events shall also be expanded in addition to its uncontrollable events. If a legal string is marked in some models but not marked at least in one model, its following controllable events shall also be expanded. All legal strings that are not marked in some model form the legal transient language.

Definition 4.5: The **legal transient language** K_{tran} consists of legal strings which are not in legal marked language K_m: $K_{tran} := \{s \in \bigcup_{j \in I} L(G_j) \mid s \notin K_{illegal} \cup K_m\}$. □

The following example uses the same problem in Example 4.1 to illustrate that the controllable events of legal transient strings have to be expanded.

Example 4.2: A system with two possible plant models is shown in Figure 4.3. The marked state is denoted by a double circle. All events are controllable and all strings are legal. Expand the system for the empty string ε such that correct decision can be made.

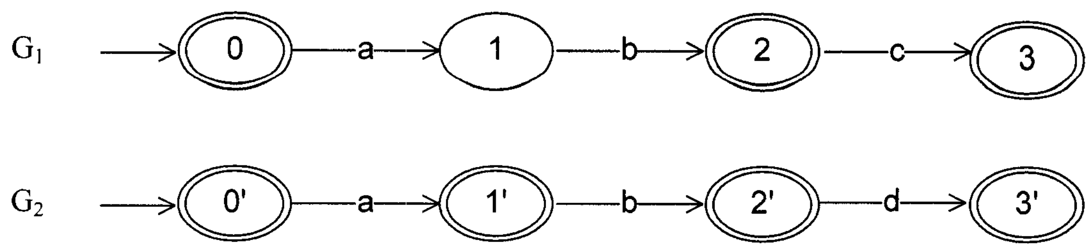

Figure 4. 3: Plant automata for Example 4.2

We show that controllable events are required to expand at a legal transient string for correct online decision.

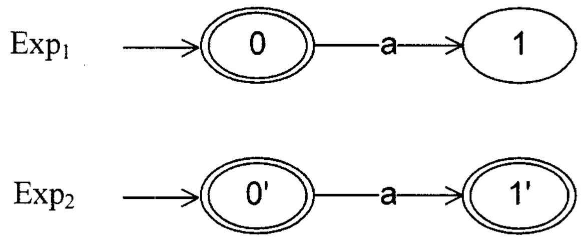

Figure 4. 4: Expansion for the initial state: Example 4.2

At the current string $s = \varepsilon$, its active event a is expanded in G_1 and G_2 simultaneously. String a is marked in G_2 but not in G_1. If we stop expansion at string a, the expansion windows will be the trees shown in Figure 4.4. The robust solution for the expansion window is $(\{\varepsilon\},\{\varepsilon\})$. Hence, the online supervisor will disable event a at string ε. However, event a is enabled by the offline robust supervisor. Therefore, expansion shall not terminate at the string a which is marked in one model but another. The active event b of the string a is required to expand. □

The legal language K_{legal} is composed of legal marked language K_m and legal transient language K_{tran}. The overall system behavior $\bigcup_{j\in I} L(G_j)$ is partitioned as:

$\bigcup_{j \in I} L(G_j) = K_{illegal} \dot{\cup} K_{mc} \dot{\cup} K_{mu} \dot{\cup} K_{tran}$. The following example uses the same problem in Example 2.1 to illustrate this.

Example 4.3: A system model could be either G_1 or G_2 shown in Figure 4.5. Assume that $\Sigma = \{\alpha, \beta, \gamma, \lambda, \mu, u, v, w\}$ with $\Sigma_c = \{\alpha, \beta, \gamma, \lambda, \mu\}$ and $\Sigma_u = \{u, v, w\}$. Given certain specifications, we find the $K_{illegal}$, K_{mc}, K_{mu}, and K_{tran} for the system and verify that the overall behavior can be partitioned as the four languages.

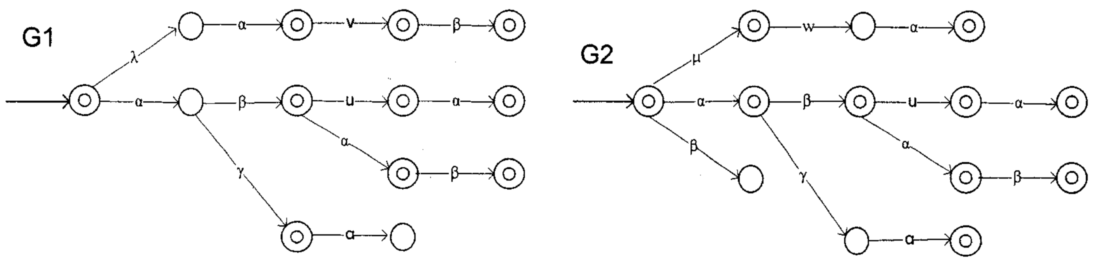

Figure 4. 5: Plant automata for Example 4.3

The overall system behavior $L(G_1) \cup L(G_2)$ is $\{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma, \alpha\gamma\alpha, \beta, \lambda, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta, \mu, \mu w, \mu w\alpha\}$. Assume that the specifications are $E_1 = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$ and $E_2 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu\}$ respectively.

The illegal language $K_{illegal}$ is $\{\alpha\gamma\alpha, \beta, \alpha\beta\alpha, \alpha\beta\alpha\beta, \mu w, \mu w\alpha\}$. The legal marked controllable language K_{mc} is $\{\varepsilon, \alpha\beta u, \alpha\beta u\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$. The legal marked uncontrollable language K_{mu} is $\{\alpha\beta, \lambda\alpha, \mu\}$. The legal transient language K_{tran} is

$\{\alpha, \alpha\gamma, \lambda\}$. We can verify that $K_{illegal}$, K_{mc}, K_{mu}, and K_{tran} are disjoint and form a partition of the overall system behavior $L(G_1) \cup L(G_2)$. □

We now can formally present the expansion rules for our variable lookahead policy. All plant models are expanded simultaneously as a tree generator. Note that a node in a tree corresponds to a sole string and thus we do not differentiate a node and its corresponding string thereafter. The expansion rules are as follows:

Expansion Rules:

1) All plant models are expanded simultaneously as tree generators.

2) All active events following the current string shall be expanded.

3) Stop expanding strings in K_{fc}.

4) Stop expanding strings in K_{mc}.

5) Stop expanding the controllable events of strings in K_{mu}.

6) Expand uncontrollable events of strings in K_{mu}.

7) Expand all events of strings in K_{tran}. □

We call expansion according to the above rules variable lookahead policy (VLP) since the expansion does not stop at a fixed length. The expansion can be done recursively with either breadth-first or depth-first approach. The following procedure Expansion(*s*) takes recursive depth-first expansion approach and returns the expansion trees Exp_i for the current string s if it terminates.

Expansion(*s*):

```
1) Initialization: $Exp_i := \{\varepsilon\}$ for all $i \in I$.
2) For all $\alpha \in \Sigma \cap (\bigcup_{i \in I} L(G_i)/s)$:
    2.1) for all $i \in I$,
            if $s\alpha \in L(G_i)$, then $Exp_i = Exp_i \cup \{\alpha\}$.
    2.2) $Expan(\alpha)$.
3) Return.    □
```

Given a set of possible plant model G_i and corresponding E_i, the expansion procedure of Exapnsion(s) for the current string s is as follows. One of the active events $\{\alpha_1,...,\alpha_m\}$ at the current string s is expanded by calling a recursive procedure $Expan(t)$. For instance, α_1 is first expanded. Once $Expan(\alpha_1)$ returns, another active event of the current string s, say α_2, is expanded the same way by calling $Expan(\alpha_2)$. The expansion is executed until all active events of the current string s are expanded.

Expan(*t*)

```
1) Case $st \in K_{tran}$:
```

```
For all $\beta \in \Sigma \wedge st\beta \in \bigcup_{i \in I} L(G_i)$,
    For all $i \in I$,
            if $st\beta \in L(G_i)$, $Exp_i := Exp_i \cup \{t\beta\}$.
    $Expan(t\beta)$;
Return.

2) Case $st \in K_m$:
For all $\beta \in \Sigma_u \wedge st\beta \in \bigcup_{i \in I} L(G_i)$,
    For all $i \in I$,
            if $st\beta \in L(G_i)$, $Exp_i := Exp_i \cup \{t\beta\}$.
    $Expan(t\beta)$;
Return.

3) Case else ($st \in K_{illegal}$):
Return.            □
```

$Expan(\alpha_1)$ for the active event α_1 of the current string s is executed recursively with a depth-first approach. If string $s\alpha_1$ is a legal transient string, all active events $\{\beta_1,...,\beta_n\}$ at string $s\alpha_1$ are expanded by calling $Expan(\alpha_1\beta_1)$ to $Expan(\alpha_1\beta_n)$. If string $s\alpha_1$ is a legal marked string with active controllable events $\{c_1,...,c_m\}$ and uncontrollable events $\{u_1,...,u_n\}$, then only $Expan(\alpha_1 u_1)$ to $Expan(\alpha_1 u_n)$ are called recursively for all uncontrollable events. Expansion of string α_1 returns without calling $Expan(\alpha_1 \beta)$ for any active event β if $s\alpha_1$ is illegal or legal marked controllable.

We call the resulting expansion of G_i (as a tree generator) Exp_i, once expansion for the current string s terminates. The current state s is treated as the initial state of the expansion Exp_i.

Definition 4.6: The set of generated strings of Exp_i is denoted as $L(Exp_i)$. The set of marked strings $L_m(Exp_i)$ consists of strings in $L(Exp_i)$ which belong to $L_m(G_i)/s$. Namely, $L_m(Exp_i) = L_m(G_i)/s \cap L(Exp_i)$. □

We take the same problem in Example 2.1 to illustrate the expansion for the empty string according to the above expansion rules.

Example 4.4: A system model could be either G_1 or G_2 shown in Figure 4.6. Assume that $\Sigma = \{\alpha,\beta,\gamma,\lambda,\mu,u,v,w\}$ with $\Sigma_c = \{\alpha,\beta,\gamma,\lambda,\mu\}$ and $\Sigma_u = \{u,v,w\}$. Assume that the specifications are $E_1 = \{\varepsilon,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\gamma,\lambda\alpha,\lambda\alpha v,\lambda\alpha v\beta\}$ and $E_2 = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma\alpha,\mu\}$ respectively. We obtain the expansion for the empty string ε using the expansion rules and depth-first approach.

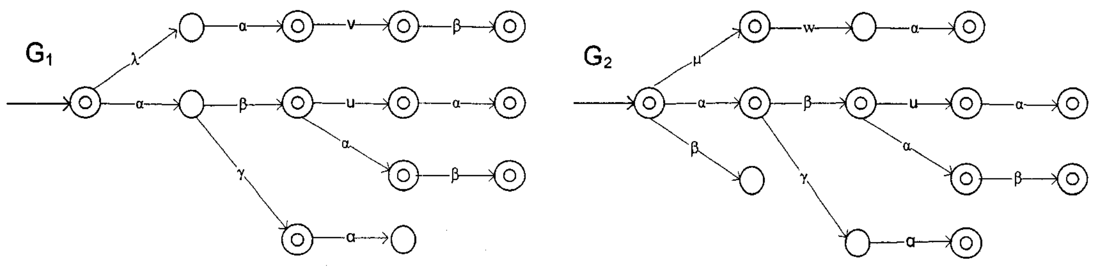

Figure 4. 6: Plant automata for Example 4.4

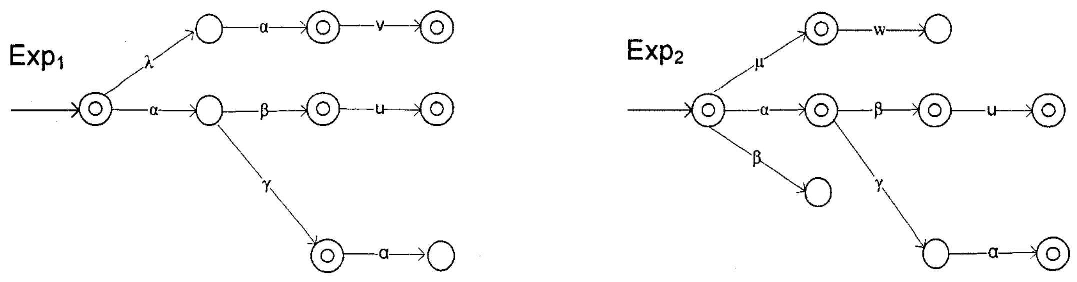

Figure 4. 7: Expansion for the initial state: Example 4.4

Initially, $L(Exp_1) = \{\varepsilon\}$ and $L(Exp_2) = \{\varepsilon\}$. All active events $\{\alpha, \beta, \lambda, \mu\}$ of the current string ε shall be expanded. For instance, α is first expanded and thus $L(Exp_1) = \{\varepsilon, \alpha\}$ and $L(Exp_2) = \{\varepsilon, \alpha\}$. The active events $\{\beta, \gamma\}$ of α shall be further expanded since string α is a transient string. Assuming that β is first expanded, we have $L(Exp_1) = \{\varepsilon, \alpha, \alpha\beta\}$ and $L(Exp_2) = \{\varepsilon, \alpha, \alpha\beta\}$. The active uncontrollable event u of string $\alpha\beta$ shall be further expanded since string $\alpha\beta$ is a legal marked uncontrollable string. At this time, we have $L(Exp_1) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u\}$ and $L(Exp_2) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u\}$. The expansion for this branch of the tree generators stops and returns at the string $\alpha\beta u$ since $\alpha\beta u$ is a legal marked controllable string. Next, the second event γ following string $s = \alpha$ is expanded. We omit the rest of expansion procedure for brevity. The resulting expansion is shown in Figure 4.7. □

The specification with respect to the expansion window Exp_i is taken as $E_i / s \cap L_m(Exp_i)$ and will be relative-closed with respect to $L_m(Exp_i)$ as shown in the following lemma.

Lemma 4.1: $E_i / s \cap L_m(Exp_i)$ is $L_m(Exp_i)$-closed.

Proof: $\overline{E_i / s \cap L_m(Exp_i)} \cap L_m(Exp_i) \subseteq \overline{E_i / s} \cap L_m(Exp_i)$

$$= \overline{E_i} / s \cap L_m(Exp_i) \qquad \text{(by Lemma 2.1)}$$

$$= \overline{E_i} / s \cap L_m(G_i) / s \cap L_m(Exp_i)$$

$$= [\overline{E_i} \cap L_m(G_i)] / s \cap L_m(Exp_i) \qquad \text{(by Lemma 2.2)}$$

$$= E_i / s \cap L_m(Exp_i). \qquad \text{(} E_i \text{ is } L_m(G_i)\text{-closed)}$$

On the other hand, $E_i / s \cap L_m(Exp_i) \subseteq \overline{E_i / s \cap L_m(Exp_i)} \cap L_m(Exp_i)$. As a result, we have $\overline{E_i / s \cap L_m(Exp_i)} \cap L_m(Exp_i) = E_i / s \cap L_m(Exp_i)$. □

All plant models are expanded simultaneously. Any string in $L(G_j)/s$ will be expanded in Exp_j if it is expanded in Exp_i. Namely, $L(Exp_i) \cap L(G_j)/s \subseteq L(Exp_j)$.

Lemma 4.2: $L(Exp_i) \cap (L(G_j)/s - L(Exp_j)) = \Phi$

Proof: $L(Exp_i) \cap (L(G_j)/s - L(Exp_j)) = L(Exp_i) \cap L(G_j)/s \cap L(Exp_j)^{co}$

$\subseteq L(Exp_j) \cap L(Exp_j))^{co}$ (All models are expanded simultaneously)

$= \Phi$.

Hence, we have $L(Exp_i) \cap (L(G_j)/s - L(Exp_j)) = \Phi$. □

Since all active events of the current string are expanded (Rule 2 of Expansion), strings in $s\Sigma \cap L(G_i)$ will be expanded in Exp_i.

Lemma 4.3: $L(G_i)/s \cap \Sigma = L(Exp_i) \cap \Sigma$

Proof: $t \in L(G_i)/s \cap \Sigma \Rightarrow st \in s(L(G_i)/s \cap \Sigma)$

$\Rightarrow st \in s(L(G_i)/s) \cap s\Sigma$ (by Lemma 4.A.7)

$\Rightarrow st \in L(G_i) \cap s\Sigma$ (by Lemma 4.A.5)

$\Rightarrow t \in L(Exp_i)$ (All active events of s are expanded)

$\Rightarrow t \in L(Exp_i) \cap \Sigma$.

Hence, we have $L(G_i)/s \cap \Sigma \subseteq L(Exp_i) \cap \Sigma$. On the other hand, it is immediate that $L(G_i)/s \cap \Sigma \supseteq L(Exp_i) \cap \Sigma$. As a result, we have $L(G_i)/s \cap \Sigma = L(Exp_i) \cap \Sigma$. □

Definition 4.7: A string t in the expansion window for the current string s is called a **boundary string** if there is no active event in all plant models or some of its active events are not expanded in some plant model:

$\{t \in \bigcup_{j \in I} L(Exp_j) \mid (\exists i \in I, t \in L(Exp_i) \wedge t\Sigma \cap (L(G_i)/s - L(Exp_i) \neq \Phi)\}$. □

The boundary strings are either illegal strings, legal marked strings. No active event of illegal strings and marked controllable strings is expanded. No active controllable event of marked uncontrollable strings is expanded. All active uncontrollable events of legal strings are expanded. In other words, only active uncontrollable events of illegal strings are not expanded. The following example illustrates the boundary strings in expansions for the current string with the same problem in Example 4.4.

Example 4.5: A system model could be either G_1 or G_2 shown in Figure 4.8. Assume that $\Sigma = \{\alpha, \beta, \gamma, \lambda, \mu, u, v, w\}$ with $\Sigma_c = \{\alpha, \beta, \gamma, \lambda, \mu\}$ and $\Sigma_u = \{u, v, w\}$. Assume that the specifications are $E_1 = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\gamma, \lambda\alpha, \lambda\alpha v, \lambda\alpha v\beta\}$ and $E_2 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\beta u\alpha, \alpha\beta\alpha, \alpha\beta\alpha\beta, \alpha\gamma\alpha, \mu\}$ respectively. We obtain the expansion for the empty string ε using the expansion rules and depth-first approach. Indicate the boundary strings of the expansions and verify that the boundary strings are either illegal strings or legal marked strings.

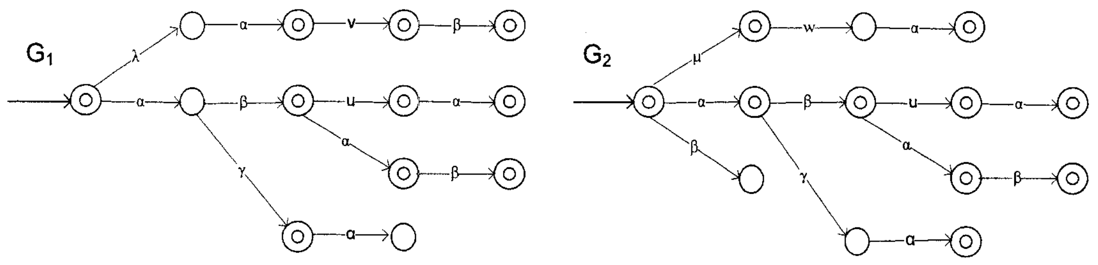

Figure 4. 8: Plant automata for Example 4.5

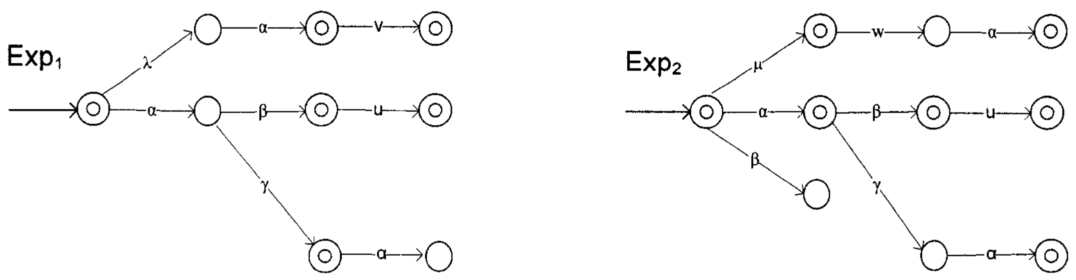

Figure 4. 9: Expansion for the initial state: Example 4.5

The classification of strings was given in Example 4.3. The expansion in Figure 4.9 is obtained as shown in Example 4.4. The boundary strings in the expansions are $\{\alpha\beta, \alpha\beta u, \alpha\gamma\alpha, \beta, \lambda\alpha v, \mu w\alpha\}$. $\{\alpha\gamma\alpha, \beta, \mu w\alpha\}$ are illegal strings. $\{\alpha\beta u, \lambda\alpha v\}$ are legal marked controllable strings. $\alpha\beta$ is a legal marked uncontrollable string. □

Lemma 4.4: Let t' be a boundary string of an expansion Exp_i (with $i \in I$) for the current string s. If $st' \notin K_{fc}$, then $t' \in L_m(Exp_i)$.

Proof: We write $t = t'\beta u$ where $\beta \in \Sigma$ and $u \in \Sigma^*$. String t could be either an illegal string or a legal marked string. Since st' is not an illegal string ($st' \notin K_{fc}$), it will be a legal marked string. Hence, we have $st' \in L_m(G_i)$, which implies that $t' \in L_m(G_i)/s$. Therefore, we have $t' \in L(Exp_i) \cap L_m(G_i)/s$, which means that $t' \in L_m(Exp_i)$. □

Lemma 4.5: Let $K \subseteq L(Exp_i)$ where Exp_i is the expansion of G_i for the current string s. If $sK \cap K_{fc} = \Phi$, then $K\Sigma_u \cap L(G_i)/s \subseteq L(Exp_i)$.

Proof: Let $t \in K$. t can only fall into the following three categories.

Case 1: t is not a boundary string. We have $t\Sigma \cap L(G_i)/s \subseteq L(Exp_i)$ since all active events of the non-boundary string t are expanded. This implies that $t\Sigma_u \cap L(G_i)/s \subseteq L(Exp_i)$.

Case 2: t is a legal marked boundary string. We have $t\Sigma_u \cap L(G_i)/s \subseteq L(Exp_i)$ since all active uncontrollable events of the marked string t are expanded.

Case 3: t is an illegal boundary string. This case is not possible since $sK \cap K_{fc} = \Phi$ implies $st \notin K_{fc}$.

Therefore, we conclude that $K\Sigma_u \cap L(G_i)/s \subseteq L(Exp_i)$. □

We would like all branches of expansion for the current string to terminate at certain steps. This will be the case if a finite number of steps will always bring a string to an illegal string or a legal marked controllable string.

Definition 4.8: N' is the maximal length of subtrace before an illegal string or a legal marked controllable string is reached:

$$N' := \begin{cases} \max\{|t| : t \in \Sigma^* \mid (\exists st \in \bigcup_{i\in I} L(G_i)) \wedge (\forall \varepsilon < p < t, sp \notin K_{illegal} \cup K_{mc})\} & \text{if existing} \\ \text{undefined} & \text{otherwise} \end{cases}$$

□

The expansion may not terminate at some strings in some problems. However, it always terminates if N' exists. In other words, the existence of N' is a sufficient condition that ensures termination of the expansion for all strings. The expansion will always terminate before or at the worst when N' step is reached for each branch.

Theorem 4.1: Expansion always terminates if N' exists.

Proof: It follows from the definition of N' and the expansion stop rules. For all branches of the expanded tree generators, a string in $K_{fc} \cup K_{mc}$ will be reached from the current string s no longer than N' steps by definition of N'. The expansion for a branch of the tree generators will terminate at the string in $K_{fc} \cup K_{mc}$ according to the stop rules. Hence, the expansion for any branch will terminate before or at the worst at the N' steps.
□

We have proved that the existence of N' will guarantee termination of expansion. Before we apply online algorithm, we may want to test if N' exists to avoid unstoppable expansion. Obviously, it exists, for example, in the special case that each plant model or specification can be represented as a finite tree generator.

In the prefix-closed case, N' exists if there is no loop formed with uncontrollable events in specification automata or plant models. There will be no uncontrollable loop in specification (legal) automata if there is no uncontrollable loop in plant automata. In case that there are uncontrollable loops in plant models or specification (legal) automata, N' still exists as long as all uncontrollable loops violate the overall specifications.

The procedure of deriving control policy online is as follows. The expansion of each model is obtained first. Then, the direct approach is taken to calculate the supremal controllable and consistent sublanguage n-tuple for the expansion. After that, the control policy for the current string is applied accordingly. Once an enabled event happens, the above procedure is repeated for the new current string. The following algorithm derives the online control decision. We denote $\bigcup_{j\in I} L(Exp_j)$ as $L(Exp)$.

Algorithm 4.1:

1) Initialization: $s = \varepsilon$.

2) Expand each model to obtain **Expi** at current string s by calling Expansion(s).

3) Let $T'_{i,0} = E_i / s \cap L_m(Exp_i)$ for all $i \in I$.

4) Calculate $(T_1',...,T_n') = SupRCS((T'_{1,0},...,T'_{n,0}))$

5) Derive control policy

$$\hat{\gamma}(s) = (\bigcup_{i\in I} \overline{T_i'}) \cap \Sigma = \{\bigcap_{i\in I} [(\overline{T_i'} \cup (\Sigma^* - L(Exp_i))]\} \cap L(Exp) \cap \Sigma .$$

6) Wait until an enabled event σ happens.

7) Replace s with $s = s\sigma$, and go back to step 2. □

Remark 4.1: The current string s may not exist in some possible plant models. For instance, if only m of total n possible plant models include s, then only the m plants that include string s are expanded. Renumber the expansion as $J = \{1',2',...,m'\}$. $(T_1',...,T_n') = SupRCS((T'_{1,0},...,T'_{n,0}))$ in Algorithm 4.1 will reduce to $(T_{1'}',...,T_{m'}') = SupRCS((T'_{1',0},...,T'_{m',0}))$ accordingly. □

The control policy at string s can be obtained from either the union operation or intersection operation once T_i' is computed. They are equivalent since we have $\bigcup_{i\in I} \overline{T_i}' = \{\bigcap_{i\in I} [(\overline{T_i}' \cup (\Sigma^* - L(Exp_i))]\} \cap L(Exp)$ which follows from consistency.

We may choose the inactive events of a state to be enabled or disabled by a supervisor without affecting the closed-loop behavior. The inactive events are chosen disabled in Algorithm 4.1.

Remark 4.2: Algorithm 4.1 takes direct approach. Instead, indirect approach can be taken for the expansions by first synthesizing overall expansion Exp and specification T_0' and then computing $T' = SupRN_bC(T_0', Exp)$. Like the offline supervision, we expect that $T' = \bigcup_{i\in I} T_i'$ and $T_i' = T' \cap L_m(Exp_i)$. □

Remark 4.3: Expansion procedure at step 2 of Algorithm 4.1 is required only for the empty string ε and all boundary strings. Instead of conducting expansion by calling Expansion(s), the expansion for non-boundary strings can be obtained directly from their previous expansions. For instance, let $Exp_{i,s}$ be the expansion for the current string s. If the next current string $s\sigma$ is a transient string in K_{trsn}, then expansion for $s\sigma$ can be obtained directly, which will be the subtree of $Exp_{i,s}$ with node $s\sigma$ as its root. □

Remark 4.4: The operation $\underline{\Omega}((T_{1,0}',\ldots,T_{n,0}')) = SupS(SupC(SupR((T_{1,0}',\ldots,T_{n,0}'))))$ that calculates $SupRCS((T_{1,0}',\ldots,T_{n,0}'))$ in Algorithm 4.1 reduces to $\underline{\Omega}((T_{1,0}',\ldots,T_{n,0}')) =$

$SupS(SupC((T_{1,0}^{'},...,T_{n,0}^{'})))$ since $(T_{1,0}^{'},...,T_{n,0}^{'})$ is relative-closed and the language remains relative-closed during the iterations. □

4.2 Validity of Algorithm 4.1

We show the validity of Algorithm 4.1 in this section. If we assume the expansion for all strings always terminates and the offline optimal solution is not empty, then Algorithm 4.1 is valid in the sense that the online direct solution $L(\hat{\gamma} / G_i)$ is the same as the offline optimal direct solution $L(\hat{V} / G_i)$ for all $i \in I$.

Theorem 4.2: (Validity) Assume N' exists for a set of plant and specification pairs G_i and E_i with $i \in I$ and the offline robust solution is not empty. Let $\hat{V}$ and $\hat{\gamma}$ be the resulting monolithic supervisors of offline Algorithm 3.2 and online Algorithm 4.1 respectively. Then, $L(\hat{V} / G_i) = L(\hat{\gamma} / G_i)$, which means $\hat{\gamma}$ is valid . □

We propose and prove some lemmas before we prove Theorem 4.2. Lemma 4.7 and Lemma 4.13 are the key to the proof of Theorem 4.2. We denote $K_i = SupRCS(E_i, L(G_i))$, $T_i = SupRCS(E_i / s, L(G_i)/s)$, and $T_i' = SupRCS$ $(E_i / s \cap L_m(Exp_i), L(Exp_i))$ with little notation abuse. K_i is the offline direct solution by Algorithm 3.2 or 3.2A. T_i is the supremal $L_m(G_i)/s$ -closed, controllable, and consistent sublanguage at the current string s with respect to the post language $L(G_i)/s$. T_i' is the supremal $L_m(Exp_i)$ -closed, controllable, and consistent sublanguage at the current string s obtained online from Algorithm 4.1. T_i is introduced to bridge the gap between T_i' and K_i.

Lemma 4.6: If $s \in \overline{K_i}$, then $\overline{K_i \cup sT_i} = \overline{K_i} \cup s\overline{T_i}$.

Proof: If follows from Lemma 4.A.2. □

If a marked strings st enabled by the offline optimal robust supervisor in $L(G_i)$, then string t will be enabled by the optimal robust supervisor in $L(G_i)/s$ and vice versa.

Lemma 4.7: Assume $s \in \overline{K_i}$. Then, $T_i = K_i / s$.

Proof: 1) First, we show $K_i / s \subseteq T_i$. For this, we prove that $K_i / s \in RCS(E_i / s, L(G_i)/s)$. Note that $K_i / s \subseteq E_i / s$ since $K_i \subseteq E_i$. Next, we establish controllability, $L_m(G_i)/s$-closure, and consistency.

i) Controllability:

$$\overline{K_i}\Sigma_{u,i} \mathrm{I}\ L(G_i) \subseteq \overline{K_i} \Rightarrow (\overline{K_i}\Sigma_{u,i} \mathrm{I}\ L(G_i))/s \subseteq \overline{K_i}/s \qquad \text{(by Lemma 4.A.3)}$$

$$\Rightarrow (\overline{K_i}\Sigma_{u,i})/s\, \mathrm{I}\ L(G_i)/s \subseteq \overline{K_i / s} \qquad \text{(by Lemma 2.2, Lemma 2.1)}$$

$$\Rightarrow (\overline{K_i}/s)\Sigma_{u,i}\ \mathrm{I}\ L(G_i)/s \subseteq \overline{K_i / s} \qquad \text{(by Lemma 2.3)}$$

$$\Rightarrow \overline{K_i / s}\Sigma_{u,i}\ \mathrm{I}\ L(G_i)/s \subseteq \overline{K_i / s}. \qquad \text{(by Lemma 2.1)}$$

ii) $L_m(G_i)/s$-closure:

$$\overline{K_i}\ \mathrm{I}\ L_m(G_i) = K_i \Rightarrow (\overline{K_i}\ \mathrm{I}\ L_m(G_i))/s = K_i / s$$

$$\Rightarrow \overline{K_i}/s \cap L_m(G_i)/s = K_i/s \qquad \text{(by Lemma 2.2)}$$

$$\Rightarrow \overline{K_i/s} \cap L_m(G_i)/s = K_i/s. \qquad \text{(by Lemma 2.1)}$$

iii) Consistency:

$$\overline{K_i} \cap (L(G_j) - \overline{K_j}) = \Phi \Rightarrow (\overline{K_i} \cap (L(G_j) - \overline{K_j}))/s = \Phi$$

$$\Rightarrow \overline{K_i}/s \cap (L(G_j) - \overline{K_j})/s = \Phi \qquad \text{(by Lemma 2.2)}$$

$$\Rightarrow \overline{K_i/s} \cap (L(G_j)/s - \overline{K_j}/s) = \Phi \qquad \text{(by Lemma 2.1, Lemma 4.A.9)}$$

$$\Rightarrow \overline{K_i/s} \cap (L(G_j)/s - \overline{K_j/s}) = \Phi. \qquad \text{(by Lemma 2.1)}$$

Therefore, $K_i/s \in RCS(E_i/s, L(G_i)/s)$ which implies that $K_i/s \subseteq T_i$ since T_i is the supremal element.

2) Now, we show $T_i \subseteq K_i/s$. We first prove that $sT_i \subseteq K_i$ by showing that $K_i \cup sT_i$ is controllable, $L_m(G_i)$-closed, and consistent.

i) Controllability:

$$\overline{K_i \cup sT_i}\Sigma_{u,i} \cap L(G_i) = (\overline{K_i} \cup s\overline{T_i})\Sigma_{u,i} \cap L(G_i) \qquad \text{(by Lemma 4.6)}$$

$$= (\overline{K_i}\Sigma_{u,i} \cup s\overline{T_i}\Sigma_{u,i}) \cap L(G_i)$$

$$= (\overline{K_i}\Sigma_{u,i} \cap L(G_i)) \cup (s\overline{T_i}\Sigma_{u,i} \cap L(G_i))$$

$$\subseteq \overline{K_i} \cup (s\overline{T_i}\Sigma_{u,i} \cap L(G_i)) \qquad (\ K_i \text{ is controllable with respect to } L(G_i)\ )$$

$$= \overline{K_i} \cup s(\overline{T_i}\Sigma_{u,i} \cap L(G_i)/s) \qquad (\text{ by Lemma 4.A.8 })$$

$$\subseteq \overline{K_i} \cup s\overline{T_i} \qquad (\ T_i \text{ is controllable with respect to } L(G_i)/s\ )$$

$$= \overline{K_i \cup sT_i}. \qquad (\text{ by Lemma 4.6 })$$

ii) $L_m(G_i)$-closure:

$$\overline{K_i \cup sT_i}\ \mathrm{I}\ L_m(G_i) = (\overline{K_i} \cup s\overline{T_i}) \cap L_m(G_i) \qquad \text{(by Lemma 4.6)}$$

$$= (\overline{K_i} \cap L_m(G_i)) \cup (s\overline{T_i} \cap L_m(G_i)$$

$$= K_i \cup (s\overline{T_i} \cap L_m(G_i)) \qquad \text{(} K_i \text{ is relative-closed wrt. Lm(Gi))}$$

$$= K_i \cup s(\overline{T_i} \cap L_m(G_i)/s) \qquad \text{(by Lemma 4.A.8)}$$

$$= K_i \cup sT_i . \qquad \text{(} T_i \text{ is relative-closed wrt. Lm(Gi/s))}$$

iii) Consistency:

$$\overline{K_i \cup sT_i} \cap (L(G_j) - \overline{K_j \cup sT_j}) = (\overline{K_i} \cup s\overline{T_i}) \cap (L(G_j) - \overline{K_j \cup sT_j}) \qquad \text{(by Lemma 4.6)}$$

$$= [\overline{K_i} \cap (L(G_j) - \overline{K_j \cup sT_j})] \cup [s\overline{T_i} \cap (L(G_j) - \overline{K_j \cup sT_j})]$$

$$\subseteq [\overline{K_i} \cap (L(G_j) - \overline{K_j})] \cup [s\overline{T_i} \cap (L(G_j) - \overline{sT_j})]$$

$$= \Phi \cup [s\overline{T_i} \cap (L(G_j) - \overline{sT_j})] \qquad \text{(} K_i \text{ is consistent wrt. L(Gi))}$$

$$= s\overline{T_i} \cap [L(G_j) - (\bar{s} \cup s\overline{T_j})] \qquad \text{(by Corollary 4.A.1)}$$

$$\subseteq s\overline{T_i} \cap (L(G_j) - s\overline{T_j})$$

$$= s[\overline{T_i} \cap (L(G_j) - s\overline{T_j})/s] \qquad \text{(by Lemma 4.A.8)}$$

$$= s[\overline{T_i} \cap (L(G_j)/s - (s\overline{T_j})/s)] \qquad \text{(by Lemma 4.A.9)}$$

$$= s[\overline{T_i} \cap (L(G_j)/s - \overline{T_j})] \qquad \text{(by Lemma 4.A.6)}$$

$$= \Phi . \qquad \text{(} T_i \text{ is consistent wrt. L(Gi)/s)}$$

Hence, we have $\overline{K_i \cup sT_i} \cap (L(G_j) - \overline{K_j \cup sT_j}) = \Phi$. As a result, $K_i \cup sT_i \in RCS(E_i, L(G_i))$ and thus $K_i \mathrm{Y} sT_i \subseteq SupRCS(E_i, L(G_i)) = K_i$ which implies

$sT_i \subseteq K_i$. Therefore, we have $(sT_i)/s \subseteq K_i/s$ by Lemma 4.A.3 and thus $T_i \subseteq K_i/s$ by Lemma 4.A.6. Hence, we conclude that $T_i = K_i/s$. □

Lemmas 4.8 to 4.13 are used to show Lemma 4.14, which indicates that the unexpanded part will not affect the control decision in comparison with full expansion.

If a string t is enabled by the optimal robust supervisor in some expansion $L(G_i)/s$, then st has to be legal. In other words, all illegal strings will be removed by the optimal robust supervisor for the expansion $L(G_i)/s$.

Lemma 4.8: If $t \in \overline{T_i}$ for some $i \in I$, then $st \notin K_{fc}$.

Proof: $t \in \overline{T_i} \Rightarrow \forall j \in I : t \notin L(G_j)/s - \overline{T_j}$. (T_i is consistent wrt. $L(G_i)/s$)

For every $j \in I$:

Case 1: $st \notin L(G_j) \Rightarrow st \notin L(G_j) - \overline{E_j}$.

Case 2: $st \in L(G_j) \Rightarrow t \in L(G_j)/s$

$$\Rightarrow t \in \overline{T_j}$$

$$\Rightarrow t \in \overline{E_j/s}$$

$$\Rightarrow t \in \overline{E_j}/s$$

$$\Rightarrow st \in \overline{E_j}$$

$$\Rightarrow st \notin L(G_j) - \overline{E_j}.$$

Hence, we always have $st \notin L(G_j) - \overline{E_j}$ for all $j \in I$. Therefore, we have $st \notin K_{fc}$. □

The optimal supervisor for the expansion $L(G_i)/s$ will be a nonblocking supervisor if it is used to control the expansion Exp_i.

Lemma 4.9: $\overline{T_i \cap L_m(Exp_i)} = \overline{T_i} \cap L(Exp_i)$

Proof: We first show that $\overline{T_i \cap L_m(Exp_i)} \supseteq \overline{T_i} \cap L(Exp_i)$. Let $r \in \overline{T_i} \cap L(Exp_i)$. Then, we have $r \in L(Exp_i)$ and $r \in \overline{T_i}$, which means there exists a string t such that $t \geq r$ and $t \in T_i$. Obviously, we have $t \in L(G_i)/s$.

Case 1: $t \in L(Exp_i)$

$$t \in T_i \cap L(Exp_i) \Rightarrow t \in T_i \cap L_m(G_i)/s \cap L(Exp_i)$$
$$\Rightarrow t \in T_i \cap L_m(Exp_i)$$
$$\Rightarrow r \in \overline{T_i \cap L_m(Exp_i)}.$$

Case 2: $t \notin L(Exp_i)$. We denote the boundary string in $L(Exp_i)$ which leads to t as t'. Obviously, we have $r \leq t' < t$.

$$t \in T_i \Rightarrow t' \in \overline{T_i}$$
$$\Rightarrow st' \notin K_{fc} \qquad \text{(by Lemma 4.8)}$$
$$\Rightarrow t' \in L_m(Exp_i) \qquad \text{(by Lemma 4.4)}$$
$$\Rightarrow t' \in \overline{T_i} \cap L_m(Exp_i)$$
$$\Rightarrow t' \in \overline{T_i} \cap L_m(G_i)/s \cap L_m(Exp_i)$$

$\Rightarrow t' \in T_i \cap L_m(Exp_i)$ (T_i is $L_m(G_i)/s$ -closed)

$\Rightarrow r \in \overline{T_i \cap L_m(Exp_i)}$.

Therefore, $\overline{T_i \cap L_m(Exp_i)} \supseteq \overline{T_i} \cap L(Exp_i)$. On the other hand, it is immediate that $\overline{T_i \cap L_m(Exp_i)} \subseteq \overline{T_i} \cap L(Exp_i)$. As a result, we have $\overline{T_i \cap L_m(Exp_i)} = \overline{T_i} \cap L(Exp_i)$. □

If a string t is enabled by the optimal robust supervisor in some expansion $L(Exp_i)$, then st has to be legal. In other words, all illegal strings will be removed by the optimal robust supervisor for the expansion $L(Exp_i)$.

Lemma 4.10: If $t \in \overline{T_i}'$ for some $i \in I$, then $st \notin K_{fc}$.

Proof: $t \in \overline{T_i}' \Rightarrow t \in L(Exp_i)$

$\Rightarrow \forall j \in I : t \notin L(G_j)/s - L(Exp_j)$. (4.1) (by Lemma 4.2)

$t \in \overline{T_i}' \Rightarrow \forall j \in I : t \notin L(Exp_j) - \overline{T_j}'$. (4.2) (T_i' is consistent wrt. $L(Exp_i)$)

For every $j \in I$:

Case 1: $st \notin L(G_j) \Rightarrow st \notin L(G_j) - \overline{E_j}$.

Case 2: $st \in L(G_j) \Rightarrow t \in L(G_j)/s$

$\Rightarrow t \in L(Exp_j)$ (by 4.1)

$\Rightarrow t \in \overline{T_j}'$ (by 4.2)

$\Rightarrow t \in \overline{E_j / s}$

$\Rightarrow t \in \overline{E_j} / s$ (by Lemma 2.1)

$\Rightarrow st \in \overline{E_j}$

$\Rightarrow st \notin L(G_j) - \overline{E_j}$.

Hence, we always have $st \notin L(G_j) - \overline{E_j}$ for all $j \in I$. Therefore, we have $st \notin K_{fc}$. □

Corollary 4.1: For every $i \in I$, $s\overline{T_i'} \cap K_{fc} = \Phi$.

Proof: Let $r \in s\overline{T_i'}$. We can write $r = st$ where $t \in \overline{T_i'}$. Hence, $st \notin K_{fc}$ by Lemma 4.10 and thus $r \notin K_{fc}$. This implies that $s\overline{T_i'} \cap K_{fc} = \Phi$. □

All active uncontrollable events of the strings that are enabled by the supremal supervisor for expansion $L(Exp_i)$ are expanded.

Lemma 4.11: $\overline{T_i'}\Sigma_u \cap L(G_i)/s = \overline{T_i'}\Sigma_u \cap L(Exp_i)$

Proof: We have $\overline{T_i'}\Sigma_u \cap L(G_i)/s = L(Exp_i)$ by Corollary 4.1 and Lemma 4.5. This implies that $\overline{T_i'}\Sigma_u \cap L(G_i)/s \subseteq \overline{T_i'}\Sigma_u \cap L(Exp_i)$. On the other hand, it is straightforward that $\overline{T_i'}\Sigma_u \cap L(G_i)/s \supseteq \overline{T_i'}\Sigma_u \cap L(Exp_i)$. As a result, we have $\overline{T_i'}\Sigma_u \cap L(G_i)/s =$ $\overline{T_i'}\Sigma_u \cap L(Exp_i)$ □

If a string t is enabled by the optimal robust supervisor in some expansion $L(Exp_i)$, then it will be expanded in other models which include it. Namely, if a string is not expanded in one model $L(Exp_j)$, it will not be enabled by the optimal robust supervisor in the expansion $L(Exp_i)$.

Lemma 4.12: $\overline{T_i}' \cap (L(G_j)/s - L(Exp_j)) = \Phi$.

Proof: We have $\overline{T_i}' \cap (L(G_j)/s - L(Exp_j)) \subseteq L(Exp_i) \cap (L(G_j)/s - L(Exp_j))$, which implies that $\overline{T_i}' \cap (L(G_j)/s - L(Exp_j)) = \Phi$ since $L(Exp_i) \cap (L(G_j)/s - L(Exp_j)) = \Phi$ by Lemma 4.2. □

If a marked string inside $L(Exp_i)$ is enabled by the optimal robust supervisor in the expansion $L(G_i)/s$, it will be enabled by the optimal robust supervisor in the expansion $L(Exp_i)$. In addition, if a marked string is enabled by the optimal robust supervisor in the expansion $L(Exp_i)$, it will be enabled by the optimal robust supervisor in the expansion $L(G_i)/s$.

Lemma 4.13: $T_i' = T_i \cap L_m(Exp_i)$.

($\subseteq$): We first show that $T_i' \subseteq T_i$ by showing $T_i' \in RCS(E_i/s, L(G_i)/s)$.

1) We prove that T_i' is controllable with respect to $L(G_i)/s$.

$$\overline{T_i}'\Sigma_u \cap L(G_i)/s = \overline{T_i}'\Sigma_u \cap L(Exp_i) \qquad \text{(by Lemma 4.11)}$$

$$\subseteq \overline{T_i}'. \qquad \text{(} T_i' \text{ is controllable with respect to } L(Exp_i) \text{)}$$

2) We then show that T_i' is $L_m(G_i)/s$-closed

$$\overline{T_i}' \cap L_m(G_i)/s = \overline{T_i}' \cap L(Exp_i) \cap L_m(G_i)/s$$

$$= \overline{T_i}' \cap L_m(Exp_i)$$

$$= T_i'. \qquad (T_i' \text{ is } L_m(Exp_i)\text{-closed})$$

3) We finally show that T_i' and T_j' are consistent with respect to $L(G_i)/s$ and $L(G_j)/s$.

$$\overline{T_i}' \cap (L(G_j)/s - \overline{T_j}') = \overline{T_i}' \cap [(L(G_j)/s - L(Exp_j) \cup (L(Exp_j) - \overline{T_j}')]$$

(by Lemma 4.A.11)

$$= [\overline{T_i}' \cap (L(G_j)/s - L(Exp_j))] \cup [\overline{T_i}' \cap (L(Exp_j) - \overline{T_j}')]$$

$$= [\overline{T_i}' \cap (L(G_j)/s - L(Exp_j))] \cup \Phi$$

(T_i' is consistent with respect to $L(Exp_i)$)

$$= \Phi. \qquad (\text{by Lemma 4.12})$$

Hence, we have $T_i' \in RCS(E_i/s, L(G_i)/s)$. We infer that $T_i' \subseteq T_i$ since T_i is the supremal element of $RCS(E_i/s, L(G_i)/s)$. In addition, it is immediate that $T_i' \subseteq L_m(Exp_i)$. Therefore, we conclude that $T_i' \subseteq T_i \cap L_m(Exp_i)$.

($\supseteq$): We next show $T_i \cap L_m(Exp_i)$ is a sublanguage of T_i'.

1) We prove that $T_i \cap L_m(Exp_i)$ is controllable with respect to $L(Exp_i)$.

$$\overline{T_i \cap L_m(Exp_i)}\Sigma_u \cap L(Exp_i) \subseteq \overline{T_i}\Sigma_u \cap L(Exp_i)$$

$$= \overline{T_i}\Sigma_u \cap L(G_i)/s \cap L(Exp_i)$$

$$\subseteq \overline{T_i} \cap L(Exp_i)$$

(T_i is controllable with respect to $L(G_i)/s$)

$$= \overline{T_i \cap L_m(Exp_i)} \qquad \text{(by Lemma 4.9)}$$

2) We next show that $T_i \cap L_m(Exp_i)$ is $L_m(Exp_i)$-closed.

$$\overline{T_i \cap L_m(Exp_i)} \cap L_m(Exp_i) \subseteq \overline{T_i} \cap L_m(Exp_i)$$

$$= \overline{T_i} \cap L_m(G_i)/s \cap L_m(Exp_i)$$

$$= T_i \mathrm{I}\ L_m(Exp_i). \qquad \text{(} T_i \text{ is } L_m(G_i)/s\text{-closed)}$$

On the other hand, we have $\overline{T_i \cap L_m(Exp_i)} \cap L_m(Exp_i) \supseteq T_i \cap L_m(Exp_i)$. Therefore, we conclude that $\overline{T_i \cap L_m(Exp_i)} \cap L_m(Exp_i) = T_i \cap L_m(Exp_i)$.

3) Finally, we last show that $T_i \mathrm{I}\ L_m(Exp_i)$ and $T_j \mathrm{I}\ L_m(Exp_j)$ are consistent with respect to $L(Exp_i)$ and $L(Exp_j)$.

$$\overline{T_i} \cap (L(G_j)/s - \overline{T_j}) = \Phi \qquad \text{(} T_i \text{ is consistent with respect to } L(G_i)/s \text{)}$$

$$\Rightarrow \overline{T_i} \cap \overline{L_m(Exp_i)} \cap L(Exp_j) \cap (L(G_j)/s - \overline{T_j}) = \Phi$$

$$\Rightarrow \overline{T_i \cap L_m(Exp_i)} \cap [L(Exp_j) \cap (L(G_j)/s - \overline{T_j})] = \Phi$$

$$\Rightarrow \overline{T_i \cap L_m(Exp_i)} \cap [(L(Exp_j) \cap L(G_j)/s) - (L(Exp_j) \cap \overline{T_j})] = \Phi$$

(by Lemma 4.A.12)

$$\Rightarrow \overline{T_i \cap L_m(Exp_i)} \cap [L(Exp_j) - \overline{T_j \cap L_m(Exp_j)}] = \Phi. \qquad \text{(by Lemma 4.9)}$$

From 1) to 3), we have $T_i \cap L_m(Exp_i) \in RCS(E_i / s \cap L_m(Exp_i), L(Exp_i))$ and thus $T_i \cap L_m(Exp_i) \subseteq SupRCS(E_i / s \cap L_m(Exp_i), L(Exp_i)) = T_i'$. Therefore, we conclude that $T_i' = T_i \cap L_m(Exp_i)$. □

Remark 4.5: The results T_i' for the current string s can be reused for the following strings until a boundary string is reached. For instance, let $Exp_{i,s}$ be the expansion and $T_{i,s}'$ be the supremal elememt for current string s. If the next current string $s\sigma$ is a transient string, then $T_{i,s\sigma}' = T_{i,s}' / \sigma$, which means the control policy at string $s\sigma$ will be $\gamma(s\sigma) = [\bigcup_{i \in I}(T_{i,s}' / \sigma)] \cap \Sigma$. □

Remark 4.6: A string included in $sT_{i,s}'$ will be included in $stT_{i,st}'$. On the other hand, a string included in $stT_{i,st}'$ will be included in $sT_{i,s}'$ if it belongs to $sL(Exp_{i,s})$. □

If an active event of the current string s is enabled by the optimal robust supervisor in the expansion $L(G_i)/s$, it will be enabled by the optimal robust supervisor in the expansion $L(Exp_i)$ and vice versa. The unexpanded part does not affect the optimal control decision for the current string s.

Lemma 4.14: $\overline{T_i'} \cap \Sigma = \overline{T_i} \cap \Sigma$.

Proof: $\overline{T_i} \cap \Sigma = \overline{T_i} \cap L(G_i)/s \cap \Sigma$

$= \overline{T_i} \cap L(Exp_i) \cap \Sigma$ (by Lemma 4.3)

$= \overline{T_i \cap L_m(Exp_i)} \cap \Sigma$ (by Lemma 4.9)

$= \overline{T_i}'\mathrm{I}\ \Sigma.$ (by Lemma 4.13) □

Lemma 4.15: If language n-tuple $(M_1,...,M_n)$ is consistent with respect to language n-tuple $(L(G_1),...,L(G_n))$, then $\overline{M_j} \cap L(G_i) \subseteq \overline{M_i}$ for all $i, j \in I$.

Proof: If $\overline{M_j} \cap L(G_i) = \Phi$, then the lemma is obviously true.

If $\overline{M_j} \cap L(G_i) \neq \Phi$, then let $s \in \overline{M_j} \cap L(G_i)$. Now we have:

$$s \in \overline{M_j} \cap L(G_i) \Rightarrow s \in \overline{M_j} \wedge s \in L(G_i)$$

$$\Rightarrow (s \notin L(G_i) - \overline{M_i}) \wedge s \in L(G_i) \qquad \text{(by Consistency)}$$

$$\Rightarrow s \in \overline{M_i}.$$

As a result, we have $\overline{M_j} \cap L(G_i) \subseteq \overline{M_i}$. □

Corollary 4.2 If language n-tuple $(M_1,...,M_n)$ is consistent with respect to language n-tuple $(L(G_1),...,L(G_n))$, then $\overline{M_j} \cap L(G_i) \subseteq \overline{M_i} \cap L(G_i)$ for all $i, j \in I$.

Proof: It is immediate from Lemma 4.15 and $\overline{M_j} \cap L(G_i) \subseteq L(G_i)$. □

Lemma 4.16: If language n-tuple $(M_1,...,M_n)$ is consistent with respect to language n-tuple $(L(G_1),...,L(G_n))$, then $(\underset{j \in I}{\mathrm{Y}} \overline{M_j}) \cap L(G_i) = \overline{M_i}$ for all $i \in I$.

Proof: $\underset{j \in I}{\mathrm{Y}} \overline{M_j} \cap L(G_i) = [\overline{M_i} \cap L(G_i)] \cup \{ \underset{j \in I, j \neq i}{\mathrm{Y}} [\overline{M_j} \cap L(G_i)]\}$

$$= \overline{M_i} \cap L(G_i) \qquad \text{(by Corollary 4.2)}$$

$= \overline{M_i}$. □

We are now ready to provide the proof of Theorem 4.1. We prove $L(\hat{V}/G_i) = L(\hat{\gamma}/G_i)$ by induction on the length of strings in $L(\hat{V}/G_i) = \overline{K_i}$.

Proof of Theorem 4.2: (Validity)

By Induction:

1) **Induction base:** The induction base holds for zero-length string since we have $\varepsilon \in L(\hat{V}/G_i)$ and $\varepsilon \in L(\hat{\gamma}/G_i)$.

2) **Induction hypothesis:** we assume that $s \in L(\hat{V}/G_i)$ iff $s \in L(\hat{\gamma}/G_i)$ for all strings with length n or less ($|s| \leq n$) .

3) **Induction step:** We then show $s\sigma \in L(\hat{V}/G_i)$ iff $s\sigma \in L(\hat{\gamma}/G_i)$ for all $\sigma \in \Sigma$.

$s\sigma \in L(\hat{V}/G_i) \Leftrightarrow s\sigma \in \overline{K_i}$

$\Leftrightarrow \sigma \in \overline{K_i}/s$

$\Leftrightarrow \sigma \in \overline{K_i/s}$ (by Lemma 2.1)

$\Leftrightarrow \sigma \in \overline{T_i}$. (by Lemma 4.7)

$\Leftrightarrow \sigma \in \overline{T_i}'$ (by Lemma 4.14)

$\Leftrightarrow \sigma \in (\bigcup_{j \in I} \overline{T_j}') \cap L(Exp_i)$

(by consistency of $(T_1',...,T_n')$ and Lemma 4.16)

$\Leftrightarrow \sigma \in \bigcup_{j \in I} \overline{T_j}'$ (All active events of string s are expanded)

$\Leftrightarrow s \in L(\hat{\gamma} / G_i) \wedge s\sigma \in L(G_i) \wedge \sigma \in \hat{\gamma}(s)$ (by the definition of $\hat{\gamma}$)

$\Leftrightarrow s\sigma \in L(\hat{\gamma} / G_i)$. (by the definition of $L(\hat{\gamma} / G_i)$) □

Definition 4.9: We say run-time error happens at string s if $T_i' = \Phi$ for some $i \in I$. □

Proposition 4.1: Assume that $K_i \neq \Phi$ for all $i \in I$. Then, no run-time error happens in Algorithm 4.1.

Proof: It suffices to show $T_i' \neq \Phi$ for all strings $s \in \overline{K_i}$. We have:

$$\begin{aligned} \overline{T_i}' &= \overline{T_i \cap L_m(Exp_i)} && \text{(by Lemma 4.13)} \\ &= \overline{T_i} \cap L(Exp_i) && \text{(by Lemma 4.9)} \\ &= \overline{K_i / s} \cap L(Exp_i). && \text{(by Lemma 4.7)} \end{aligned}$$

We also have $s \in L(G_i)$ and thus $L(Exp_i) \neq \Phi$. Moreover, we have $K_i / s \neq \Phi$ by assumption and thus $\overline{K_i / s} \neq \Phi$. As a result, we conclude that $\overline{T_i}' \neq \Phi$ and thus there is no run-time error. □

Algorithm 4.1 derives a single robust control policy. Only one supervisor is synthesized which we refer to as monolithic supervisor. Alternatively, we may derive a supervisory decision from each language T_i' and combine the supervisory decisions to make control decision. We refer to this as modular supervision even though T_i' is calculated using all expansion windows. The step 5 of Algorithm 4.1 can be modified as follows in the modular supervision. The other steps are unchanged.

Algorithm 4.1A:

All steps are the same as Algorithm 4.1 except step 5:

Step 5) Derive control action $\gamma_i(s) = \overline{T_i'} \cap \Sigma$ for each model. Obtain control policy $\tilde{\gamma}(s) = \bigcup_{i \in I} \gamma_i(s) = \{\bigcap_{i \in I} [\gamma_i(s) \cup (\Sigma - (L(Exp_i) \cap \Sigma))]\} \cap L(Exp)$. □

Just like the offline algorithms, we expect the equivalence of the online monolithic and modular supervision. The following theorem gives the formal proof of their equivalence.

Theorem 4.3: Let $\hat{\gamma}$ and $\tilde{\gamma}$ be the monolithic and modular supervisor obtained from Algorithm 4.1 and 4.1A. Then, $L(\hat{\gamma}/G_i) = L(\tilde{\gamma}/G_i)$ and $L_m(\hat{\gamma}/G_i) = L_m(\tilde{\gamma}/G_i)$.

Proof: We know $\varepsilon \in L(\hat{\gamma}/G_i)$ and $\varepsilon \in L(\tilde{\gamma}/G_i)$. It suffices to prove $L(\hat{\gamma}/G_i) = L(\tilde{\gamma}/G_i)$ by showing $\hat{\gamma}(s) = \tilde{\gamma}(s)$ for any string s. We have $\hat{\gamma}(s) = (\bigcup_{i \in I} \overline{T_i'}) \cap \Sigma$ and $\tilde{\gamma}(s) = \bigcup_{i \in I} (\overline{T_i'} \cap \Sigma)$ for all $s \in L(G)$.

$(\bigcup_{i \in I} \overline{T_i'}) \cap \Sigma = \bigcup_{i \in I} (\overline{T_i'} \cap \Sigma)$ implies that $\hat{\gamma}(s) = \tilde{\gamma}(s)$ and thus $L(\hat{\gamma}/G_i) = L(\tilde{\gamma}/G_i)$. In addition, we have $L_m(\hat{\gamma}/G_i) = L_m(\tilde{\gamma}/G_i)$ since by definition $L_m(\hat{\gamma}/G_i) = L(\hat{\gamma}/G_i) \cap L_m(G_i)$ and $L_m(\tilde{\gamma}/G_i) = L(\tilde{\gamma}/G_i) \cap L_m(G_i)$. □

We have shown the validity of Algorithm 4.1. The validity of Algorithm 4.1A follows immediately since it is equivalent to Algorithm 4.1. Similarly, there is also no run-time error in Algorithm 4.1A.

4.3 VLP Direct Solution: Example

Online direct approach Algorithm 4.1 is used to solve the same problem of Example 2.1. The control policy for the empty string is derived in the following example to illustrate the procedure of the algorithm. The control policy for other strings can be obtained the same way.

Example 4.6: A system model could be either G$_1$ or G$_2$ shown below in Figure 4.10. Assume that $\Sigma = \{\alpha,\beta,\gamma,\lambda,\mu,u,v,w\}$ with $\Sigma_c = \{\alpha,\beta,\gamma,\lambda,\mu\}$ and $\Sigma_u = \{u,v,w\}$. Given certain specifications, we synthesize an optimal admissible nonblocking robust supervisor which solves the problem.

The language generated and marked by the plant models are:

$L(G_1) = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma,\alpha\gamma\alpha,\lambda,\lambda\alpha,\lambda\alpha v,\lambda\alpha v\beta\}$

$L(G_2) = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma,\alpha\gamma\alpha,\mu,\mu w,\mu w\alpha,\beta\}$

$L_m(G_1) = \{\varepsilon,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma,\lambda\alpha,\lambda\alpha v,\lambda\alpha v\beta\}$

$L_m(G_2) = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma\alpha,\mu,\mu w\alpha\}$.

Assume the specification languages are:

$E_1 = \{\varepsilon,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\gamma,\lambda\alpha,\lambda\alpha v,\lambda\alpha v\beta\}$.

$E_2 = \{\varepsilon,\alpha,\alpha\beta,\alpha\beta u,\alpha\beta u\alpha,\alpha\beta\alpha,\alpha\beta\alpha\beta,\alpha\gamma\alpha,\mu\}$.

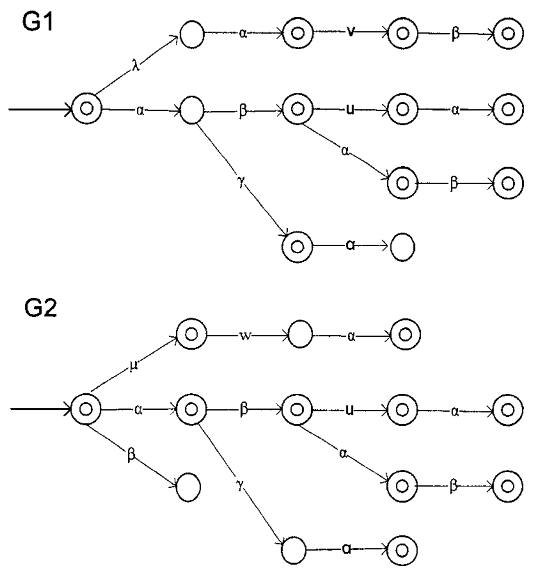

Figure 4. 10: Plant automata for Example 4.6

We now employ Algorithm 4.1 to derive control policy for the empty string.

1) Initialization: $s = \varepsilon$.

2) Expansion for the empty string is illustrated in Example 4.4. The resulting expansions of G_1 and G_2 for the initial string are Exp_1 and Exp_2 as shown in Figure 4.11.

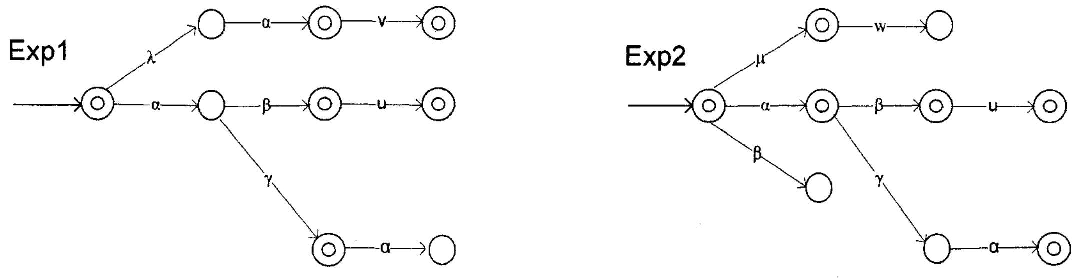

Figure 4. 11: Expansion for the initial state: Example 4.6

3) Let $T_{1,0}' = E_1 / s \cap L_m(Exp_i) = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\gamma, \lambda\alpha, \lambda\alpha v\}$ and $T_{2,0}' = E_2 / s \cap L_m(Exp_2)$ $= \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\gamma\alpha, \mu\}$.

4) Calculate $(T_1', T_2') = SupRCS((T_{1,0}', T_{2,0}'))$: Algorithm 3.2A is employed to compute the supremal element. $T_{1,0}'$ and $T_{2,0}'$ are relative-closed with respect to $L_m(Exp_1)$ and $L_m(Exp_2)$ respectively.

Iteration 1:

$T_{1,0}'$ is controllable, however, $T_{2,0}'$ is not controllable since uncontrollable event w at string μ leads outside $\overline{T_{2,0}'}$. The supermal controllable sublanguages are first calculated in this iteration.

$SupC(T_{1,0}') = \{\varepsilon, \alpha\beta, \alpha\beta u, \alpha\gamma, \lambda\alpha, \lambda\alpha v\}$ and $SupC(T_{2,0}') = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \alpha\gamma\alpha\}$.

Obviously, $(SupC(T_{1,0}'), SupC(T_{2,0}'))$ is not consistent. For example, $\alpha\gamma\alpha$ is disabled in Exp_1 but enabled in Exp_2. We next calculate the supremal consistent sublanguage in this iteration.

$T_{1,1}' = SupS(SupC(T_{1,0}')) = \{\varepsilon, \alpha\beta, \alpha\beta u, \lambda\alpha, \lambda\alpha v\}$.

$T_{2,1}' = SupS(SupC(T_{2,0}')) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u\}$.

After this iteration, $T_{1,1}'$ and $T_{2,1}'$ are different with $T_{1,0}'$ and $T_{2,0}'$ respectively and thus next iteration has to be taken.

Iteration 2:

$SupC(T_{1,1}') = \{\varepsilon, \alpha\beta, \alpha\beta u, \lambda\alpha, \lambda\alpha v\}$ and $SupC(T_{2,1}') = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u\}$.

$T'_{1,2} = SupS(SupC(T'_{1,1})) = \{\varepsilon, \alpha\beta, \alpha\beta u, \lambda\alpha, \lambda\alpha v\}$.

$T'_{2,2} = SupS(SupC(T'_{2,1})) = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u\}$.

Neither $T'_{1,2}$ nor $T'_{2,2}$ changes from $T'_{1,1}$ and $T'_{2,1}$ respectively. Hence, we stop here. We can verify that the resulting language 2-tuple $(T'_{1,2}, T'_{2,2})$ is an element of $RCS(T'_{1,0}, T'_{2,0})$. As a result, we have $T'_1 = \{\varepsilon, \alpha\beta, \alpha\beta u, \lambda\alpha, \lambda\alpha v\}$ and $T'_2 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u\}$.

5) The overall solution is $T'_1 \cup T'_2 = \{\varepsilon, \alpha, \alpha\beta, \alpha\beta u, \lambda\alpha, \lambda\alpha v\}$. Hence, the control decision for the empty string is $\hat{\gamma}(\varepsilon) = (\overline{T'_1 \cup T'_2}) \cap \Sigma = \{\alpha, \lambda\}$. □

The online control decision for the empty string is the same as the offline direct solution. The procedure of algorithm 4.1 is illustrated in this example. Online algorithm only computes the solution for the expansion window rather than the entire plant. It reduces computer memory requirement at the cost of online computing power.

Appendix 4.1: Lemmas for Validity Proof

Lemma 4.A.1: Let $A, B \in \Sigma^*$. Then, $\overline{AB} = \overline{A} \cup A\overline{B}$.

Proof: $(\supseteq)$: First we show $\overline{A} \subseteq \overline{AB}$ and $A\overline{B} \subseteq \overline{AB}$.

If $\overline{A} = \Phi$, then $\overline{A} \subseteq \overline{AB}$ will be ture. Suppose $\overline{A} \neq \Phi$. For every $s \in \overline{A}$, there exists $st \in A$ thus we have $stu \in AB$ for any $u \in B$. This implies that $s \in \overline{AB}$. Hence, we have $\overline{A} \subseteq \overline{AB}$.

For every $s \in A\overline{B}$, there exists $u \in A$ and $v \in \overline{B}$ such that $s = uv$. $v \in \overline{B}$ implies that there exists $vw \in B$. Hence, we have $uvw \in AB$, which means $sw \in AB$ and thus $s \in \overline{AB}$. Therefore, we have $A\overline{B} \subseteq \overline{AB}$.

As a result, we conclude that $\overline{A} \cup A\overline{B} \subseteq \overline{AB}$.

$(\subseteq)$: Let $s \in \overline{AB}$. There exists $st \in AB$. Rewrite st as uv such that $u \in A$ and $v \in B$.

Case 1: $s \leq u$.

$s \leq u$ implies that $s \in \overline{A}$ and thus $s \in \overline{A} \cup A\overline{B}$.

Case 2: $s > u$.

Rewrite s as ua. This implies that $at = v$ and thus $a \in \overline{v}$. We infer that $a \in \overline{B}$ and thus $ua \in A\overline{B}$. Hence, we have $s \in A\overline{B}$, which means $s \in \overline{A} \cup A\overline{B}$.

From case 1 and 2, we have $\overline{AB} \subseteq \overline{A} \cup A\overline{B}$. Therefore, $\overline{AB} = \overline{A} \cup A\overline{B}$. □

Corollary 4.A.1: For $s \in \Sigma^*$ and $A \subseteq \Sigma^*$, $\overline{sA} = \overline{s} \cup s\overline{A}$.

Proof: It is immediate from Lemma 4.A.1. □

Lemma 4.A.2: Let $A, B, C \in \Sigma^*$. If $\overline{A} \subseteq \overline{C}$, then $\overline{C \cup AB} = \overline{C} \cup A\overline{B}$.

Proof: $\overline{C \cup AB} = \overline{C} \cup \overline{AB} = \overline{C} \cup \overline{A} \cup A\overline{B}$ by Lemma 4.A.1. $\overline{C} \cup \overline{A} \cup \overline{AB} = \overline{C} \cup A\overline{B}$ since $\overline{A} \subseteq \overline{C}$. Hence, we have $\overline{C \cup AB} = \overline{C} \cup A\overline{B}$. □

Lemma 4.A.3: Let $A, B \subseteq \Sigma^*$. If $A \subseteq B$, then $A/s \subseteq B/s$.

Proof: $t \in A/s$ implies that $st \in A$. Therefore, we have $st \in B$ and thus $t \in B/s$ since $A \subseteq B$. As a result, we have $A/s \subseteq B/s$. □

Lemma 4.A.4 Let $A, B \subseteq \Sigma^*$. Then, $A \subseteq B \Leftrightarrow sA \subseteq sB$.

Proof: $(\Rightarrow)$: For $t \in sA$, there exists $u \in A$ such that $t = su$. Therefore, $\Rightarrow t = su \in sB$. Hence, we infer that $sA \subseteq sB$.

$(\Leftarrow)$: $t \in A$ implies $st \in sA$ and thus $st \in sB$. Hence, we have $t \in B$. Therefore, we infer that $A \subseteq B$. □

Lemma 4.A.5: Let $A \subseteq \Sigma^*$. Then, $s(A/s) \subseteq A$.

Proof: For $t \in s(A/s)$, there exists $u \in A/s$ such that $t = su$. $u \in A/s$ implies $su \in A$ and thus we have $t \in A$. Therefore, we conclude that $s(A/s) \subseteq A$. □

Lemma 4.A.6: Let $A \subseteq \Sigma^*$. $(sA)/s = A$.

Proof: Let $t \in (sA)/s$. Then, we have $st \in sA$ which implies that $t \in A$. Thus, we infer that $(sA)/s \subseteq A$. On the other hand, let $t \in A$. Then, we have $st \in sA$ which implies that $t \in (sA)/s$. Thus, we infer that $A \subseteq (sA)/s$. As a result, we conclude that $(sA)/s = A$.

□

Lemma 4.A.7: Let $A, B \subseteq \Sigma^*$. Then, $s(A \cap B) = sA \cap sB$.

Proof: $(\supseteq)$: Let $t \in sA \cap sB$. Rewrite t as su. $t \in sA \cap sB$ implies $t \in sA$ and $t \in sB$, which means $su \in sA$ and $su \in sB$. Therefore, we infer that $u \in A \cap B$ and thus $t = su \in s(A \cap B)$.

$(\subseteq)$: Let $t \in s(A \cap B)$. Rewrite t as su. $t \in s(A \cap B)$ implies $u \in A \cap B$, which means $su \in sA$ and $su \in sB$. Therefore, we infer that $t = su \in sA \cap sB$. □

Lemma 4.A.8: Let $A, B \subseteq \Sigma^*$. Then, $sA \cap B = s(A \cap B/s)$.

Proof: $(\supseteq)$: $s(A \cap B/s) = sA \cap s(B/s)$ (by Lemma 4.A.7)

$\subseteq sA \cap B$. (by Lemma 4.A.5)

$(\subseteq)$: $t \in sA \cap B \Rightarrow t \in sA \wedge t \in B$.

$\Rightarrow \exists u \in A : t = su \in B$ (Write t as su)

$\Rightarrow u \in A \wedge u \in B/s$

$\Rightarrow u \in A \cap B/s$

$\Rightarrow su \in s(A \cap B/s)$

$\Rightarrow t \in s(A \cap B/s)$.

Hence, we infer that $sA \cap B \subseteq s(A \cap B/s)$. As a result, we have $sA \cap B = s(A \cap B/s)$.

□

Lemma 4.A.9: Let $A, B \subseteq \Sigma^*$. Then, $(A - B)/s = A/s - B/s$.

Proof: $t \in (A - B)/s \Leftrightarrow st \in A - B$

$$\Leftrightarrow st \in A \wedge st \notin B$$

$$\Leftrightarrow t \in A/s \wedge t \notin B/s$$

$$\Leftrightarrow t \in A/s - B/s.$$

Hence, we infer that $(A - B)/s = A/s - B/s$. □

Lemma 4.A.10: Let $A, B, C \subseteq \Sigma^*$. Then, $A - C \subseteq (A - B) \cup (B - C)$.

Proof: $(A - B) \cup (B - C) = (A \cap B^{co}) \cup (B \cap C^{co})$

$$= [A \cup (B \cap C^{co})] \cap [B^{co} \cup (B \cap C^{co})]$$

$$\supseteq A \cap [B^{co} \cup (B \cap C^{co})]$$

$$= A \cap [(B^{co} \cup B) \cap (B^{co} \cup C^{co})]$$

$$= A \cap [\Sigma^* \cap (B^{co} \cup C^{co})]$$

$$= A \cap (B^{co} \cup C^{co})$$

$$\supseteq A \cap C^{co}$$

$$= A - C.$$ □

Lemma 4.A.11: Let $A, B, C \in \Sigma^*$. If $C \subseteq B \subseteq A$, then $A - C = (A - B) \cup (B - C)$.

Proof: $(\subseteq)$: It follows from Lemma 4.A.10.

($\supseteq$): It follows from $A-C=A\cap C^{co}\supseteq A\cap B^{co}=A-B$ and $A-C\supseteq(B-C)$. Therefore, we conclude that $A-C=(A-B)\cup(B-C)$. $\square$

Lemma 4.A.12: Let $A,B,C\in\Sigma^{*}$. Then, $A\cap(B-C)=(A\cap B)-(A\cap C)$.

Proof: First, $A\cap(B-C)=A\cap(B\cap C^{co})=A\cap B\cap C^{co}=(A\cap B)-C$. Also, $(A\cap B)-(A\cap C)=(A\cap B)\cap(A\cap C)^{co}=(A\cap B)\cap(A^{co}\cup C^{co})=[(A\cap B)\cap A^{co}]\cup[(A\cap B)\cap C^{co}]=\Phi\cup[(A\cap B)-C]=(A\cap B)-C$. Therefore, we conclude that $A\cap(B-C)=(A\cap B)-(A\cap C)$. $\square$

Chapter 5

State-based Robust Nonblocking Supervisory Control

We have solved RNSCP offline and online in Chapter 3 and 4 respectively. RNSCP takes a linguistic approach and the control domain of the resulting supervisors is based on strings. Online algorithm VLP for RNSCP works only if expansion terminates. However, this is not always the case.

In Section 5.1, we take a state-based approach and formulate State-based Robust Nonblocking Supervisory Control Problem (RNSCP-S). Its online solution eliminates the termination problem of VLP. A robust problem can be treated as RNSCP-S only if certain conditions are satisfied. This does not imply any limitation in comparison with RNSCP since an arbitrary RNSCP can be transferred to an equivalent RNSCP-S after automaton refinement. Fault recovery problem of a spacecraft propulsion system is solved as a special case of RNSCP-S manually and using TTCT in Section 5.2.

5.1 Robust Nonblocking Supervisory Control with State Information

Subsection 5.1.1 investigates the preconditions that ensure control domain can be taken as states. Mutual refinement of automata is defined there. RNSCP-S is formally formulated in Subsection 5.1.2. Fault recovery problem with illegal states is a special case of RNSCP-S. A procedure is proposed in Subsection 5.1.3 to convert an RNSCP to RNSCP-S if a robust problem initially does not satisfy the assumptions in RNSCP-S. In

Subsection 5.1.4, the state-based online algorithm is proposed and the validity of the state-based algorithm is also established.

5.1.1 Preconditions for State Control Domain

The states of automaton model of a system are usually represented by the states of system components. Hence, states have physical meaning. In addition, we can directly identify the system state by detecting the component states. Therefore, it is preferable in many situations that the domain of control law be states rather than strings.

Obviously, the control domain can not be taken as states for arbitrary given plant and specification automata. In order to be able to take states as control domain, the plant and specification automata shall meet certain criteria. We next investigate what these requirements of the plant and specification automata have to be to ensure that control domain can be states.

Like the standard state-based supervision, in a robust control problem the control policy at each state shall be the same for all strings leading to the state. This implies that the specification automaton H_i is required to be a subautomaton of the plant model G_i for all $i \in I$. Moreover, a state in one plant model shall correspond to only one state existing in another plant model or none since the control domain of VLP-S is state. Therefore, if a string belongs to both model G_i and G_j, it shall lead to the corresponding states. In the case that a string s belongs to one model G_i ($s \in L(G_i)$) but not to another model G_j ($s \notin L(G_j)$), the state of string s in G_i shall not correspond to any state of G_j.

We call the relation of plants satisfying the above requirements mutually refined. We formally define the mutually refined relation of two generators below.

Definition 5.1: Mutually Refined Generator Relation Consider two generators $R_1 = (X_1, \Sigma_1, \delta_1, x_{01})$ and $R_2 = (X_2, \Sigma_2, \delta_2, x_{02})$. Assume that there exists a one-to-one relation between a subset of X_1 and a subset of X_2. For simplicity, we assume the corresponding states in the aforementioned subsets have the same labels. We say R_1 and R_2 are **mutually refined** if $\delta_1(x_{01}, s) = \delta_2(x_{02}, s)$ in case $s \in L(R_1) \cap L(R_2)$, $\delta_1(x_{01}, s) \neq \delta_2(x_{02}, t)$ in case $s \in L(R_1) - L(R_2)$ for all $t \in L(R_2)$, and $\delta_1(x_{01}, t) \neq \delta_2(x_{02}, s)$ in case $s \in L(R_2) - L(R_1)$ for all $t \in L(R_1)$. □

Lemma 5.1: Consider two generators $R_1 = (X_1, \Sigma_1, \delta_1, x_{01})$ and $R_2 = (X_2, \Sigma_2, \delta_2, x_{02})$ with $L(R_1) \subseteq L(R_2)$. If they are mutually refined, then $R_1 \subseteq R_2$.

Proof: For any string $s \in L(R_1)$, we have $s \in L(R_2)$ since $L(R_1) \subseteq L(R_2)$. This implies that $\delta_1(x_{01}, s) = \delta_2(x_{02}, s)$ since R_1 and R_2 are mutually refined. Hence we have $R_1 \subseteq R_2$ by the definition of subgenerators. □

The inverse implication of Lemma 5.1 does not hold. Namely, even if a generator is a subgenerator of another generator, they may not satisfy the mutually refined generator relation. For two generator G_1 and G_2 with $L(G_1) \subseteq L(G_2)$, mutually refined relation is stronger than the subgenerator relation. The following example illustrates this.

Example 5.1: Given two generators G_1 and G_2 shown in Figure 5.1, we verify that $G_2 \subseteq G_1$, but G_1 and G_2 are not mutually refined.

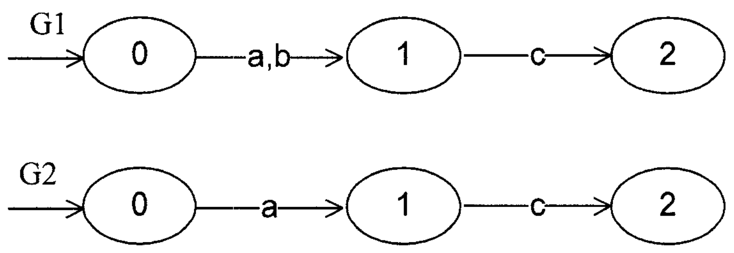

Figure 5. 1: Plant automata for Example 5.1

The corresponding states in G_1 and G_2 are marked with the same number. Any string in G_2 leads to the corresponding state in G_1. For example, string *ac* leads to state 2 in G_2 and the state 2 in G_1. Hence, $G_2 \subseteq G_1$.

Since string b does not exist in G_2, it shall not lead to any state in G_1 that has a corresponding state in G_2. However, string b leads to state 1 in G_1 which corresponds to state 1 in G_2. Therefore, the condition of mutually refined relation is not satisfied. □

We next formally give the preconditions ensuring that the control domain of a robust control problem can be taken as states.

PC: Preconditions of State Control Domain:

PC1) For all $i, j \in I$, G_i and G_j are mutually refined.

PC2) For all $i \in I$, $H_i \subseteq G_i$. □

The following example shows the necessity of mutual refinement of plant automata for state-based control domain.

Example 5.2: Two possible plant models of a system are represented by automata G_1 and G_2 in Figure 5.2. The specification automata for G_1 and G_2 are H_1 and H_2 respectively in Figure 5.2. We examine if the control domain can be taken as states.

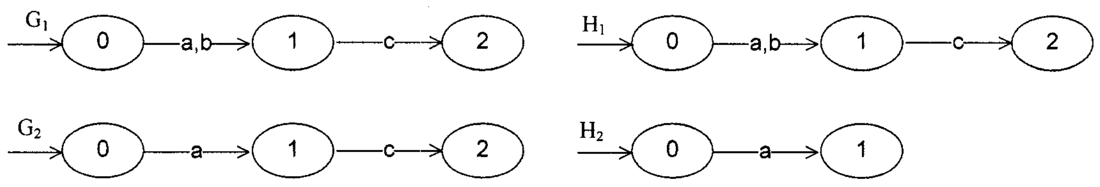

Figure 5. 2: Plant and specification automata for Example 5.2

Specification automata are subgraphs of plant automata since $H_1 \subseteq G_1$ and $H_2 \subseteq G_2$. In addition, one state of G_1 corresponds to only one state of G_2. All common strings of G_1 and G_2 lead to the corresponding states in G_1 and G_2. However, string b is not in G_2 but leads to state 1 that exists in G_2. Both strings a and b lead to state 1. The control policy of string a and b are different since $\gamma(a) = \{\varepsilon\}$ and $\gamma(b) = \{c\}$; thus the control policy of state 1 depends on which string leads to it. Therefore, the problem can not be solved as a state-based supervisory control problem. □

Theorem 5.1 provides a formal proof that the preconditions PC will ensure that control domain can be taken as states.

Theorem 5.1: The control domain can be taken as states if the preconditions PC are satisfied.

Proof: For all strings denoted as $[x]$ leading to a given state x in one plant model G_i, they shall also exist in another plant model G_j and lead to state x if state x exists in G_j. In addition, there is no other string leading to state x in G_j. Therefore, the post behaviors of all strings $[x]$ in plant model G_j are the same. The specifications of all the strings $[x]$ are also the same since the specification is a subautomaton of the plant. Hence, the control policy for all strings $[x]$ leading to the given state x will be the same. As a result, the domain of supervisory control map can be taken to be the state set. □

5.1.2 Problem Formulation

In many circumstances, the control target is to restrict the system inside the desirable states. In other words, we would like to prevent a system from entering illegal states. In this case, specification automaton H_i can be obtained by keeping only the legal states of the corresponding plant model G_i and trimming the resulting automaton. Sate-based robust nonblocking supervisory control problem is formulated as follows.

State-Based Robust Nonblocking Supervisory Control Problem: (RNSCP-S)

Given a set of plant automata G_i with $i \in I$, assume they are mutually refined and the specification automaton H_i is represented by an automaton obtained after deleting illegal states and trimming the resulting automaton. The state $X(G_i)$ of plant G_i is required to be restricted into its legal states $X(H_i)$ for all $i \in I$. Synthesize an optimal admissible robust nonblocking supervisor, which solves the problem. □

In RNSCP-S, the plants are assumed mutually refined and thus meet PC1. In addition, specifications are in terms of legal states and thus meet PC2. Therefore, the control domain of RNSCP-S can be taken as states.

Remark 5.1: We assume that specifications are given as legal states in RNSCP-S which implies that the specifications are relative-closed. □

Remark 5.2: In order to make control domain be states, specification is only required to be a subautomaton of the corresponding plant $H_i \subseteq G_i$, which means the specification could be in terms of both illegal states and illegal transitions between some legal states. In contrast, RNSCP-S requires that the specification is given as only illegal states, which is a stronger requirement than $H_i \subseteq G_i$ in that in RNSCP-S $H_i \subseteq G_i$ holds and all transitions between legal states are always desirable. □

Remark 5.3: Fault recovery problems with one permanent failure at a time and specification given as illegal states can be directly solved as RNSCP-S without automaton refinement. Plant generators are mutually refined inherently since the normal states are included in each plant model, and the failure states after different failures are disjointed. □

We can treat RNSCP-S as a special case of RNSCP. However, this by no means implies that RNSCP-S has any limitation. As a matter of fact, a problem with arbitrary plants and specification automata can be converted to a corresponding state-based problem after automaton refinement. The next section develops a procedure of transferring an arbitrary RNSCP problem to a corresponding RNSCP-S problem.

5.1.3 Converting RNSCP to RNSCP-S

The specification in RNSCP may not be relative-closed. However, we assume that specification is relative-closed in RNSCP without loss of generality since we can always obtain the supremal relative closed sublanguage of the specification otherwise.

The automata in RNSCP usually do not satisfy the two assumptions of RNSCP-S. Nevertheless, we can always refine plants and specification automata to solve RNSCP as an equivalent RNSCP-S problem. We first provide and prove a sufficient condition which ensures the assumptions of RNSCP-S. After that, we propose a procedure to ensure the above-mentioned sufficient condition.

SC: Sufficient condition for RNSCP-S: Any two generators in $\{G_1,...,G_n,H_1,...,H_n\}$ are mutually refined. □

Lemma 5.2: Consider a specification generator H and a plant generator G with $L(H) \subseteq L(G)$. If H and G are mutually refined, then specification will be in terms of legal states.

Proof: Follows from the definition of mutual refinement. All states in G that have corresponding states in H are legal. All events leading from a legal state to another legal state of G are allowed by H. H can be obtained from G after removing the states from G that have no corresponding states in H. As a result, the specification is in terms of legal states. □

Theorem 5.2: The sufficient condition SC guarantees that an RNSCP problem can be solved as an RNSCP-S problem.

Proof: Specification is in terms of legal states by Lemma 5.2. In addition, all plant automata are mutually refined. Therefore, the problem can be treated as RNSCP-S. □

In case that the sufficient condition SC is not satisfied in a given RNSCP problem, all plant and specification automata can be refined to ensure it. A procedure is required to convert RNSCP to RNSCP-S. Unlike standard supervisory control, robust supervisory control deals with a set of plants and specifications.

We first generalize the biased synchronous production of two generators [15] [6] to more generators before we formally propose a procedure to guarantee the sufficient condition. In the standard biased synchronous production $||_r$, one generator leads another. In the generalized operation $||_{mr}$, one generator leads a set of other generators.

Definition 5.2: (Multiple Biased Synchronous Product $||_{mr}$)

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: $R_k \,||_{mr}\, (R - \{R_k\}) :=$ $Ac(X_1 \times ... \times X_2, \Sigma_k, \delta, (x_{01}, ..., x_{0n}), X_1 \times ... \times X_{k-1} \times X_{mk} \times X_{k+1}, ... \times X_n)$ where

$$\delta((x_1, ..., x_n), \sigma) = \begin{cases} (x_1', ..., x_n') & \text{if } \sigma \in \Sigma_{R_k}(x_k) \\ \text{undefined} & \text{otherwise} \end{cases} \text{ and } x_i' = \begin{cases} \delta_i(x_i, \sigma) & \sigma \in \Sigma_{R_i}(x_i) \\ x_i & \sigma \notin \Sigma_{R_i}(x_i) \end{cases}. \; \square$$

The operation of multiple biased synchronous production of R_k does not alter its generated or marked languages.

Lemma 5.3: $L(R_k \|_{mr} (R-\{R_k\})) = L(R_k)$ and $L_m(R_k \|_{mr} (R-\{R_k\})) = L_m(R_k)$.

Proof: Follows from the definition of multiple biased synchronous production $\|_{mr}$. □

We are now ready to present the following procedure which converts RNSCP to RNSCP-S. Consider a set of arbitrary plant automata $(G_1, G_2, ..., G_n)$ and corresponding specification automata $(H_1, H_2, ..., H_n)$ and assume that the specifications are relative-closed with respect to the corresponding plant. The following procedure returns automata $(G_1'', G_2'', ..., G_n'')$ and $(H_1'', H_2'', ..., H_n'')$ which generates and marks the same languages and satisfies the sufficient condition SC for RNSCP-S.

Procedure 5.1:

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 of $\Sigma - \Sigma_{R_i}(x)$ 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 $G_i'' = G_i \|_{mr} (R' - \{G_i\})$ for all $i \in \{1, ..., n\}$.

3) Replace H_i' in R' with H_i, and derive $H_i'' = H_i \|_{mr} (R' - \{H_i\})$ for all $i \in (1, ..., n)$. □

Assume the number of states of each plant and specification automaton is *m*. Then, the number of states of any resulting generator of the above procedure will be at most $(m+1)^{2n}$.

Theorem 5.3: Let $(G_1'', G_2'', ..., G_n'')$ and $(H_1'', H_2'', ..., H_n'')$ be the resulting automata from the above procedure. Then, the languages of plants and specifications are unchanged, and any two of resulting automata are mutually refined. Therefore, the sufficient condition SC holds and the resulting problem can be solved as an RNSCP-S.

Proof: Rename $(G_1, ..., G_n, H_1, ..., H_n)$ as $(R_1, R_2, ..., R_{2n})$, $(G_1', ..., G_n', H_1', ..., H_n')$ as $(R_1', R_2', ..., R_{2n}')$, and $(G_1'', ..., G_n'', H_1'', ..., H_n'')$ as $(R_1'', R_2'', ..., R_{2n}'')$. We have $L(R_i'') = L(R_i)$ and $L_m(R_i'') = L_m(R_i)$ by Lemma 5.3. This implies that languages of plants and specifications remain the same after the procedure.

We denote the dump state of R_i' as x_{id}. For any string $r \notin L(R_k) = L(R_k'')$, we have $\delta_k(x_{0k}, r) = x_{kd}$. For any string $r \in L(R_k) = L(R_k'')$, we have $\delta_k(x_{0k}, r) \neq x_{kd}$.

We next show any two resulting automata R_i'' and R_j'' are mutually refined. Denote the state of a string *s* in R_i'' as $(x_{1s}, x_{2s}, ..., x_{2ns})$ where $x_{is} = \delta_i(x_{0i}, s)$.

Case 1: $s \in L(R_j'')$.

We only need to show *s* leads to the corresponding state $(x_{1s}, x_{2s}, ..., x_{2ns})$ in R_j''. This is true by the definition of $||_{mr}$.

Case 2: $s \notin L(R_j'')$.

We need to show that any string t in R_j'' will not lead to the state $(x_{1s}, x_{2s}, \ldots, x_{2ns})$. It is obvious that $x_{js} = x_{jd}$ by definition of $\|_{mr}$. Denote the state of t in R_j'' as $(x_{1t}, x_{2t}, \ldots, x_{2nt})$. It is straightforward that $x_{jt} \neq x_{jd}$ by definition. Hence, we have $x_{js} \neq x_{jt}$ and thus $(x_{1s}, x_{2s}, \ldots, x_{2ns}) \neq (x_{1t}, x_{2t}, \ldots, x_{2nt})$.

We conclude that R_i'' and R_j'' are mutually refined from the above two cases. Thus, the sufficient condition SC of RNSCP-S is satisfied. This implies that the problem can be solved as RNSCP-S after automaton refinement using Procedure 5.1. □

The following example illustrates the above procedure.

Example 5.3: Two possible plant models of a system are represented by automata G_1 and G_2 in Figure 5.3. The specification automata for G_1 and G_2 are H_1 and H_2 respectively in Figure 5.3. We apply Procedure 5.1 to refine them and verify that the resulting generators meet the sufficient condition SC and that the specifications are in terms of legal states.

First, we add a dump state to each generator and add transition of inactive events from all states to the dump state. Self-loops of all events are added to each dump state. The resulting generators are shown in Figure 5.4.

Then, we take the multiple biased synchronous productions for each plant generators and specification automata. The resulting generators are displayed in Figure 5.5.

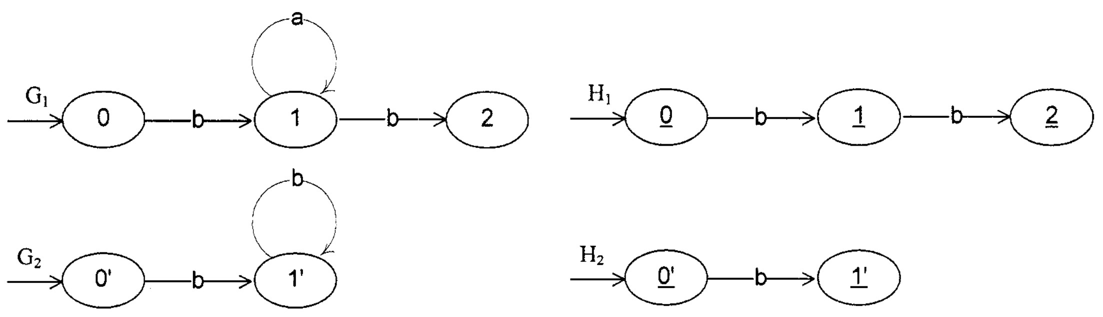

Figure 5. 3: Given plant and specification automata: Example 5.3

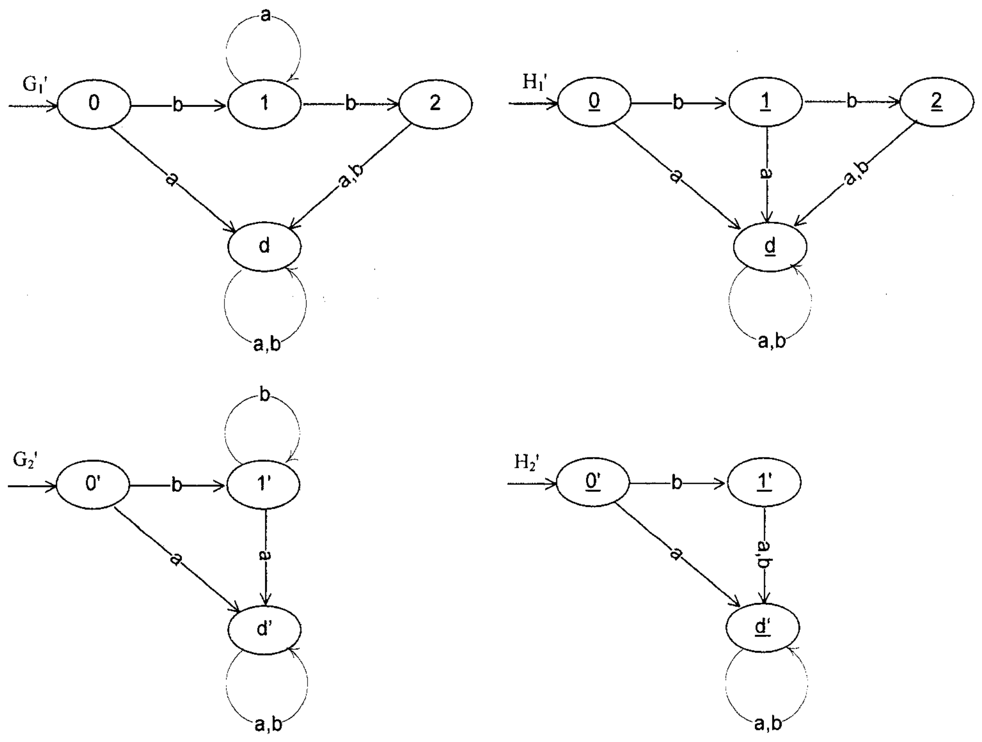

Figure 5. 4: Modified automata: Example 5.3

The states with the same name are the corresponding states in each generator. We can verify that the resulting generators have the same languages as those of the original generators, and are mutually refined with each other. Moreover, the illegal states of G_1'' are $1d'\underline{dd'}$ and $2d'\underline{dd'}$, and the illegal states of G_2'' are $21'\underline{2d'}$ and $d1'\underline{dd'}$. □

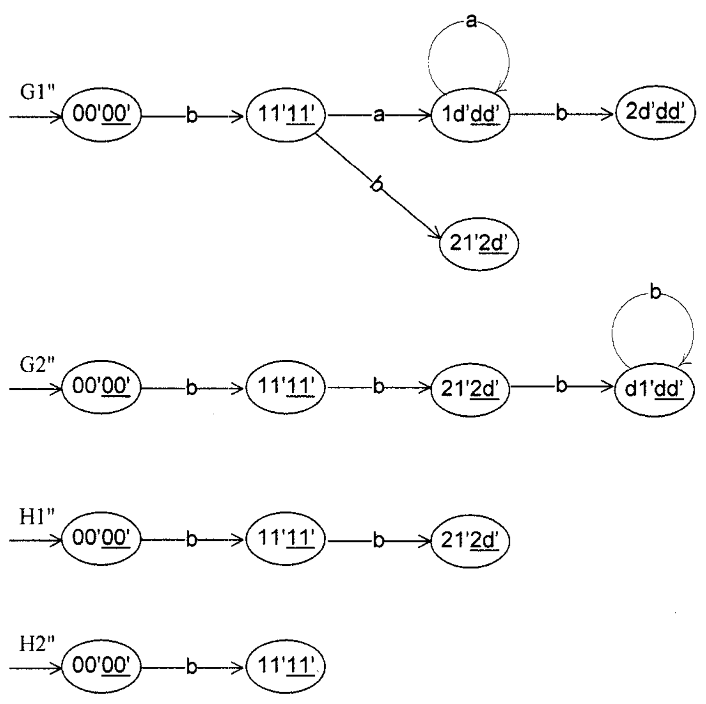

Figure 5. 5: Resulting automata: Example 5.3

5.1.4 Solution to RNSCP-S

RNSCP takes a linguistic approach. Its corresponding online VLP algorithm works only if the expansion terminates. For example, the expansion of VLP (Algorithm 4.1) for some strings may not terminate if N' is infinite or undefined. Thus, VLP is not always applicable.

RNSCP-S takes state-based approach. It has the advantage over RNSCP in that its corresponding online algorithm always works since the expansion always terminates. The state-based online algorithm eliminates the expansion termination problem of VLP by using state information.

Offline solution to RNSCP-S is exactly the same as solving RNSCP offline. Like RNSCP, offline solution to RNSCP-S can take either the indirect approach or the direct approach.

Here, we propose an online direct solution to RNSCP-S. Unlike VLP, the state-based online algorithm expands the plant graphs directly as subgraphs rather than tree generators. We first investigate the expansion stop rules and then derive an algorithm for state-based online robust supervision.

We denote states in plant model G_i as $X(G_i)$ and marked states in G_i as $X_m(G_i)$. Similarly, we denote states in specification H_i as $X(H_i)$ and marked states in H_i as $X_m(H_i)$. Under the assumption, a state in one automaton will correspond to a sole state in another. Therefore, we refer to a state without specifying a plant or specification automaton. The overall states form a state set X. A state in X either refers to a corresponding state in an automaton or does not exist in the automaton. However, a state in X refers to a state in at least one plant automaton.

A state in X that is illegal in one model may be legal in another model. A state in X marked in one possible model may not be marked in another possible plant model. A marked state in X with only controllable events in one model may be a marked state with uncontrollable continuation in another model.

The definitions of illegal states, legal marked states, legal marked controllable states, and legal marked uncontrollable states are different from the standard (non-robust) case. They are formally defined below.

Definition 5.3: Illegal states X_{fc} consist of states that are illegal in some possible plant model: $X_{fc} := \bigcup_{i \in I}(X(G_i) - X(H_i))$. □

With the assumptions of RNSCP-S, a state in H_i will be marked in H_i if it is marked in G_i. Note that all states marked in H_i will exist and be marked in G_i in RNSCP-S.

Definition 5.4: Legal marked states X_m consist of states that are legal and marked in all plant models that contain them:

$X_m := \{x \in \bigcup_{j \in I} X_m(H_j) : \forall i \in I, x \in X(G_i) \Rightarrow x \in X_m(H_i)\}$. □

Definition 5.5: Legal marked controllable states X_{mc} consist of states that are legal and marked and have only controllable active event in all plant models that contain them: $X_{mc} := \{x \in X_m : \forall i \in I, x \in X(G_i) \Rightarrow \Sigma_{G_i}(x) \subseteq \Sigma_c\}$. □

Definition 5.6: Legal marked uncontrollable states X_{mu} consist of states that are legal marked and have uncontrollable active event in some plant model that contains them: $X_{mu} := \{x \in X_m : \exists i \in I, x \in X(G_i) \wedge \Sigma_{G_i}(x) \cap \Sigma_u \neq \Phi\}$. □

Obviously, we have $X_m = X_{mc} \dot{\cup} X_{mu}$ where $\dot{\cup}$ denotes disjoint union. We are now ready to formally present the expansion rules.

Expansion Rules:

1) All plant models are expanded simultaneously as subgraphs.

2) All active events of the current state are expanded.

3) Stop expanding a state if it was previously expanded.

4) Do not expand a state if it is an illegal state in X_{fc}.

5) Do not expand a state if it is a legal marked controllable state in X_{mc}.

6) Expand only the uncontrollable events of a state if it is a legal marked uncontrollable state in X_{mu}.

7) Expand all events if a state is not in $X_{fc} \cup X_m$. □

The expansion of VLP is a tree generator. In order to make VLP applicable, expansion shall be a finite tree and thus finite language. However, the expansion of VLP may not terminate in some problems.

In contrast, the expansion of VLP-S is a subgraph of plant automaton and thus it can represent a language with infinite number of strings. In addition, the expansion of VLP-S always terminates since each plant model is represented by a finite-state automaton.

If we allow infinite tree in VLP, the expansion of VLP can also represent infinite language. We would like to compare it with the language of VLP-S expansion for a given current string. The only difference is that the controllable events of marked strings other than the current string but leading to the current state are expanded in VLP-S but not expanded in VLP if the current string is marked. In this case, the language of VLP expansion will be a sublanguage of VLP-S expansion. Otherwise, if the current string is not marked, then the expansion of VLP and VLP-S will be the same.

The following example illustrates the expansion using the above stop rules.

Example 5.4: Given two mutually refined plant generators G_1 and G_2 with $\Sigma_c = \{a,b,c,d,e\}$, $\Sigma_u = \{u,v,w\}$, and illegal states 1 and 7 shown in Figure 5.6, we obtain the expansion for the initial state using the above expansion rules.

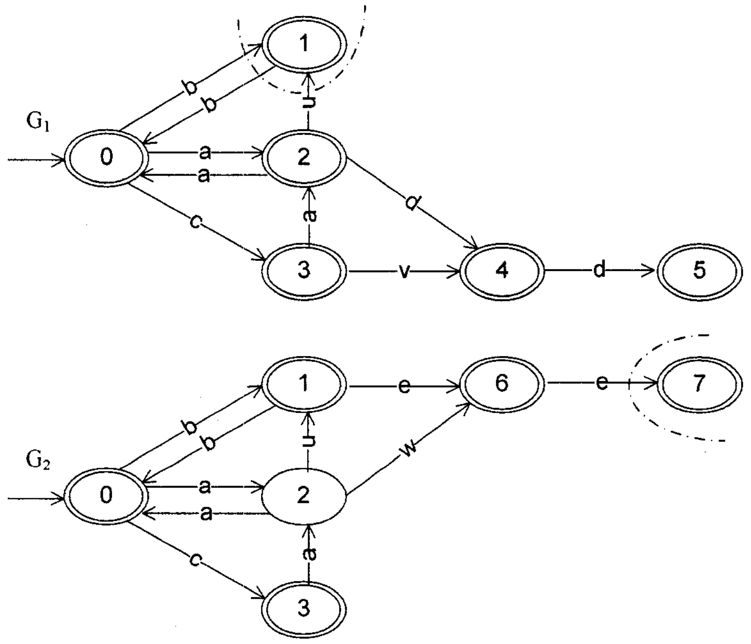

Figure 5. 6: Plant automata for Example 5.4

The state set X is $\{0,1,\ldots,7\}$. All events of the current state 0 are expanded and thus states 1, 2, and 3 are included in the expansion. The active events b and e of state 1 are unexpanded since state 1 is an illegal state. State 2 is a state that is not in $X_{fc} \cup X_m$ and thus all its active events are expanded. Event a and u leads to existing state 0 and 1 respectively. Event d and w lead to states 4 and 6 respectively. State 3 is a state in X_{mu}

and thus only uncontrollable event v is expanded and the existing state 4 is reached. No expansion is conducted at states 4 and 6 since they are states in X_{mc}. The resulting expansion for the initial state is shown in Figure 5.7. □

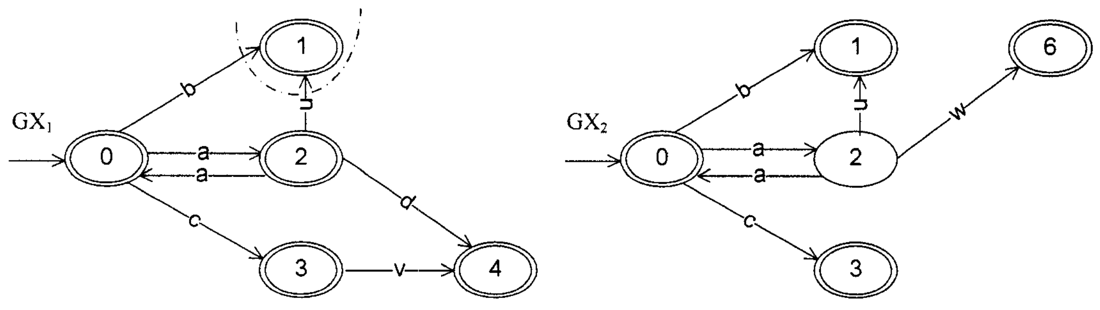

Figure 5. 7: Expansion for the initial state: Example 5.4

Alternatively, the expansion can be obtained directly from the following procedure.

Procedure 5.2:

1) Do not delete any active event of the current state.

2) Delete all active events of the illegal states in X_{fc}.

3) Delete all controllable events of the marked states in X_m.

4) For each model, take the current state as initial state and obtain the reachable part.

□

The following algorithm VLP-S employs state-based variable-lookahead policy to derive supervisors for RNSCP-S. VLP control decision is in terms of the current string. In contrast, VLP-S control decision is derived for the current state.

Algorithm 5.1: (VLP-S)

1) Initialization: $x = x_0$.

2) Expand each model to obtain GX_i for all $i \in I$ at current state x according to the expansion rules.

3) Keep the legal states of GX_i to obtain specification HX_i for all $i \in I$.

4) Let $EX_i = L_m(HX_i)$ for all $i \in I$.

5) Calculate $(T_1',...,T_n') = SupRCS((EX_1,...,EX_n))$.

6) Derive control policy $\hat{\gamma}(x) = (\bigcup_{i \in I} \overline{T_i'}) \cap \Sigma$.

7) Wait until an enabled event σ happens.

8) Replace x with $\delta(x,\sigma)$, and go back to step 2. □

$(T_1',...,T_n') = SupRCS((T_{1,0}',...,T_{n,0}'))$ in the above algorithm is the largest fixed-point of the operator $\underline{\Omega}((K_1,...,K_n)) = SupS(SupC(SupR((K_1,...,K_n))))$ and can be computed iteratively. In RNSCP-S, all strings leading to a state have exactly the same post behavior and specification. The following example illustrates the above algorithm.

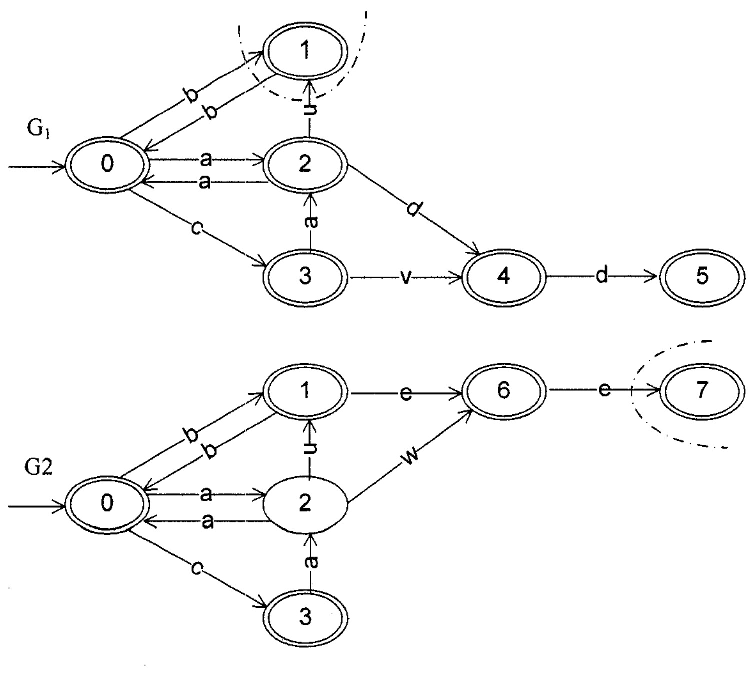

Figure 5. 8: Plant automata for Example 5.5

Example 5.5: The above VLP-S algorithm is employed to solve the same problem in Example 5.4. Given two mutually refined plant generators G_1 and G_2 with $\Sigma_c = \{a,b,c,d,e\}$, $\Sigma_u = \{u,v,w\}$, and illegal states 1 and 7 shown in Figure 5.8, we derive control decision for the initial state 0 using the above algorithm.

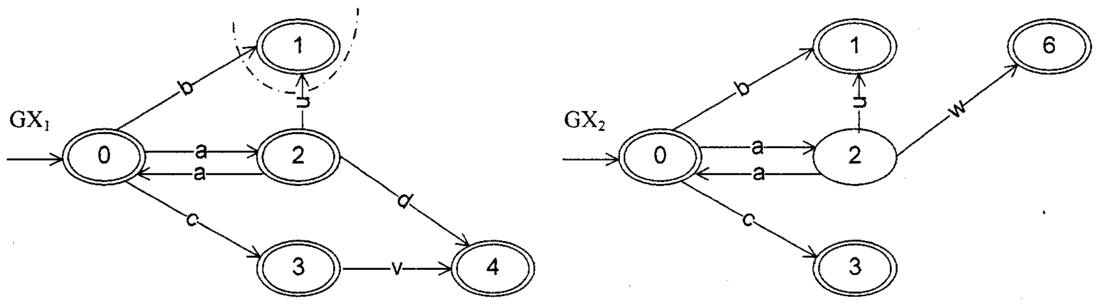

Figure 5. 9: Expansion for the initial state: Example 5.5

The expansions GX_1 and GX_2 for the initial state 0 are shown in Figure 5.9. They are expanded the same way as in Example 5.4. The specifications EX_1 and EX_2 for the expansions are represented by automata HX_1 and HX_2 in Figure 5.10. We then employ the Algorithm 3.2A to iteratively calculate $(T_1', T_2') = SupRCS((EX_1, EX_2))$.

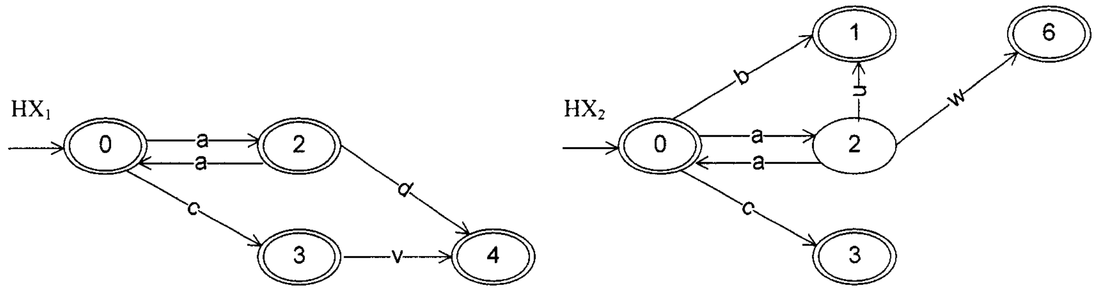

Figure 5. 10: Specification automata for the expansion: Example 5.5

Observe that EX_1 and EX_2 are relative-closed with respect to $L_m(GX_1)$ and $L_m(GX_1)$ respectively. Initially, we have $T_{1,0}' = L_m(HX_1)$ and $T_{2,0}' = L_m(HX_2)$. Then, $(T_{1,j+1}', T_{2,j+1}') = SupS(SupC((T_{1,j}', T_{2,j}')))$ is computed iteratively.

$T_{1,0}'$ is not controllable with respect to $L(GX_1)$ since uncontrollable event u leads state 2 to illegal state 1. The supremal controllable sublanguage $SupC(T_{1,0}')$ is represented by the automaton *HX1-1C* in Figure 5.11 and is obtained from HX_1 with state 2 removed. $T_{2,0}'$ is controllable with respect to $L(GX_1)$ and thus $SupC(T_{2,0}')$ is represented by the automaton *HX2-1C* in Figure 5.11. This automaton is the same as HX_2.

$SupC(T_{1,0}')$ and $SupC(T_{2,0}')$ are not consistent since states 1 and 2 are disabled in expansion GX_1 but enabled in expansion GX_2 . The result after the first iteration

$(T_{1,1}^{'},T_{2,1}^{'}) = SupS(SupC((T_{1,0}^{'},T_{2,0}^{'})))$ is represented by generator *HX1-2* and *HX2-2* shown in Figure 5.12.

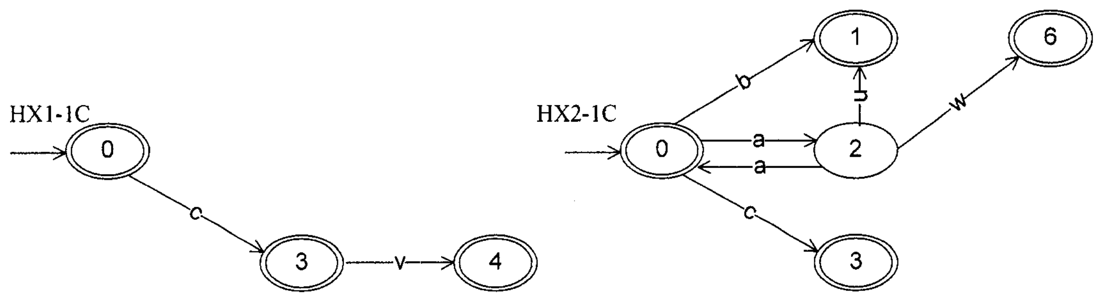

Figure 5. 11: Controllable automata of the first iteration: Example 5.5

There is no change in the second iteration and thus the algorithm terminates with the supremal element $(T_1',T_2') = SupRCS((T_{1,0}^{'},T_{2,0}^{'})) = (T_{1,1}^{'},T_{2,1}^{'})$. $T_1^{'}$ and $T_2^{'}$ are represented by generator *HX1-2* and *HX2-2*. We can verify that they are relative-closed, controllable, and consistent. The control policy for the initial state 0 will be $\hat{\gamma}(0) = \{c\}$ from the results $T_1^{'}$ and $T_2^{'}$. We can verify that $T_1^{'}$ and $T_2^{'}$ are computed by keeping removing states at each iteration from the initial generators HX_1 and HX_2. □

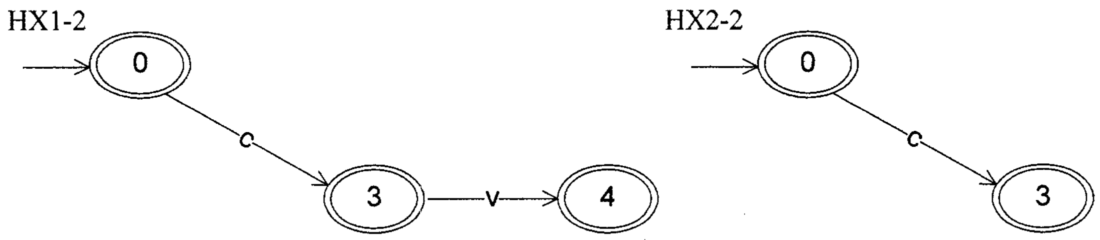

Figure 5. 12: Resulting automata of the first iteration: Example 5.5

The proof of validity of VLP algorithm in Chapter 4 can be used for validity proof of VLP-S algorithm after replacing the current string s with current state x. We do not reiterate the proof procedure. Instead, we provide intuitive explanation why VLP's proof works for VLP-S.

The key to the validity of VLP is the expansion rules which ensure that the unexpanded part always starts from an event at an illegal boundary string or a controllable event at a legal marked boundary string. This is also true in VLP-S since expansion of VLP-S stops only at all active events of illegal states and all controllable events of marked states. In addition, no property of tree generator is used in the validity proof of VLP. Moreover, the validity proof of VLP still holds if expansion tree is infinite though we assume expansion of VLP always terminates as a finite tree.

Remark 5.4: All active events from an illegal state will lead to illegal states in RNSCP-S. Therefore, no active events of illegal strings will be expanded in VLP-S. This is similar to the expansion in VLP that all active events from an illegal string are not expanded. If we relax illegal state specification to subautomaton specification in RNSCP-S, then active events at illegal strings that lead to the legal states may be expanded in VLP-S. □

VLP-S is directly applicable to some situations such as fault recovery with illegal state specifications where the assumptions are naturally satisfied. All failure modes share a common normal behavior and their failure states are disjoint. Since specification is given by illegal states, the specification of each model is obtained by simply removing the illegal states in the corresponding plant mode.

Unlike VLP, the advantage of VLP-S is that the expansion of VLP-S algorithm always terminates and thus it is applicable for the most general situations. If the assumption of RNSCP-S does not hold in some problems, the plants and specification automata can be refined to guarantee the sufficient condition SC. Then, VLP-S can be employed to solve the problems.

Fault recovery problems with specification given as illegal states automatically satisfy the preconditions of RNSCP-S. We apply solution to RNSCP-S to solve a fault recovery problem offline and online manually and using TTCT in the next section.

5.2 Application to Fault Recovery of A Spacecraft Propulsion System

In this section, state-based robust supervision is used to solve fault recovery problem of a simplified spacecraft propulsion system [35] [36]. Spacecraft propulsion systems are first introduced to give background knowledge in Subsection 5.2.1. Then, a simplified propulsion system is introduced and its DES model is given in Section 5.2.2. After that, the problem is formulated in Subsection 5.2.3 and solved with both offline and online direct approaches manually in Subsection 5.2.4 and using TTCT in Subsection 5.2.5 to illustrate the procedure of supervisor synthesis.

5.2.1 Introduction

Rocket propulsion [17] is to gain momentum by expelling mass stored in the vehicle itself. These systems can be divided to spacecraft propulsion systems and booster systems. Spacecraft engines operate in space environment and thus are different from air-breathing aircraft jet engines. Characteristics of the environment of spacecraft are high vacuum, thermal radiation, nuclear radiation, meteoric bombardment, and absence of gravity.

Launch Vehicle (Booster) requires large momentum continuously to overcome the gravity of a plane. In contrast, spacecraft works in the vacuum and zero gravity environment; therefore, the propulsion system is relatively small and may only work intermittently. The spacecraft engines can be grouped into **main engines** and **reaction control engines** [17].

The source of energy of propulsion system could be chemical (liquid or solid) or electrical. Chemical propulsion is the most widely used type since it provides high thrust. Typical chemical propulsion systems [4] include **cold gas systems**, **monopropellant systems**, **bipropellant systems**, and **solid propulsion systems**.

A monopropellant system has only fuel [4]. It is simple and reliable. The commonly used fuel for monopropellant systems is Hydrazine. Energy is released when hydrazine decomposes in the combustion chamber. A bipropellant system has both fuel and oxidizer. Thrust is generated when they are mixed and combusted inside the chamber. The bipropellant system is suitable for larger thrust generation. The propellants for bipropellant systems are usually liquid Hydrogen and Oxygen.

Propulsion systems [17] can also be divided into **pressurized feed** and **pump-feed systems**. A typical pressurized propulsion system consists of propellant tanks, propellant feed systems, and thrust chamber. High-pressure gas such as Helium provides required pressure to the propellant. Pump-feed systems require pumps, turbines, and sometimes gas generators. They can provide large propellant flow rates and thus high thrust.

A simple pressurized propellant feed system includes high-pressure gas, pipes, and valves. Different kinds of valves are used for different purposes. **Propellant valves** [13] control flow of propellants to main thrust chamber, gas generators, or preburners. **Servovalves** [13] are electrohydraulic devices that control fluid flow or pressure in response to an electrical signal. **Pilot valves** [13] are used to control a fluid, which in turn controls or actuates other fluid-flow-contorl components. **Check valves** [17] enable flow of one direction and disable the other. **Pyrovalves** [17] can only operate once. It can seal the propellant tightly for a long time.

Spacecraft main propulsion motors [17] are used in the missions such as orbit insertion, retrofiring prior to a planetary landing, ascending from the surface of a planet, plane change, orbital maneuver, and Hohmann transfers.

Typical applications of reaction control motors (thrusters) [17] include attitude control, thrust vector control, station keeping, spin-up and spin-down of spin-stabilized spacecraft, midcourse corrections on deep-space mission, on-orbit drift, on-orbit phase change, gravity turn, and rendezvous maneuvers.

The main engine controls [17] include mixture ratio control, propellant utilization control, and thrust control. The secondary controls are composed of tank pressurization controls, safety control, duration control of engine main stage, engine-system control calibration, and start-up and shut-down control. The main operations are start-up and cut-off engines. The common secondary operations are purging, chill-down before start engine.

There are many motors in one spacecraft [4]. They need to be operated according to some requirements. For example, some motors need to be fired simultaneously in order to generate torque. On the other hands, some motors must not be fired at the same time to avoid canceling their thrust.

There are numerous failures in spacecraft [13]. The common failures are valve stuck-closed, stuck-open, leakage of pipe, sensor failure, combustion chamber failure. We require the system (propulsion system) under supervision to meet design specification and to be nonblocking despite failures. Namely, design specifications are required to be met even when failure happens.

5.2.2 Modeling of A Simple Propulsion System and its Specification

Robustness of spacecraft is critical since they are required to run for a long time with little attention of human being. Propulsion systems are one of the most important subsystems on spacecraft. Spacecraft face numerous failures which make offline implementation of robust supervisory control almost impossible due to computational complexity. We would like to apply our online robust nonblocking supervisory control algorithms to solve the fault recovery problem of a simplified spacecraft propulsion system.

We consider a monopropellant rocket with two engines. The schematic diagram of the system is shown in Figure 5.13. It consists of a fuel tank, three valves, fuel delivery pipes, and two combustion chambers. V_1 and V_2 are pyrovalves. They prevent fuel leakage during the long storage period. Once V_1 and V_2 are opened, they will remain open unless they fail (stuck-closed).

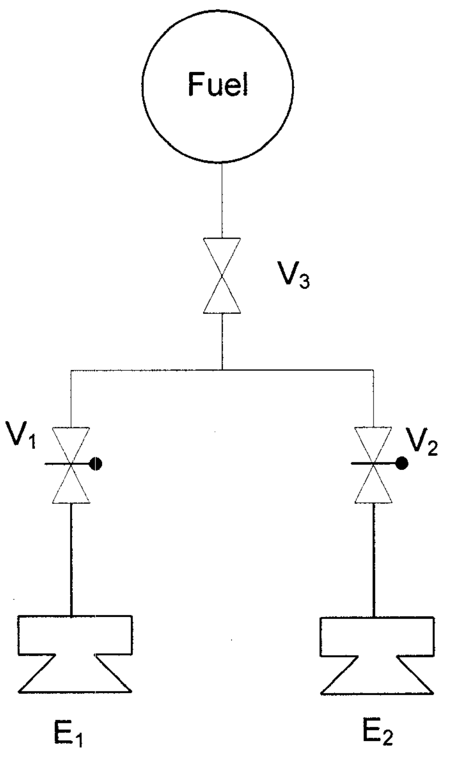

Figure 5. 13: A propulsion system

Table 5.1: Event list of the propulsion system

Event label	**Event**	**Event type**
a_i	Open Valve *i*	Controllable
b_3	Close Valve 3	Controllable
f	Valve 1 stuck-closed	Uncontrollable
u_i	Engine *i* Thrust up	Uncontrollable
d_i	Engine *i* Thrust down	Uncontrollable

For simplicity, we only consider one failure mode which is Valve 1 stuck-closed when the valve is open. Assume the fuel tank always has sufficient fuel.

The possible states of components are: Valve 1={0(closed), 1(open), 2(stuck-closed)}, Valve 2, Valve 3 ={0(closed),1(open)}, and Engine={0,1,2,3}. Events and their properties are listed in Table 5.1.

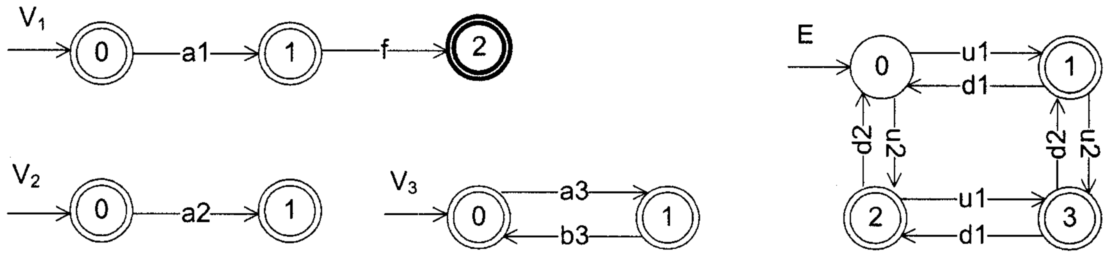

Figure 5. 14: Automata of components

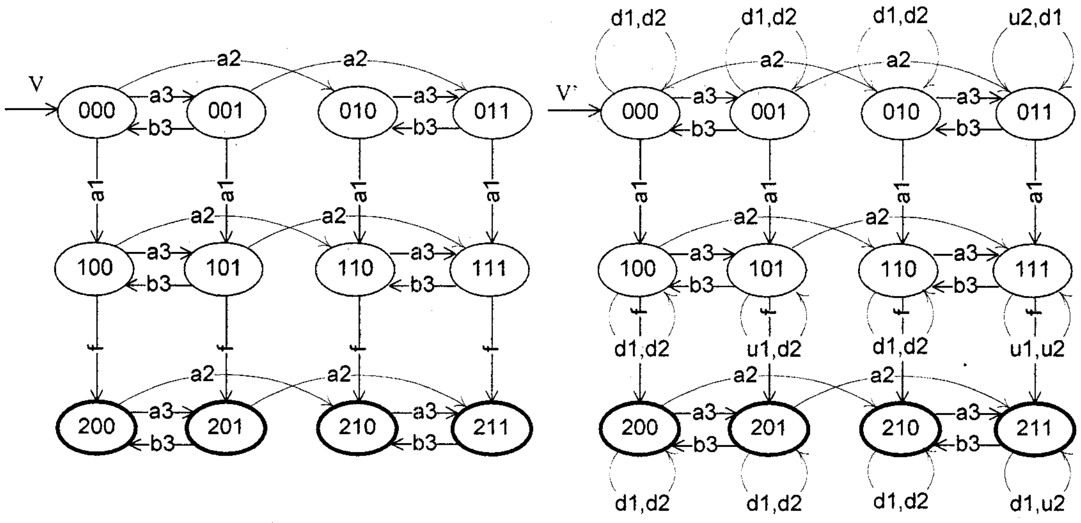

Figure 5. 15: Combined automaton of the three valves

Automata of components are shown in Figure 5.14. In the model V_1, we only consider failure *f* at state 1 for simplicity. The two combustion chambers are modeled as a whole. We mark all states of valves and only the states of the combustion chambers where at least one engine generates thrust. At state 0 of *E*, both Engine 1 and Engine 2 are off. Only Engine 1 is fired at state 1 and only Engine 2 is on at state 2. At state 3, both Engine 1 and Engine 2 are fired. The marked states are shown with double circles. Failure states are highlighted.

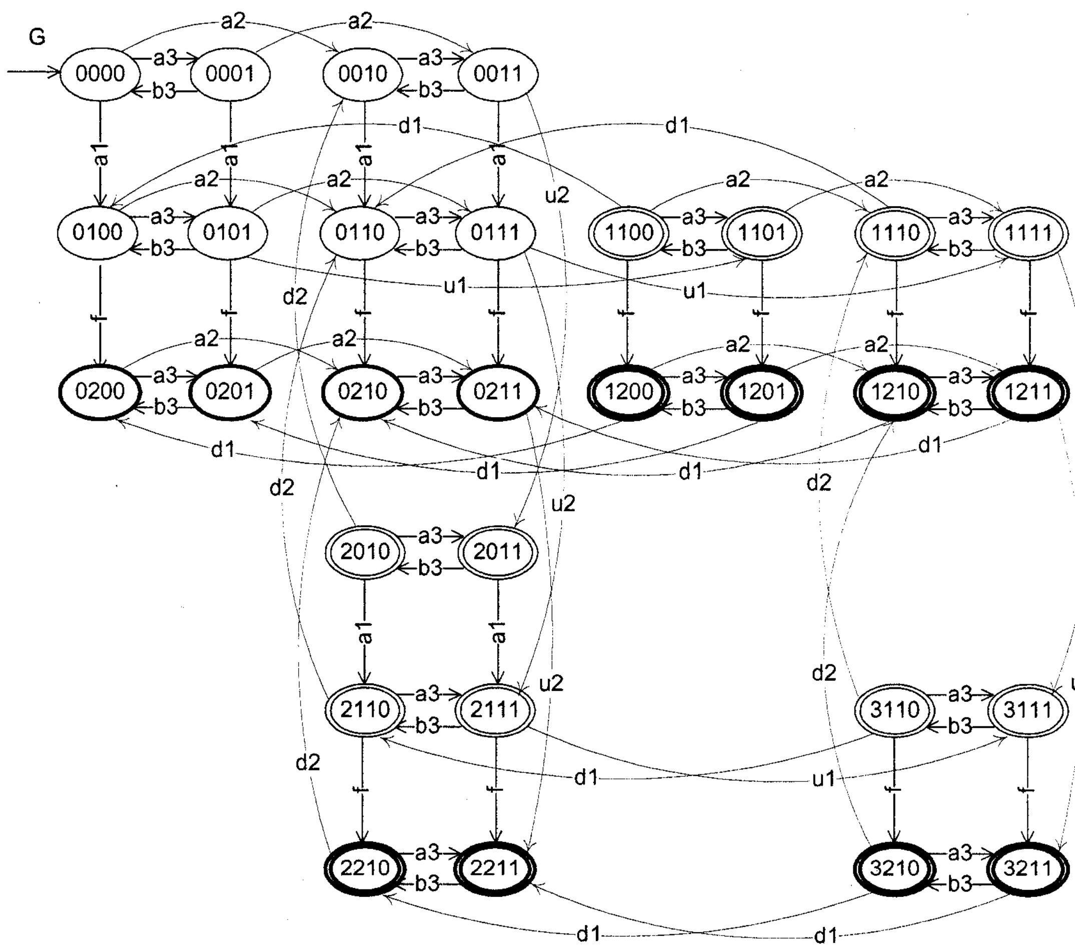

Figure 5. 16: The entire plant automata

The combined model of all three valves, $V=Sync(V_1, V_2, V_3)$ is shown in Figure 5.15. The state of V is represented by a three-tuple (V_1, V_2, V_3). In order to derive the overall plant automaton from V and E, V shall be modified to include the events $\{u_1, d_1, u_2, d_2\}$ since they are related to the valves.

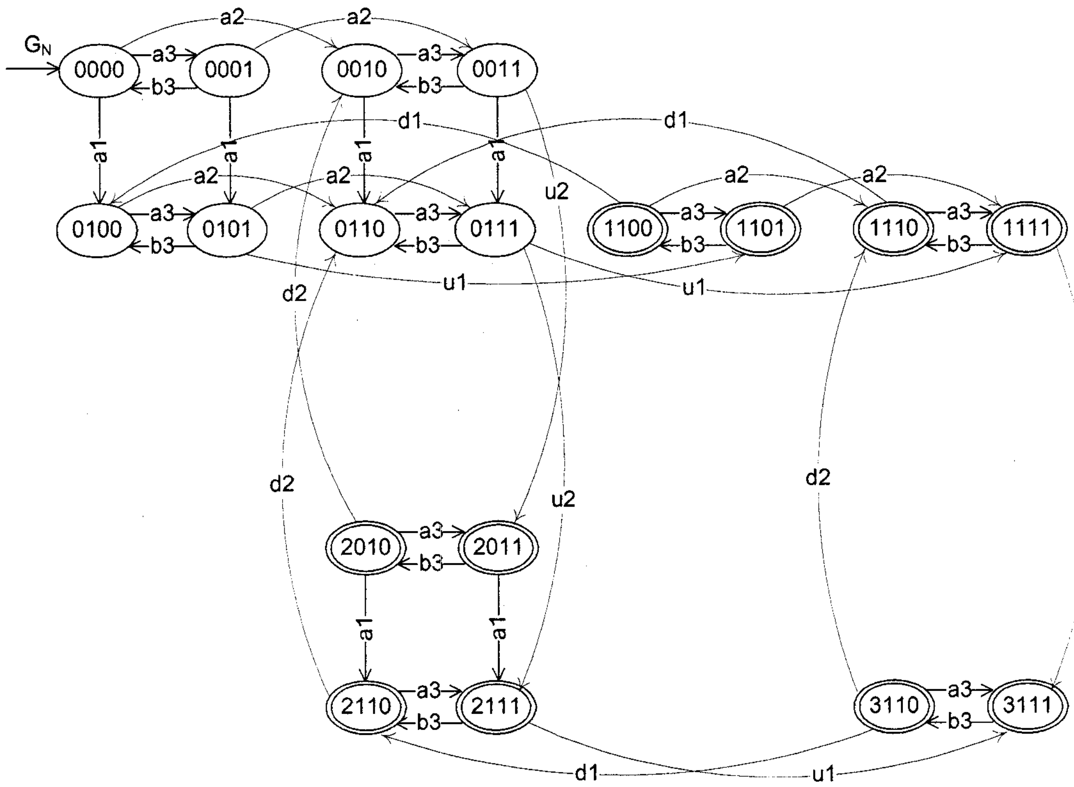

Figure 5. 17: The normal model of the propulsion system

We know u_1 (u_2) and d_1 (d_2) can not happen simultaneously at a given state. In addition, u_1 (u_2) is allowed to happen only if V_1 (or V_2) and V_3 are opened. Otherwise, d_1 (resp. d_2) is allowed. For example at state 000, all values are closed and thus events d_1 and d_2 may happen. Hence, we add d_1 and d_2 self-loops to state 000 of model V. At state 011, valve 1 is closed and Valve 2 and 3 are opened. Thus, d_1 and u_2 are allowed to happen. Hence, we add d_1 and u_2 self-loops to state 011. Necessary self-loop can be added to other states similarly. The automaton of the entire propulsion system G can be obtained by synchronizing V' with E: $G=Sync(V',E)$.(Figure 5.16).

The state of the propulsion system is expressed as a four-tuple (E, V_1, V_2, V_3). There are $4\times3\times2\times2=48$ states in total. The number of reachable states is 30. The marked states are states where at least one engine generates thrust.

The normal model G_N of the system is illustrated as Figure 5.17. It is obtained by removing the faulty states of G. The normal-failure model G_{NF} of the system is the same as the entire propulsion system model G shown in Figure 5.16.

The specification is that Engine 1 and Engine 2 shall not be fired simultaneously. In addition, the closed-loop behavior shall be nonblocking for both the normal model G_N and the normal-failure model G_{NF}. Specifications are marked by the automaton SP shown in Figure 5.18.

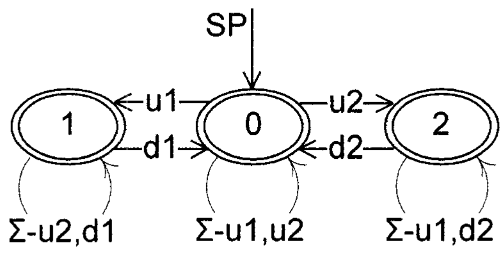

Figure 5. 18: Overall specification automaton

5.2.3 Robust Problem Formulation for the Fault Recovery Problem

As mentioned in Chapter 2, fault recovery problems can be treated as special cases of robust supervision problems. To solve the fault recovery problems, one can synthesize an admissible nonblocking supervisor which meets safety requirements in both normal and failure modes.

In our simple example, the system under supervision shall be nonblocking in both normal mode and failure mode. In addition, only at most one engine must be on at a time. Intuitively, the possible control operations will be:

1) Use Engine 1: Operate on Valve 3 to start up and cut off Engine 1. If Valve 1 is stuck-closed, then open Valve 2.

2) Use Engine 2. Operate on Valve 3 to start up and cut off Engine 2. There will be no failure since we assume that Valve 1 stuck-closed only happens when Valve 1 is open.

If we solve the problem using conventional supervisory control, Valve 2 will be allowed to open after Valve 1 is opened in the system under supervision of the offline supremal supervisor. Valve 3 is not allowed to open at string a_1a_2 to avoid both engines firing simultaneously. If Valve 1 becomes stuck-closed, Valve 3 can then be enabled allowing Engine 2 to fire without risk of two engines firing at the same time.

However, the conventional supervision will reach a deadlock in the normal model if Valve 1 and Valve 2 are both opened. The active events of the current state are a_3 and f. Event a_3 is disabled by the offline supervisor to avoid both engines firing simultaneously. Failure event f never happens in the normal mode. As a result, no engine can fire and a deadlock is reached in the normal model. Therefore, the traditional supervisory control will not meet the supervisor synthesis requirements.

The two possible models of the system are G_N and G_{NF}. The legal behavior of the normal mode E_N can be derived by taking the product of specification automaton *SP* and the normal model G_N. Similarly, the legal behavior of the normal-failure mode E_{NF} is obtained from *SP* and G_{NF}. The resulting automata of E_N and E_{NF} are H_N and H_{NF} show in Figure 5.19 and 5.20.

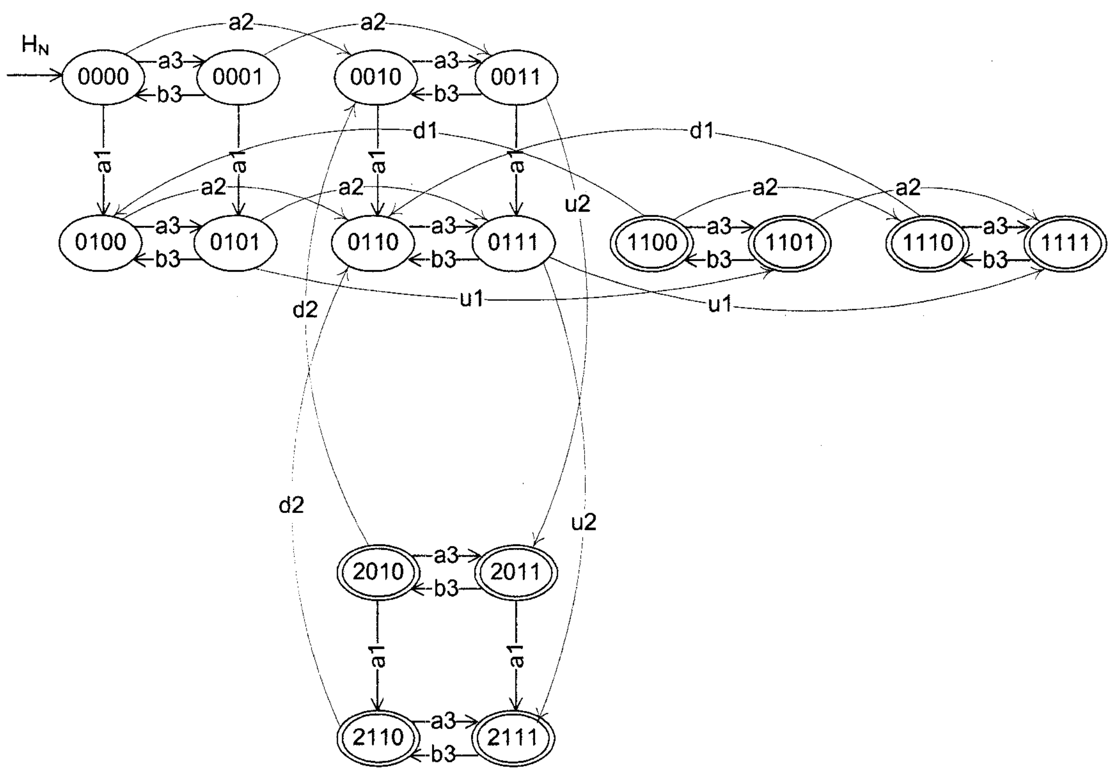

Figure 5. 19: Legal behavior of the normal model

We can see that legal behavior H_N is a subautomaton of G_N with states 3××× deleted. Similarly, specification H_{NF} is the subautomaton of G_{NF} by removing the states with both engines on. Therefore, the specifications can be in terms of illegal states where both engines are fired. We can also see that E_N and E_{NF} are relative-closed with respect to G_N and G_{NF} respectively. We first set up the RNSCP-S problem for the fault recovery problem as follows.

Problem 5.1: Given G_N and G_{NF} shown in Figure 5.16 and 5.17 and legal behavior marked by H_N and H_{NF} in Figure 5.19 and 5.20, design an optimal state-based robust supervisor such that $L_m(G_N,\gamma) \subseteq E_N$, $\overline{L_m(G_N,\gamma)} = L(G_N,\gamma)$, $L_m(G_{NF},\gamma) \subseteq E_{NF}$, and $\overline{L_m(G_{NF},\gamma)} = L(G_{NF},\gamma)$. □

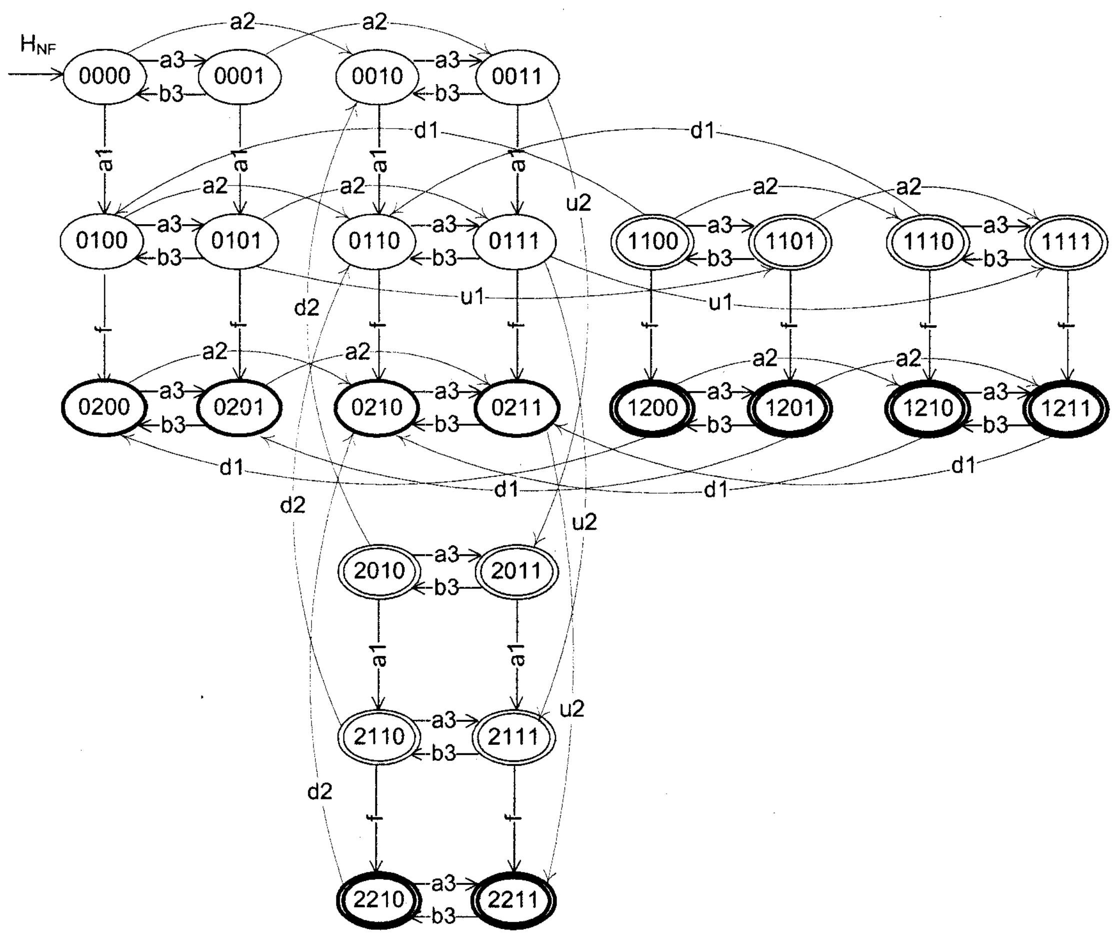

Figure 5. 20: Legal behavior of the normal-failure model

5.2.4 Manual Solution to Problem 5.1

We now employ the direct approach to solve the above fault recovery problem manually offline and online. We only derive online control policy for the initial state and state reached by string $a_1a_3u_1$ as an example. The online control decision for the two states is verified to be the same as the offline optimal decision.

Offline Solution: Now, we apply the Algorithm 3.2A to compute the supremal solution $SupRCS((E_N, E_{NF}))$ and then synthesize an optimal robust nonblocking supervisor for the fault recovery problem of the propulsion system.

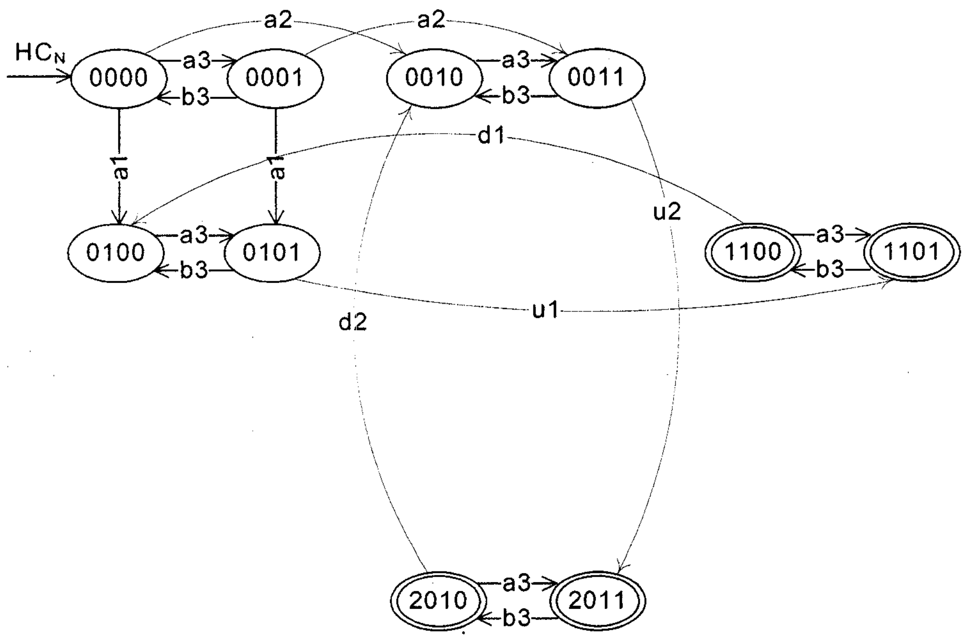

Figure 5. 21: Controllable normal automaton of the first iteration: Offline Solution

E_N and E_{NF} are the legal behaviors of the normal model G_N and the normal-failure model G_{NF} respectively. E_N and E_{NF} are relative-closed with respect to G_N and G_{NF}.

Hence, $K_1^0 = SupR(E_N) = E_N$ and $K_2^0 = SupR(E_{NF}) = E_{NF}$. However, K_1^0 and K_2^0 are not controllable. For example, uncontrollable event u_2 takes the state of H_{NF} from 1211 to illegal state 3211 where two engines are fired. We next calculate the supremal controllable sublanguage $SupC(K_1^0)$ and $SupC(K_2^0)$. They are marked by the automata HC_N and HC_{NF} displayed in Figure 5.21 and Figure 5.22.

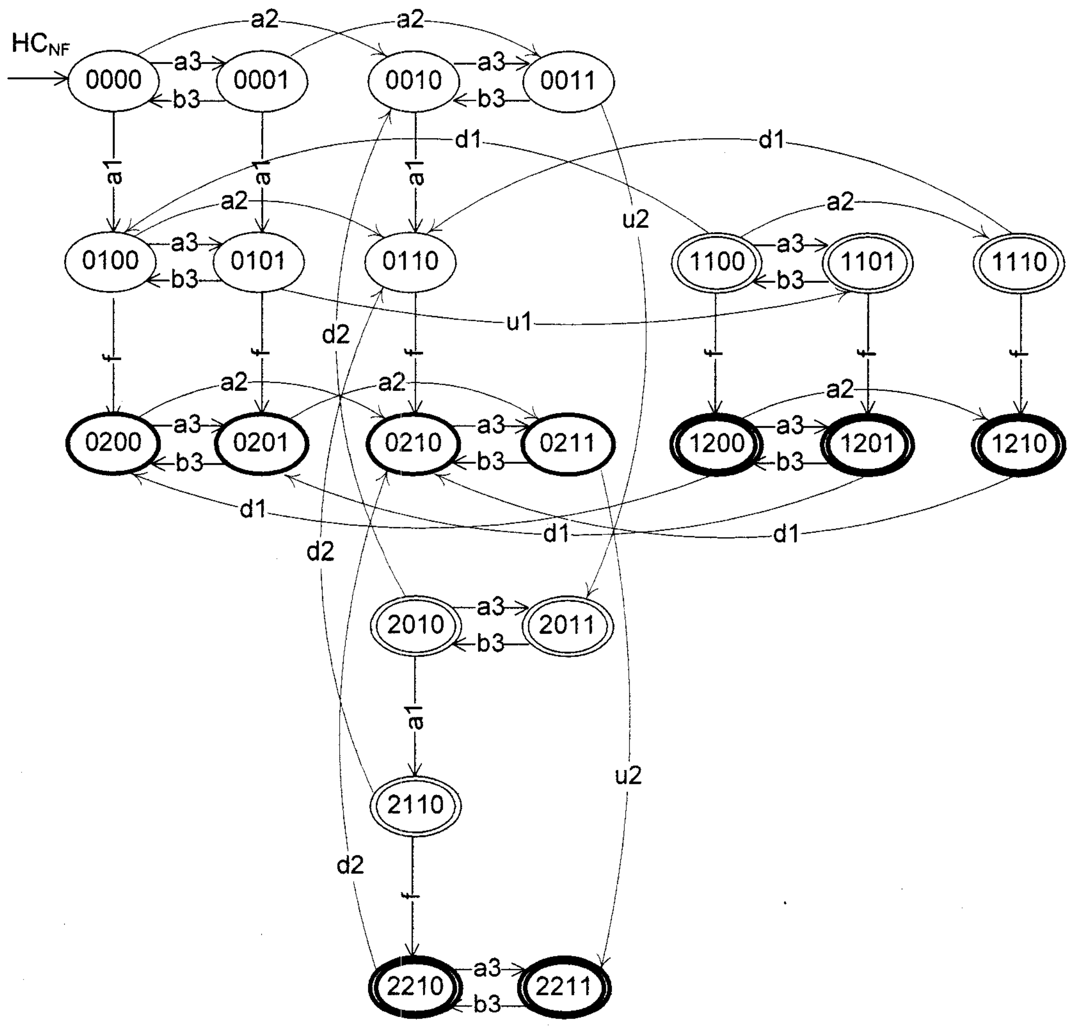

Figure 5. 22: Controllable normal-failure automaton of the first iteration: Offline Solution

$SupC(K_1^0)$ and $SupC(K_2^0)$ are controllable and remain relative-closed with respect to G_N and G_{NF}. However, they are not consistent. For example, state 0110 can be reached in G_{NF} but is removed in G_N. Physically, state 0110 corresponds to state (both engines off, Valve 1 opened, Valve 2 opened, Valve 3 closed). The supremal consistent sublanguage $K_1^1 = SupS(SupC(K_1^0))$ marked by HCS_N (the same automaton as HC_N in Figure 5.21) is the same with $SupC(K_1^0)$. The supremal consistent sublanguage $K_2^1 = SupS(SupC(K_2^0))$ can be marked by automaton HCS_{NF} in Figure 5.23 which is obtained by removing states 0110, 1110, and 2110 from HC_{NF}.

We can verify that (K_1^1, K_2^1) are now controllable, consistent, and relative-closed with respect to G_N and G_{NF}. Thus, (K_1^1, K_2^1) characterizes the optimal solution to the fault recovery problem. If we realize a supervisor V_F as automaton HCS_{NF} and a supervisor V_N as automaton HCS_N, then the union of supervisors V_F and V_N will be the optimal robust supervisor for the fault recovery problem.

The resulting supervisor basically disables a_1 if a_2 is executed and disables a_2 if a_1 happens. In other words, it prevents Valve 1 and Valve 2 from being open at the same time to avoid reaching illegal states or being blocked. The supervisor allows Valve 2 to be opened to fire Engine 2 if Valve 1 becomes stuck-closed. The function of the supervisor is explained below in more detail.

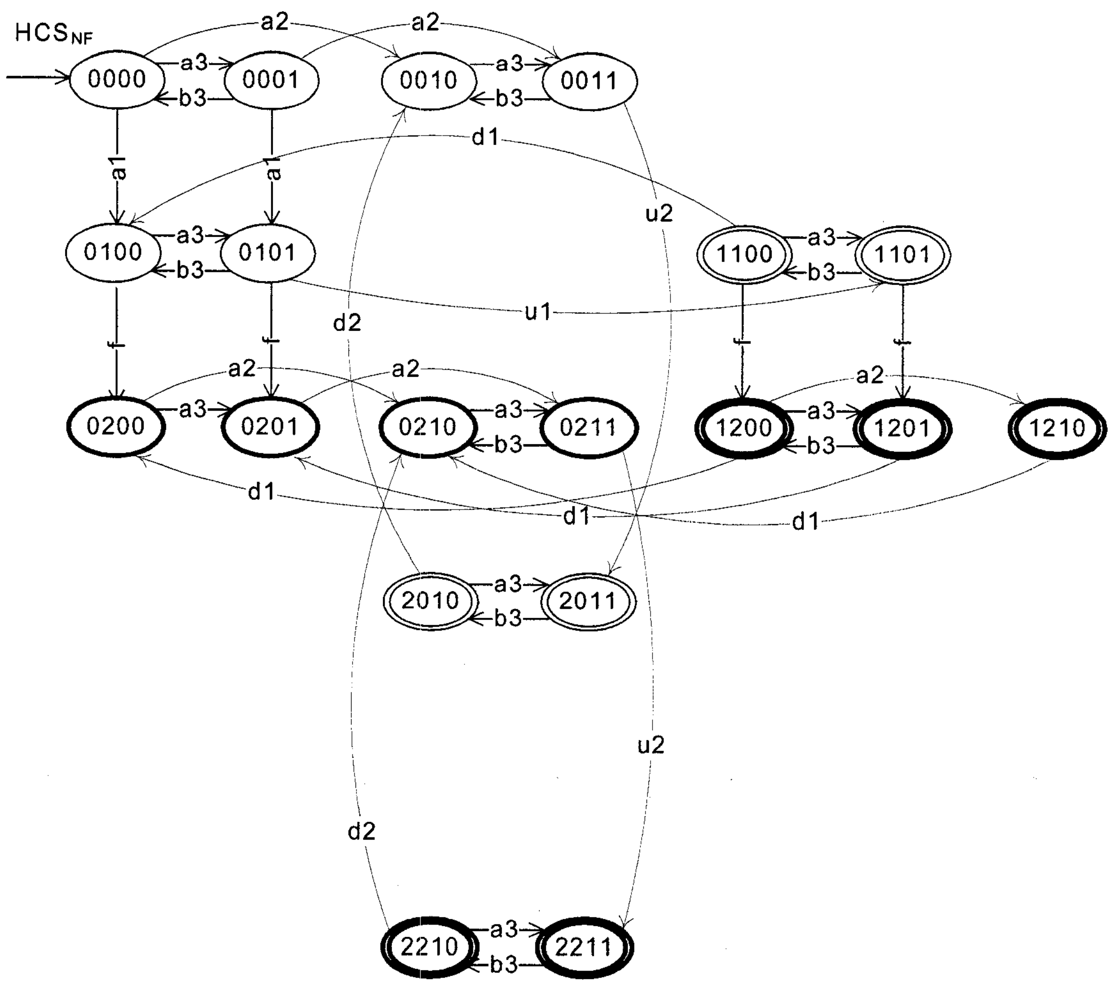

Figure 5. 23: Resulting normal-failure automaton of the first iteration: Offline Solution

At state 0010, a_1 is disabled by the supervisor to prevent the system from entering state 0110. State 0110 is not a desirable state in the sense that the only active event a_3 in the normal model has to be disabled to avoid entering illegal states. As a result, the system will remain in state 0110 in the normal model without being able to reach any marked state. In other word, blocking happens in the normal model.

At state 2011, event a_1 is disabled by the supervisor to prevent system from entering state 2111 which leads to illegal state 3111 by uncontrollable event u_1.

At state 2010, the supervisor disables event a_1 to avoid entering state 2110 where uncontrollable event d_2 brings the system to state 0110. We know that state 0110 is not a desirable state from the previous analysis.

The function of the above supervisor at other states can be interpreted similarly. We further explain the physical function of the supervisor at some states.

Physically, we can explain the behavior of the closed-loop system at state 2010 as follows. At the current state 2010, Engine 2 is on and Engine 1 is off, Valve 1 and 3 are closed, and Valve 2 is open. If a_1 is executed, it means Valve 1 is open and the system reaches state 2110. Engine 2 will be off and thus state 0110 will be reached. Only Valve 3 can be operated at the state 0110 in the normal model. If we turn Valve 3 open, then both engines will be on inevitably, leading to illegal states. If we keep Valve 3 closed, then both engines will be off which means the system will never reach a marked state and thus blocking happens in the normal mode. Therefore, event a_1 at state 2010 has to be disabled by the supervisor.

From the above analysis, we can see that the resulting supervisor solves the fault recovery problem and indeed does the work as described in the intuitive operation discussed before.

Online Solution: We next solve the fault recovery problem using the online direct approach. Obviously, the fault recovery problem satisfies preconditions of RNSCP-S and thus VLP-S can be employed. The empty string ε and string $a_1a_3u_1$ are taken as examples to illustrate the online direct robust supervisor synthesis procedure.

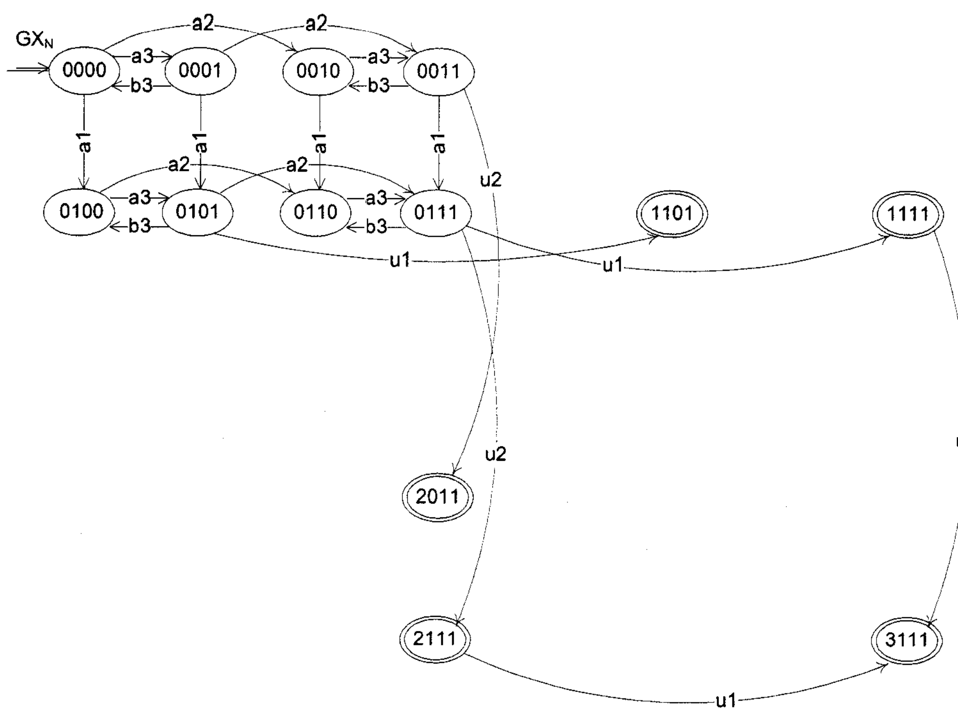

Figure 5. 24: Normal expansion for the initial state

At Initial State x_0 with $s = \varepsilon$: At string $s = \varepsilon$, we first expand G_N and G_{NF} simultaneously according to the stop rules. For example, only uncontrollable event f is expanded at the marked state 1101 of G_N and G_{NF}, and no event is expanded at the illegal state 3111 of G_N and G_{NF}. The expansion of the normal model and the failure model are GX_N and GX_{NF} shown in Figure 5.24 and 5.25.

The specification automata HX_N and HX_{NF} for the expansions can be obtained by removing the illegal states 3111 and 3211 from GX_N and GX_{NF}. They are relative-closed but not controllable. For example, state 1111 leads to the illegal state 3111 by uncontrollable event u_2. The supremal controllable sublanguages are represented by the automations HXC_N and HXC_{NF} that are obtained from HX_N and HX_{NF} by removing the states 1111,1211, 2111, and 0111.

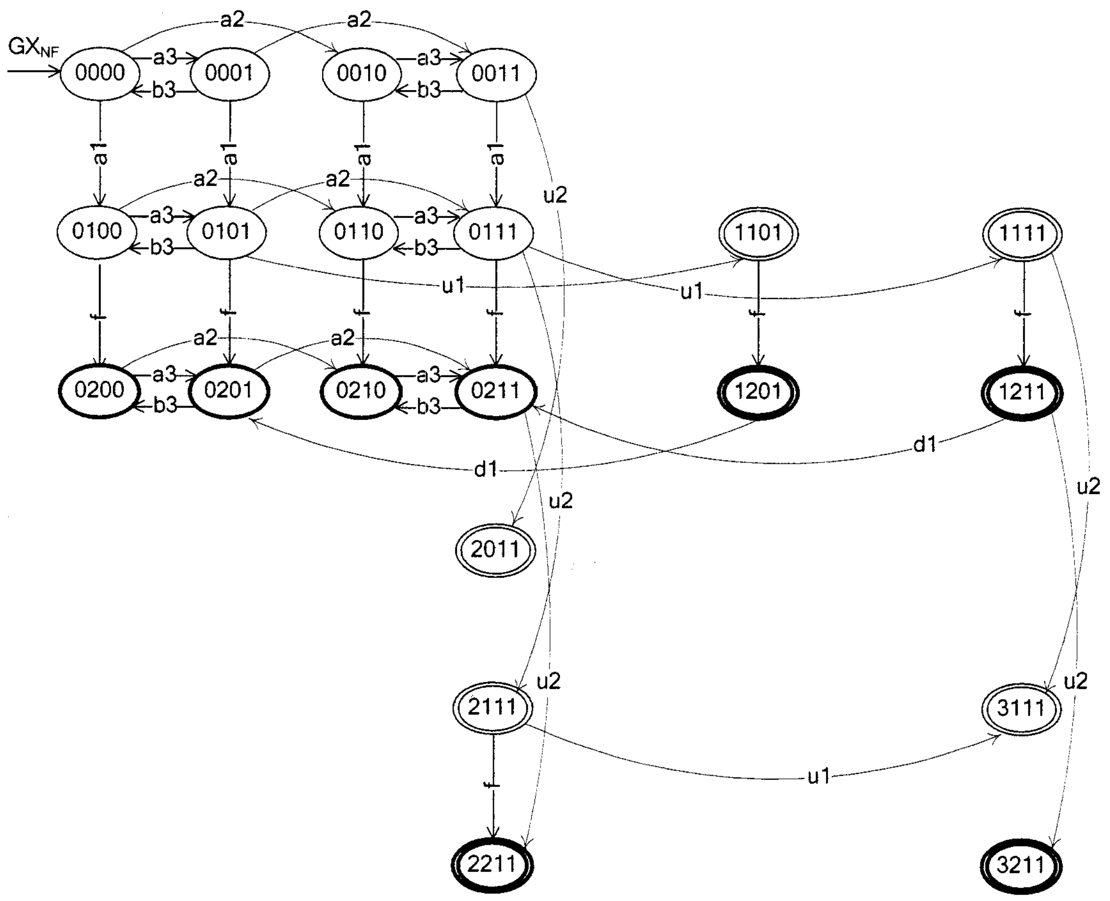

Figure 5. 25: Normal-failure expansion for the initial state

HXC_N and HXC_{NF} are not consistent. For instance, the state 0110 can be reached in HXC_{NF} but is removed in HXC_N. The supremal consistent sublanguages are represented by $HXCS_N$ and $HXCS_{NF}$ shown in Figure 5.26 and 5.27. $HXCS_{NF}$ is the automata by removing state 0110 from HXC_{NF}. $HXCS_N$ is the same automaton as HXC_N.

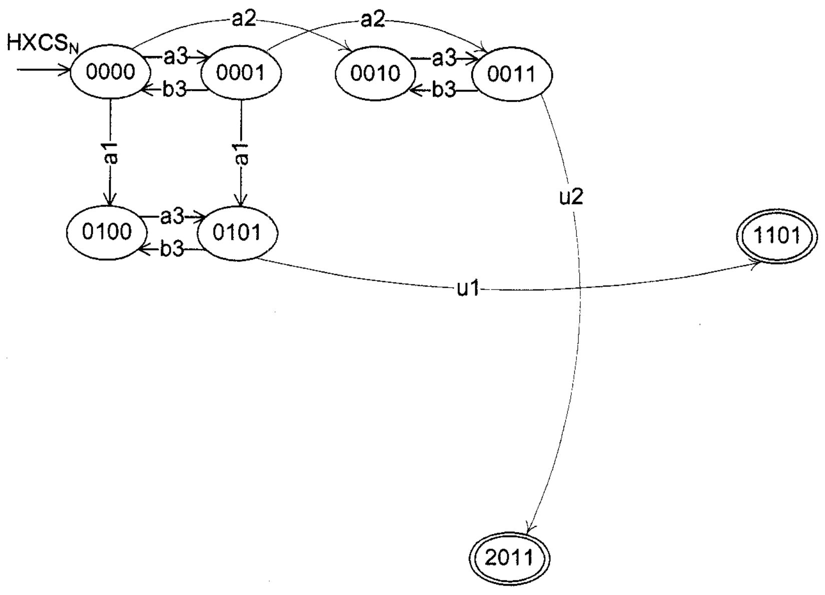

Figure 5. 25: Resulting normal automaton of the first iteration:
Online Solution at the initial state

We can verify that the languages represented by $HXCS_N$ and $HXCS_{NF}$ are relative-closed, controllable, and consistent with respect to GX_N and GX_{NF}. Therefore, the iteration stops and $HXCS_N$ and $HXCS_{NF}$ represent the supremal controllable, relative-closed, and consistent sublanguages.

The control policy at the empty string for the failure model and normal model are then obtained from automaton $HXCS_N$ and $HXCS_{NF}$ respectively. The overall supervision policy for the empty string will be the union of them. As a result, events $\{a_1, a_2, a_3\}$ are enabled by the online supervisor. The online supervisor has the same control policy at the initial state as the offline solution.

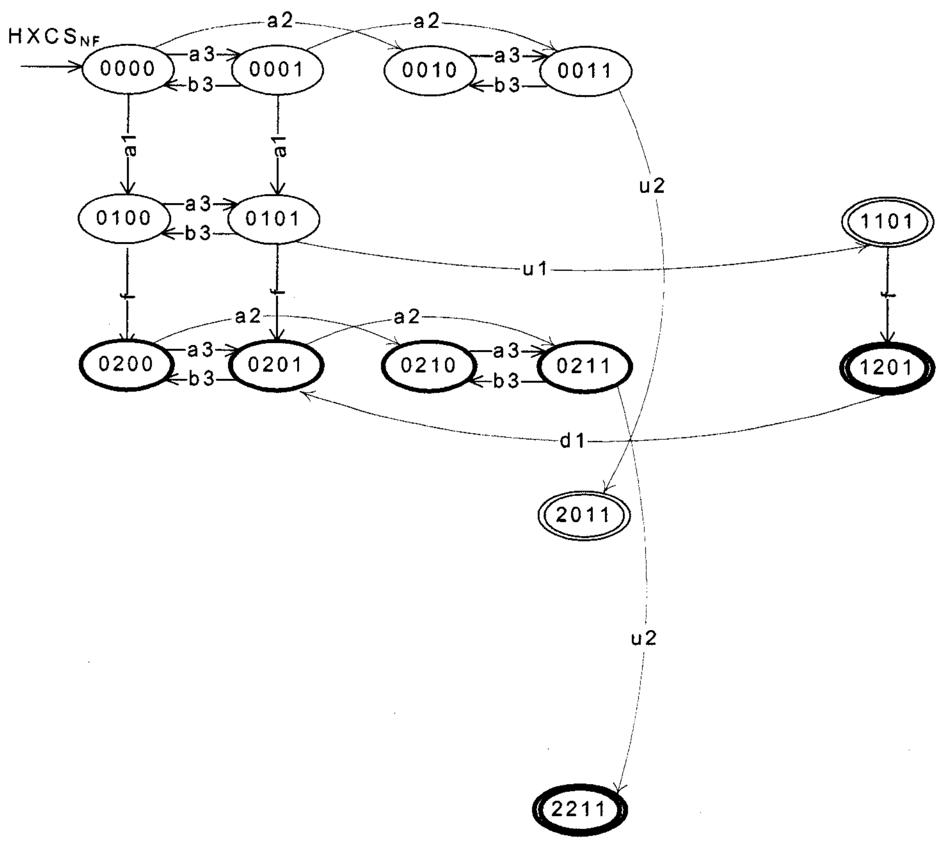

Figure 5. 26: Resulting normal-failure automaton of the first iteration: Online Solution at the initial state

At State x **reached by** $s = a_1a_3u_1$:At string $s = a_1a_3u_1$, we take state 1101 of G_N and G_{NF} as the initial state for expansion and call the reachable part GM_N and GM_{NF}. Then, we expand GM_N and GM_{NF} simultaneously according to the expansion rules. For example, only uncontrollable events d_1 and f at the marked state 1100 of GM_N and GM_{NF} are expanded. No active event of the illegal state 3211 of GM_N and GM_{NF} is expanded. The resulting expansion automata of GM_N and GM_{NF} are GX_N and GX_{NF} as shown in Figure 5.28 and 5.29.

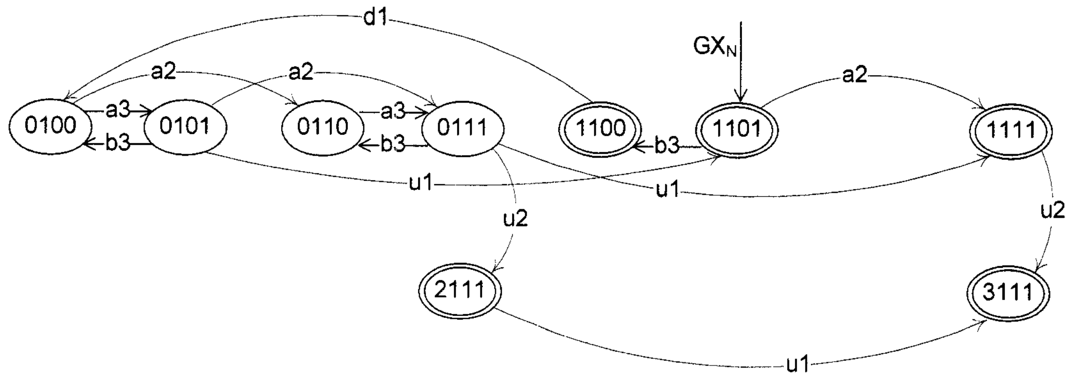

Figure 5. 27: Normal expansion for the state 1101

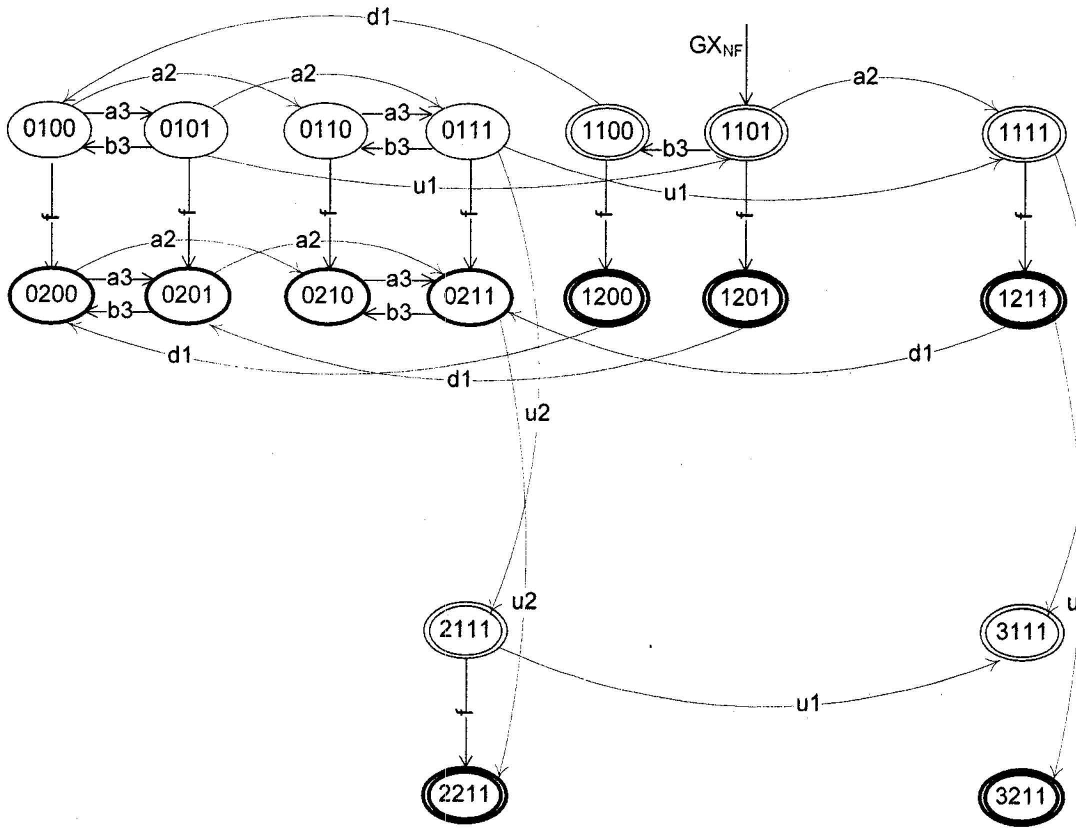

Figure 5. 28: Normal-failure expansion for the state 1101

The specification for the expansions will be represented by automata HX_N and HX_{NF} by removing states 3111 and 3211 from GX_N and GX_{NF}. They are relative-closed but not controllable with respect to GX_N and GX_{NF}. For instance, uncontrollable event u_1 leads the state 2111 in HX_N and HX_{NF} to the illegal state 3111. The supremal controllable sublanguages are represented by the automata HXC_N and HXC_{NF} with states 1111,1211, 2111, and 0111 removed from HX_N and HX_{NF}.

Obviously, HXC_N and HXC_{NF} are not consistent since the state 0110 can be reached in HXC_{NF} but is removed in HXC_N. The supremal consistent sublanguages are represented by automata $HXCS_N$ and $HXCS_{NF}$. $HXCS_N$ is exactly the same automaton as HXC_N. $HXCS_{NF}$ is the automaton from HXC_{NF} with state 0110 deleted.

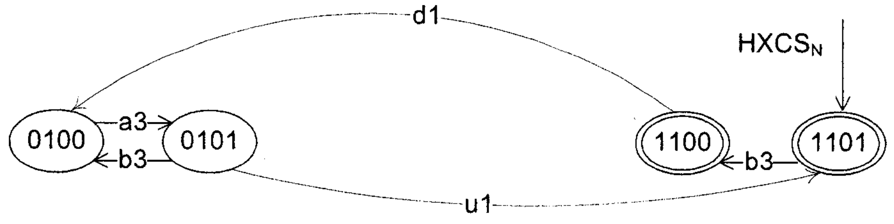

Figure 5. 29: Resulting normal automaton of the first iteration:
Online Solution at the state 1101

We can verify that the languages represented by $HXCS_N$ and $HXCS_{NF}$ are relative-closed, controllable, and consistent with respect to GX_N and GX_{NF}. Thus, the algorithm terminates and $HXCS_N$ and $HXCS_{NF}$ provide the supremal solution to the fault recovery problem.

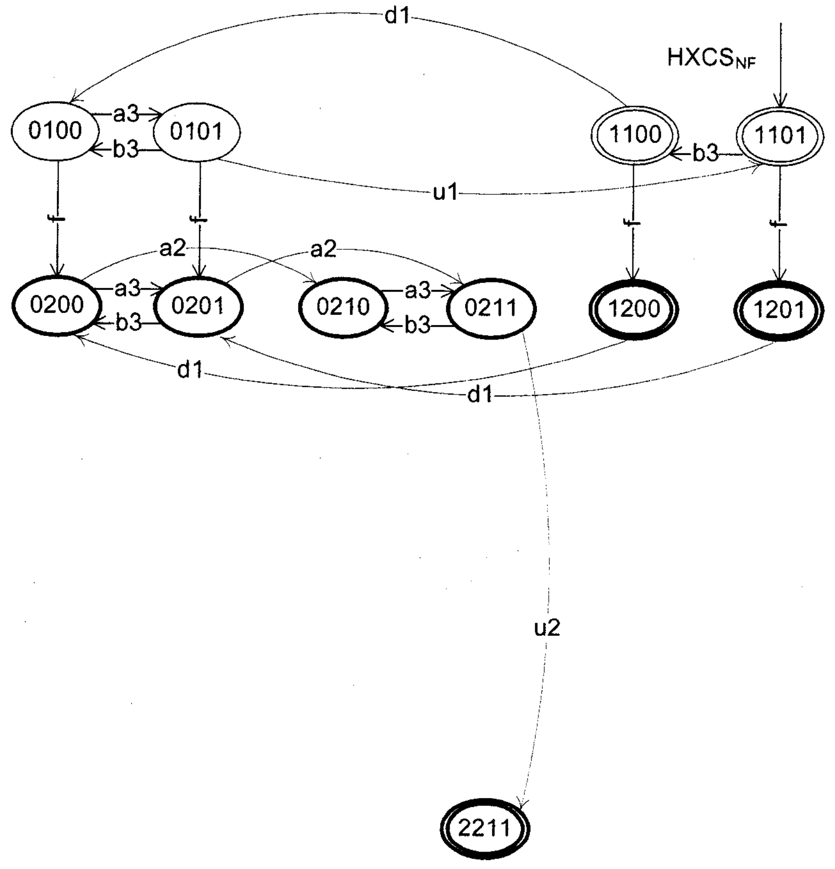

Figure 5. 30: Resulting normal-failure automaton of the first iteration: Online Solution at the state 1101

We then derive a control policy for the failure model and the normal model from $HXCS_N$ and $HXCS_{NF}$ respectively. Events b_3 and f are enabled by the supervisor of the failure model. Event b_3 is enabled by the supervisor of the normal model. The overall set of enabled events will be the union of them $\{b_3, f\}$. As a result, a_2 is disabled at the current state reached by $a_1a_3u_1$. This agrees with the offline direct solution.

The online control policy for other strings can be derived similarly and the same control policy with the offline control will be obtained. Since the online control policy is exactly the same as the offline control policy at each string, the online supervisor is valid.

5.2.5 TTCT Solution to Problem 5.1

In the previous subsection, the fault recovery problem was solved manually. Here, the problem is solved using Procedure 3.5 offline and online.

TTCT Offline Solution to Problem 5.1:

1) **Initialization**

Build automata G_N, G_{NF}, H_N, and H_{NF} using TTCT. H_N, and H_{NF} represent specification languages E_N and E_{NF}. Compute $H1_N = SupR(H_N)$ and $H1_{NF} = SupR(H_{NF})$ using Procedure 3.4.

2) **Iteration 1:**

Calculate $H1C_N = SupCon(G_N, H1_N)$ and $H1C_{NF} = SupCon(G_{NF}, H1_{NF})$. Calculate $(H2_N, H2_{NF}) = SupS((H1C_N, H1C_{NF}))$ using Procedure 3.3. Call procedure Eq and obtain $Eq(H2_N, H1_N) = False$ and $Eq(H2_{NF}, H1_{NF}) = False$. Since $H2_N$ and $H2_{NF}$ are different with $H1_N$ and $H1_{NF}$, go to the next iteration.

3) **Iteration 2**

Calculate $H2C_N = SupCon(G_N, H2_N)$ and $H2C_{NF} = SupCon(G_{NF}, H2_{NF})$. $(H3_N, H3_{NF}) = SupS((H2C_N, H2C_{NF}))$ using Procedure 3.3. Call procedure Eq (Procedure 3.2) and obtain $Eq(H3_N, H2_N) = True$ and $Eq(H3_{NF}, H2_{NF}) = Ture$. $H3_N$ and $H3_{NF}$ are the same with $H2_N$ and $H2_{NF}$. Hence, $H3_N$ and $H3_{NF}$ are controllable and consistent. STOP.

4) Solution:

The solution is $SupRCS((E_N, E_{NF})) = (L_m(H3_N), L_m(H3_{NF}))$. If we realize a supervisor V_F as automaton $H3_N$ and a supervisor V_N as automaton $H3_{NF}$, then the union of supervisors V_F and V_N will be the optimal robust supervisor for the fault recovery problem. □

We have illustrated the offline solution using TTCT. We can verify that the solution agrees with the manual offline solution.

TTCT Online Solution to Problem 5.1

We next use TTCT to solve the fault recovery problem online. In order to use TTCT to solve a robust problem online, the problem shall satisfy the preconditions of RNSCP-S. Since fault recovery problem with specification given as illegal states can be treated as a special case of RNSCP-S, the fault recovery problem can be solved online using TTCT. We expect the same control policy at the current string with the offline solution. We first derive control policy using TTCT for the empty string.

At initial state x_0 with s=ε:

1) Expansion

Build automata G_N, G_{NF}, and H_N, and H_{NF} using TTCT. H_N, and H_{NF} represent specification languages E_N and E_{NF}. We can verify that G_N, G_{NF}, H_N, and H_{NF} satisfy the precondition of RNSCP-S. Hence, we can obtain the expansion for the current string ε using Procedure 5.2. The expansion GX_N and GX_{NF} for G_N and G_{NF} can be obtained by deleting controllable events of marked states except the current state and all active events of illegal states from G_N and G_{NF}, and then taking the accessible part of them. Then we can employ Procedure 3.5 to compute the supremal element for the expansion windows as follows.

2) Initialization

Compute $HX_N = Meet(H_N, GX_N)$ and $HX_{NF} = Meet(H_{NF}, GX_{NF})$ for the specification EX_N and EX_{NF} of expansion GX_N and GX_{NF}. Compute $HX1_N = SupRa(HX_N)$ and $HX1_{NF} = SupRa(HX_{NF})$ using Procedure 3.4.

3) Iteration 1

Calculate $HX1C_N = SupCon(GX_N, HX1_N)$ and $HX1C_{NF} = SupCon(GX_{NF}, HX1_{NF})$. Calculate $(HX2_N, HX2_{NF}) = SupSis((HX1C_N, HX1C_{NF}))$ using Procedure 3.3. Call procedure Eq (Procedure 3.2) and obtain $Eq(HX2_N, HX1_N) = False$ and $Eq(HX2_{NF}, HX1_{NF}) = False$. Since $HX2_N$ and $HX2_{NF}$ are different with $HX1_N$ and $HX1_{NF}$, go to the next iteration.

4) Iteration 2

Calculate $HX2C_N = SupCon(GX_N, HX2_N)$ and $HX2C_{NF} = SupCon(GX_{NF}, HX2_{NF})$. $(HX3_N, HX3_{NF}) = SupSis((HX2C_N, HX2C_{NF}))$ using Procedure 3.3. Call procedure *Eq* (Procedure 3.2) and obtain $Eq(HX3_N, HX2_N) = True$ and $Eq(HX3_{NF}, HX2_{NF}) = Ture$. $H3_N$ and $H3_{NF}$ are the same with $HX2_N$ and $HX2_{NF}$. Hence, $HX3_N$ and $Hx3_{NF}$ are controllable and consistent. STOP.

5) Solution:

The solution for expansion is $SupRCS((EX_N, EX_{NF})) = (L_m(HX3_N), L_m(HX3_{NF}))$. The control policy for the initial state x_0 will be the union of the active events of the initial states in $HX3_N$ and $HX3_{NF}$. The result is $\gamma(x_0) = \{a_1, a_2, a_3\}$. □

We now derive control policy for a state reached by $s = a_1 a_3 u_1$ which is not the initial state. Online control policy to other non-initial states can be computed similarly. There is no way to assign the current state as initial state in TTCT. Thus, the expansion window can not be directly represented by an automaton obtained in TTCT. To solve the problem, we derive an automaton that represents the catenation of the current string with all strings in the expansion window. In the end, we take the post behavior after the current string to derive the control policy for the current state.

At state x reached by $s = a_1 a_3 u_1$:

1) Expansion for the current string $s = a_1 a_3 u_1$:

Build an automaton A with $L_m(A) = a_1 a_3 u_1 \Sigma^*$. Generate $GA_N = Meet(G_N, A)$ and $GA_{NF} = Meet(G_{NF}, A)$ in TTCT. Obtain Specification automata $HA_N = Meet(H_N, GA_N)$ and $HA_{NF} = Meet(H_{NF}, GA_{NF})$ in TTCT. Verify that GA_N , GA_{NF} , HA_N , and HA_{NF} satisfy the precondition of RNSCP-S. Hence, the expansion for string $s = a_1 a_3 u_1$ can be obtained directly using Procedure 5.2. The expansion GX_N and GX_{NF} for GA_N and GA_{NF} can be obtained by deleting controllable events of marked states except the states reached by strings in $\bar{s}$ and all active events of illegal states from GA_N and GA_{NF}, and then taking the accessible part of them. We then compute the supremal element of specification for expansions GA_N and GA_{NF} using Procedure 3.5 *SupRCSa* as below.

2) Initialization

Compute $HX_N = Meet(HA_N, GX_N)$ and $HX_{NF} = Meet(HA_{NF}, GX_{NF})$ for the specification languages EX_N and EX_{NF} of expansions GX_N and GX_{NF} . Compute $HX1_N = SupRa(HX_N)$ and $HX1_{NF} = SupRa(HX_{NF})$ using Procedure 3.4.

3) Iteration 1.

Calculate $HX1C_N = SupCon(GX_N, HX1_N)$ and $HX1C_{NF} = SupCon(GX_{NF}, HX1_{NF})$ in TTCT. Calculate $(HX2_N, HX2_{NF}) = SupSis((HX1C_N, HX1C_{NF}))$ using Procedure 3.3. Call procedure *Eq* (Procedure 3.2) and obtain $Eq(HX2_N, HX1_N) = False$ and $Eq(HX2_{NF}, HX1_{NF}) = False$. Since $HX2_N$ and $HX2_{NF}$ are different with $HX1_N$ and $HX1_{NF}$, go to the next iteration.

4) Iteration 2

Calculate $HX2C_N = SupCon(GX_N, HX2_N)$ and $HX2C_{NF} = SupCon(GX_{NF}, HX2_{NF})$. $(HX3_N, HX3_{NF}) = SupS((HX2C_N, HX2C_{NF}))$ using Procedure 3.3. Call procedure *Eq* (Procedure 3.2) and obtain $Eq(HX3_N, HX2_N) = True$ and $Eq(HX3_{NF}, HX2_{NF}) = Ture$. $H3_N$ and $H3_{NF}$ are the same with $HX2_N$ and $HX2_{NF}$. Hence, $HX3_N$ and $Hx3_{NF}$ are controllable and consistent. STOP.

5) Solution:

The solution for expansion is $SupRCS((EX_N, EX_{NF})) = (L_m(HX3_N), L_m(HX3_{NF}))$. The control policy for the state reached by $s = a_1a_3u_1$ will be the union of the active events of state reached by string $s = a_1a_3u_1$ in $HX3_N$ and $HX3_{NF}$. The result is $\gamma(x) = \{b_3, f\}$. □

The online control policy is verified exactly the same as the offline control policy at the initial state and the state reached by string $a_1a_3u_1$. The equivalence can be verified for other states similarly.

Obviously, the memory requirements of online control policy at the current state are less than of a supervisor designed based on offline control policy in this simple example. The benefit of online implementation will be significant for large and complex systems where offline solution is very complex.

Chapter 6:

Conclusion

This chapter summarizes our work in this thesis and points out possible directions for future work.

6.1 Summary

In this thesis, supervisory control is studied under model uncertainty in which the exact system model is uncertain but could be one of a set of known models. Offline solution to robust nonblocking supervisory control problem with a direct approach is first developed in Chapter 3 and then implemented online with a variable lookahead policy algorithm in Chapter 4. Nonblocking supervisory control problem with multiple markings is solved as a special case of robust nonblocking supervisory control problem. State-based robust nonblocking supervisory control problem is formulated and solved using variable lookahead policy in Chapter 5. However, RNSCP-S by no means imposes any restriction in comparison with RNSCP since any RNSCP can be transferred to an equivalent RNSCP-S after automaton refinement. As an illustrative example, the VLP-S algorithm is used to solve a fault recovery problem.

Offline robust supervision with indirect approach needs to synthesize the overall plant model and specification model. Offline robust supervision with direct approach is proposed in Chapter 3 to relax this requirement. It defines consistent languages which offer insight to robustness. The sufficient and necessary conditions for direct solution to robust nonblocking supervisory control problem are given and proved. We further show that there exists a supremal solution to the robust problem. Then, algorithms that achieve the offline direct optimal solution to RNSCP are developed.

A special application of robust problems is in the control of a plant with multiple sets of marked states (each set corresponding to completion of a different task). The computation is simplified by taking the specific property of the problem into consideration that the generated languages of all possible models are the same. The consistency condition reduces to "equiclosure" in this kind of problems.

Online solutions to robust problems with direct approaches are proposed in Chapter 4 to avoid storing supervisors in computer memory by developing control policy only for the current string. We assume that expansion for each string always terminates. In VLP, the expansion window terminates for each branch until the uncertainty is resolved and thus no attitude is required and validity is guaranteed. It is valid in that it has the same control policy as the offline direct maximal solution does. VLP is suitable for problems where expansion terminates.

We formulated RNSCP-S in Chapter 5 where control domain can be taken as states. VLP-S algorithm was proposed to solve RNSCP-S online. VLP solves RNSCP only if the expansion for each string terminates. In contrast, VLP-S always works for RNSCP-S. VLP-S has advantage over VLP in that any RNSCP that can not be solved using VLP can be transferred to RNSCP-S after automaton refinement and then solved using VLP-S.

Fault recovery problems can not be treated as traditional supervisory control problems. Instead, they can be solved as robust problems. Fault recovery problem with illegal state specification naturally satisfies the setup of RNSCP-S. VLP-S is applied to solve a fault recovery problem for a simple spacecraft propulsion system. The procedure of computing solution to robust problems in TTCT (Appendix 3.1) is used to solve the fault recovery problem of the propulsion system.

6.2 Future Research

We solve the examples of robust control problems manually. Some of the operations were speeded up using TTCT. However, for studying real-model problems (which are typically much more complicated), it would be necessary to have all algorithms implemented as computer code. To have a complete implementation of the design procedure, we need to establish the finite convergence of the computational procedure for supremal controllable, relative-closed, and consistent sublanguages.

We assumed full observation of the system events in this thesis. Future research includes the generalization of the online solution to the robust problem under partial observation. In order to ensure validity, the expansion stop rules have to be modified to accommodate unobservable events. Conventional nonrobust online nonblocking supervisory control problem under partial observation shall be first investigated before online robust nonblocking supervisory control problem under partial observation is tackled.

Our results can be generalized to timed discrete event systems and decentralized robust supervision. Future research could focus on obtaining necessary and sufficient conditions for the existence of a robust supervisor in the above two kinds of robust supervision problems and the corresponding offline and online solutions.

The supervisor design procedure for a given set of possible plant and specification pairs is examined in this thesis. If some new possible plant models are added or some possible models are removed, online approach can potentially adjust to the change accordingly. However, do we need to design a new offline supervisor from scratch? In other words, can we make good use of the current offline solution to accommodate the changing environment and save effort of new supervisor synthesis? The validity of the online approach is another topic for future research.

References

[1] N. Ben Hadj-Alouane, S. Lafortune and F. Lin. "Variable lookahead supervisory control with state information," IEEE Transactions on Automatic Control, Vol. 39, pp. 2398-2410, 1994.

[2] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin, "Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation," Discrete Event Dynamic Systems: Theory and Applications, Vol. 6, pp. 379-427, 1996.

[3] S. E. Bourdon, M. Lawford, and W. M. Wonham, "Robust nonblocking supervisory control of discrete-event systems," IEEE Transactions on Automatic Control, Vol. 50, No. 12, pp. 2015-2021, December 2005.

[4] C. D. Brown, *Spacecraft Propulsion*, AIAA, Washington, DC, 1996.

[5] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, Kluwer Academic, Boston, 1999.

[6] E. Chen and S. Lafortune, "On nonconflicting languages that arise in supervisory control of discrete event systems," Systems and Control Letters, Vol.17, pp. 105-113, 1991.

[7] S. L. Chung, S. Lafortune, and F. Lin, "Limited lookahead policies in supervisory control of discrete event systems," IEEE Transactions on Automatic Control, Vol. 37, No.12, pp. 1921-1935, 1992.

[8] S. L. Chung, 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, College of Engineering, Control Group Reports, University of Michigan, April 1992.

[9] S. L. Chung, S. Lafortune, and F. Lin, "Recursive computation of limited lookahead policies of supervisory controls for discrete event systems," Discrete Event Dynamic Systems: Theory and Applications, Vol. 3, pp. 71-100, 1993

[10] S. L. Chung, S. Lafortune, and F. Lin, "Supervisory Control using variable lookahead policies," Discrete Event Dynamic Systems: Theory and Applications, Vol. 4, pp. 237-268, 1994

[11] J. E. R. Cury and B. H. Krogh, "Robustness of supervisors for discrete-event systems," IEEE Transactions on Automatic Control, Vol. 44, pp.376-379, 1999.

[12] M. Heymann and F. Lin, "On-line control of partially observed discrete event systems," Discrete Event Dynamic Systems, Vol. 4, No. 3, pp. 221-236, 1994.

[13] D. K. Huzel and D. H. Huang, *Modern Engineering for Design of Liquid-propellant Rocket Engines*, AIAA, Washington, DC, 1992.

[14] R. Kumar, H. M. Cheung, and S. I. Marcus, "Extension based limited lookahead supervision of discrete event systems," Automatica, Vol. 34, No. 11, pp. 1327-1344, 1998

[15] S. Lafortune and E. Chen, "The infimal closed controllable superlanguage and its application in supervisory control," IEEE Transactions on Automatic Control, Vol. 35, No. 4, pp.398-405, 1990.

[16] F. Lin, "Robust and adaptive supervisory control of discrete event systems," IEEE Transactions on Automatic Control, Vol. 38, No. 12, pp. 1848-1852, December 1993.

[17] R. X. Meyer, *Elements of Space Technology for Aerospace Engineers*, Academic Press, San Diego, 1999.

[18] M. Moosaei and S. Hashtrudi Zad, "Fault recovery in control systems: A modular discrete-event approach," Proceedings of International Conference on Electrical and Electronics Engineering (CINVESTAV/IEEE), Acapulco, Mexico, pp. 445-450, Sep. 2004.

[19] S. J. Park and J. T. Lim, "Robust nonblocking supervisors for partially observed discrete event systems with model uncertainty under partial observation," IEEE Transactions on Automatic Control, Vol. 45, No. 12, pp. 2393-2396, December 2000.

[20] S. J Park and J. T. Lim, "Robust and nonblocking supervisory control of nondeterministic discrete event systems using trajectory models," IEEE Transactions on Automatic Control, Vol. 47, No. 4, pp. 655-658, April 2002.

[21] S. J. Park and J. T. Lim, "On robust and nonblocking supervisor for nondeterministic discrete event systems," IEICE Transactions on Information and Systems, Vol. E86-D, No. 2, pp. 330-333, 2003.

[22] S. J. Park and J. T. Lim, "Non-blocking supervision for uncertain discrete event systems with internal unobservable transitions," IEE Proceedings of Control Theory and Applications, Vol. 152, pp. 165-170, 2005.

[23] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," SIAM Journal on Control and Optimization, Vol. 25, No. 1, pp. 206-230, January 1987.

[24] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," Proceedings of the IEEE, Vol. 77, No. 1, pp.81-98, Jan. 1989.

[25] A. Saboori and S. Hashtrudi Zad, "Robust supervisory control and blocking-invariant languages," Proc. 43th Annual Allerton Conference on Communication, Control, and Computing, University of Illinois at Urbana-Champaign, pp. 658-667, September 2005.

[26] A. Saboori and S. Hashtrudi Zad, "Fault recovery in discrete event systems," Computational Intelligence Methods and Applications, ICSC Congress, Dec. 2005.

[27] A. Saboori and S. Hashtrudi Zad, "Robust nonblocking supervisory control of discrete-event systems under partial observation," Systems and Control Letters, Vol. 55, pp. 839-848, 2006.

[28] S. Takai, "Estimate based limited lookahead supervisory control for closed language specifications," Automatica, Vol. 33, No. 9, pp. 1739-1743, 1997.

[29] S. Takai, "Maximally permissive robust supervisors for a class of specification languages," IFAC Conference on System Structure and Control, Nantes, France, Vol.2, pp.429-434, 1998.

[30] S. Takai, "Robust supervisory control of a class of timed discrete event systems under partial observation," Systems and Control Letters, Vol. 39, No. 4, pp. 267-273, 2000.

[31] S. Takai, "Synthesis of maximally permissive and robust supervisors for prefix-closed language specifications," IEEE Transactions on Automatic Control, Vol. 47, pp. 132-136, 2002.

[32] S. Takai, "Maximizing robustness of supervisors for partially observed discrete event systems," Automatica, Vo. 40, pp.531-535, 2004.

[33] TTCT download site, " http://www.control.utoronto.ca/DES/ "

[34] C. H. Wang and S. Hashtrudi Zad, "Fault recovery in discrete-event systems using observer-based supervisors" IEEE Indicon Conference, Chennai, India, Dec. 2005

[35] B. C. Williams and P. P. Nayak, "Immobile robots: artificial intelligence in the new millennium," AI Magazine, Vol. 17, No.3, pp.16-35, Fall 1996

[36] B.C. Williams, M.D. Ingham, S.H. Chung, P.H. Elliott, "Model-based programming of intelligent embedded systems and robotic space explorers," Proceedings of the IEEE, Vol. 91, Issue 1, pp. 212 – 237, Jan. 2003.

[37] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," SIAM Journal on Control and Optimization, Vol. 25, No. 3, pp. 635-659, 1987.

[38] W. M. Wonham, *Supervisory Control of Discrete-Event Systems*, Control Group, Department of Electrical and Computer Engineering Department, University of Toronto, 2007, " http://www.control.utoronto.ca/DES/ ".