Robust Decentralized Supervisory Control of Discrete-Event Systems

Mohammad Rahnamaei

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 & Computer Engineering) at

Concordia University

Montréal, Québec, Canada

August 2009

Your file Votre référence
ISBN: 978-0-494-63111-9
Our file Notre référence
ISBN: 978-0-494-63111-9

ABSTRACT

Robust Decentralized Supervisory Control of Discrete-Event Systems

Mohammad Rahnamaei

In this thesis we study robust supervisory control of discrete event systems in two different settings. First, we consider the problem of synthesizing a set of decentralized supervisors when the precise model of the plant is not known, but it is known that it is among a finite set of plant models. To tackle this problem, we form the union of all possible behaviors and construct an appropriate specification, from the given set of specifications, and solve the conventional decentralized supervisory control associated with it. We also prove that the given robust problem has a solution if and only if this conventional decentralized supervisory control problem has a solution. In another setting, we investigate the problem of synthesizing a set of communicating supervisors in the presence of delay in communication channels, and call it Unbounded Communication Delay Robust Supervisory Control problem (UCDR-SC problem). In this problem, We assume that delay is unbounded but it is finite, meaning that any message sent from a local supervisor will be received by any other local supervisors after a finite but unknown delay. To solve this problem, we redefine the supervisory decision making rules, introduce a new language property called unbounded-communication-delay-robust (UCDR), and present a set of conditions on the specification of the problem. We also show that the new class of languages that is the solution to this problem has some interesting relations with other observational languages.

"One thing is certain and the rest is lies;

The Flower that once has blown for ever dies."

-Omar Khayyam

With my deepest gratitude,

I thank and dedicate this dissertation to my Mom and Dad:

Fereshteh & M.Hossein

Acknowledgements

Thanks are due first to my supervisors, Dr. Shahin Hashtrudi-Zad and the late Dr. Peyman Gohari. I am deeply indebted for their constant support and guidance throughout the work with this thesis. It is my greatest sorrow that Dr. Peyman Gohari is not amongst us, but I will never forget his vision, courage, and passion for life.

I thank my parents for believing in me and sacrificing while I studied. Both of them showed nothing but enthusiasm and excitement for my work, never once complaining or resenting. I thank them both and continue to count on them as my soft place to fall.

Lastly, thanks are due to my friends in Montreal, those who have supported me and have always been there for me.

Contents

List of Figures

Chapter 1

Introduction

The expanding use of highly complex technological systems in everyday life has given rise to new dynamic systems: computer and communication networks, automated manufacturing systems, and intelligent transportation systems to name a few. The dynamics of these systems are best characterized by the occurrence of some discrete events, such as hitting a keyboard, in an asynchronous fashion. This way, sending a packet through a communication channel is as much an *event* as turning that piece of communication device "on". This makes way for a better understanding of underlying system structure without using differential and difference equations that have been used for many decades. Moreover, for this class of systems, which appropriately named *discrete-event systems*, the modelling frameworks and mathematical tools that have been used to study time-driven precesses are inefficient. Discrete-event systems (DES for short) have been studied in different disciplines; mathematicians, computer scientists, and engineers have all added to the capabilities of discrete-event systems and made them powerful yet relatively young tools.

As our understanding of discrete-event systems grows, so does the size of the problems we face. To face these problems we need to adapt some of the concepts and

techniques that we have used for time-driven systems. For instance, there comes the problem of state explosion as the number of subsystems increases which calls for a *modular* solution, or an inherently distributed system will encourage one to think of *decentralized* design to solve the problem.

The doctorate thesis of P.J. Ramadge [1] under W.M. Wonham in 1983, was the first attempt to bring about two areas of 'control systems' and 'discrete mathematic' together, and the result of their work is what is known as the "Supervisory Control Theory of Discrete-Event Systems". Since then, many researchers contributed to this topic and broadened its domain to include topics such as fault recovery (e.g. [2; 3]), robustness (e.g. [4; 5]), and communication (e.g. [6; 7; 8]).

In this thesis we first consider the problem of robust supervisory control with the framework that was proposed by Lin [4]. The work of Saboori and Hashtrudi Zad [9] will be our starting point as it includes partial observation, which is inevitable in distributed systems, and also provides necessary and sufficient conditions for the existence of centralized supervisor that satisfies the given conditions. We try to extend the results of [9] to include the case where a set of decentralized supervisors is required rather than just one centralized supervisor. Then, we focus on the problem of delay in communication channels, specifically, the problem of synthesizing distributed supervisors in the presence of unbounded delay in communication channels. We show that the existing work on this subject does not fully capture the problem and propose a new language property which combined with other conditions form the necessary and sufficient conditions for the existence of a solution to this problem. The next section reviews some of the related and background works on these subjects.

1.1 Literature Review

1.1.1 Supervisory Control

In Ramadge-Wonham framework [1], a DES is modeled as the generator of a formal language and can be thought of as the set of trajectories (or behaviors) of the system (or plant). This generator models the 'uncontrolled behavior' of the system, and so a *supervisor* is an external agent whose task is to change and modify this behavior. Each run of the systems is modeled by a sequence of events executable by the plant, and is called a *string.* The set of all executable events is called the *alphabet.* This set is partitioned into two disjoint sets of *controllable* and *uncontrollable* events, and so the supervisor's task is to observe a string generated by the plant and to restrict its possible extensions by disabling a subset of controllable events. The desired behavior is called *specification* and is used by the supervisor to determine whether a controllable event should be disabled or not. The concept of *controllability* is introduced to solve the problem of synthesizing a supervisor that can implement a given specification. The necessary and sufficient conditions for the solavbility of this problem are presented by the authors in [1].

Lin and Wonham [10] (also Cieslak *et al.* [11]) partitioned the set of all executable events into two disjoint sets of *observable* and *unobservable* events, based on the assumption that some of the executable events might not be available for the supervisor to decide upon, either for the lack of appropriate sensor or the nature of the event (e.g. failure event), hence the term *partial observation.* A supervisor under partial observation of the system only sees observable events in any run (string) of the system and thus it is possible for two different strings to cause the same observation for the supervisor. Should a supervisor makes similar decisions for any two look alike strings, the supervisor is called *feasible.* The concept of *observability* is then introduced to

form a set of necessary and sufficient conditions to solve the problem of synthesizing a feasible supervisor that can implement a given specification under partial observation.

In another approach, Lin and Wonham investigated the problem of controlling distributed systems [12] and discussed that in such systems, no single supervisor is enough to generate the useful solution. And so, they argued that this type of systems require a *decentralized* solution, meaning more than one supervisor working together to achieve the given specification. Each of these individual supervisors (*local* supervisors) observes and controls part of the overall process and fusion of their individual decisions forms the control pattern. The authors considered the case where a set of specifications is given (each over a set of local events) that the associated local supervisor should achieve, and called the problem of synthesizing such a set of supervisors DSCOP, for *Decentralized Supervisory Control and Observation Problem*; later Rudie and Wonham called this type of problems LP, for *local problems* [13]. [12] presented sufficient conditions for the existence of a solution to DSCOP.

Cieslak *et al.* [11] worked on distributed systems but considered the case where a single specification is given over the whole set of events and the problem is synthesizing local supervisors such that behavior of the overall system under supervision precisely matches the given specification. [11] provided necessary and sufficient conditions for the existence of a solution to this problem. Later [13] called this class of problems GPZT, for *global problem with zero tolerance*, and presented another class of decentralized control problems which is more general than GPZT and called it GP, for *global problem.* GP is a synthesis problem with tolerance which asks for the existence of a set of decentralized supervisors such that the behavior of the system under supervision lies in some given range.

In [13] the authors argued that while LPs are enough to consider for manufacturing system problems, communication protocol synthesis problems require GPs to be

addressed fully. Therefore they presented a decentralized counterpart to the concept of observability, namely *coobservability*, and presented a set of necessary and sufficient conditions for the existence of a solution to GPZT. They also presented a condition for the solvability of GP but argued that their method for constructing supervisors "plays it too safe" by choosing a solution close to the lower end of the given range of specification, and that no largest solution in a given range exits.

Tsitsiklis [14] showed that observability of a specification can be checked in polynomial time but argued that no polynomial-time algorithm can be found to construct the supervisor implementing that specification. Rudie and Willems [15] extended the results of Tsitsiklis for coobservability property and showed that it can also be checked in polynomial time with respect to the number of system's states. Both of these results does not hold if the specification is given as a range and the question is whether there exists an observable (a coobservable) solution in a given range.

The problem with coobservability is that it is not preserved under union whereas controllability is preserved under union and thus a controllable sublanguage of a given specification can be found. Coobservability in the way that was introduced in [13] was proved to be closed under intersection. Many researchers worked based on these results but Prosser *et al.* [16] was the first to considered other fusion rules to combine the local supervisor's decisions, and later Yoo and Lafortune [17; 18] completed that work. [18] renamed conventional coobservability as *C&P coobservability*, for *Conjunctive and Permissive*, based on the fact that local supervisor's decisions are intersected to form global control action and thus the default control action for a supervisor with non-sufficient information is to "enable" an event. They presented the notion of *D&A coobservability*, for *Disjunctive and Antipermissive*, for an architecture that extracts global control actions by forming the union of local decisions and so requires the default control action of local supervisors to be disablement of an event. [18] Com-

bined these two methods to bring forward the most general class of coobservability, known as *C&P ∨ D&A coobservability*, which employs fusion by union for a subset of controllable events and fusion by intersection for the rest of the controllable events. [18] also showed that D&A coobservability and C&P ∨ D&A coobservability are not preserved under union or intersection.

Takai *et al.* [19] presented fixed-point based characterization of all the classes of coobservable languages, investigated their closure under intersection (resp. union), gave a formula for computing superlanguage (resp. sublanguage), and also provided upper (resp. lower) bound for their formula where the respective coobservable class is not preserved under intersection (resp. union). The authors' mathematical approach in [19] has lead to the introduction of four new classes of coobservable languages which are more restrictive than the three investigated by [18].

The common assumption that in a distributed systems a local supervisor's view of an event is fixed (i.e. it always observes or never observes an event) was challenged by Huang *et al.* [20]. They argued that a supervisor may observe only some occurrence of an event and derived necessary and sufficient conditions for solving this variant of the decentralized control problem. They assumed that observation of a particular event by a local supervisor is dependant on that supervisor's state and introduced an analogous coobservability notion, namely *state-based coobservability.* [20] argued that this idea is particularly useful in problems where observation of some events is communicated between agents.

1.1.2 Communicating Agents

As researchers were busy trying to extend the domain of decentralized control, some began exploring the idea of incorporating communication between local supervisors. [21] tries to shed some light on this subject by assuming that local supervisors commu-

nicate their state-estimates among themselves, and also comes up with a set of states where these communications should take place. In their approach, after coobservability fails and thus it becomes clear that no set of non-communicating supervisors could achieve the given specification, a set of *communication pairs* is identified which consists of a communication state and a string leading to that state. The *communication event* is an event that is observable to both supervisors, the sender and the receiver, but is only controlled by the sender, and later will be incorporated into the system structure. Measures are taken to insure *consistency* of communication and also an algorithm to find a minimal set of communication pairs is provided by the authors. Their method asks for a communication between agents 'at first opportunity' and this was the motivation for others to explore the idea further.

Barrett and Lafortune [22] propose a framework for communication between agents, and to this end they consider *extended traces* over plant events to model communication. They derive necessary and sufficient conditions for the existence of a communication policy that enables local supervisors to precisely achieve a given specification. In this process they identify two different cases when controllers do and do not anticipate future communication, and show that controllers that anticipate future communication achieve a strictly larger class of languages than the ones that do not anticipate future communication. Finally, they present an algorithm to find an optimal communication policy but argue that such an optimal solution is not unique. As [21], their work assumes zero-delay and lossless communication channels and also what is communicated between agents are state-estimates, but unlike [21], communication is *two-way broadcast* which means both supervisors exchange their local state-estimates when they initiate communication. Also the method used in [22] asks for a 'latest opportunity' communication between agents.

Other researchers such as Rudie, Lafortune, and Lin [7; 8; 23] model communi-

cating agents that instead of state-estimates, send occurrence of events among themselves. [7] considers only two communicating supervisors whose task is to distinguish between the states of their associated finite-state automaton and communicate their direct observation to one another. They also provide an algorithm which produces minimal communication pairs with a computational complexity that is exponential in the number of states of the two given finite-state automata. Wang *et al.* [23] considers a more general problem in which only the system model is given and the objective is that agents distinguish between a given set of system states for their unspecified monitoring, diagnosis, or control task. They further assume that in the system model no cycles other than self-loops exist, and derive a polynomial-time complexity (with respect to the number of system states) algorithm that computes a set of minimal communication pairs.

[8] continues the previous work in [7] and introduces a new problem, minimizing the set of communication pairs when some *essential transitions* should always be included in the set. The authors change the assumption of [7] pertaining agents' task but use some definitions of that work to solve the new problem. They argue that essential transitions problem is more general than state disambiguation problem in the sense that any state disambiguation problem could be solved by the method proposed for the essential transitions problem the reverse could not be done. [8], like [7], is restricted to two communicating agents and the question of how to identify the essential transitions for a specific problem is left unanswered.

1.1.3 Delay in Communication

"Delay in communication" has been addressed in several contexts. Balemi [24] investigats an input/output supervisory control problem in which the plant and supervisor work in harmony through a closed-loop connection, that may or may not be subject

to communication delay. [24] models the delay of a language as a shift to the right in the position of elements of the language, and derives conditions for the existence of a supervisor when the specification is given over the output event set.

Debouk *et al.* [25] considers the coordinated decentralized failure diagnosis problem when local sites communicate to a coordinator responsible for diagnosis. [25] assumes that communication delay causes out-of-order messages at the coordinator's site and argues that to achieve global ordering of these messages, one might either time-stamp each message for which synchronization of local clocks is needed, or alternatively design algorithm that order the messages arriving at the coordinator's site. The authors choose the second approach and present conditions on system structure under which failure diagnosis is eventually possible. [25] is restricted to two local sites when delay causes at most 'one-step out of order' messages in the coordinator site.

Ricker and Schuppen [26] present a model for failure detection in decentralized timed DES with an arbitrary communication delay in which communication of local clock values are used to restore the reordering of messages. This approach had been considered by the authors in [25] as "too constraining".

Tripakis in [27] proposes a class of languages called *jointly observable* and proves that decentralized observation problem is undecidable whereas centralized counterpart, i.e. checking joint observability w.r.t. one observer, is decidable. The proof of undecidability is by reduction of *Post's Correspondence* problem [28]. Tripakis also argues that observation is related to control in the sense that controllers should base their decisions on their observations and by reducing a decentralized control problem to checking joint observability, he suggests that decentralized control problem is undecidable as well. Interestingly, comparing joint observability with coobservability shows that these two classes of languages are incomparable. [29] extends the previous work and presents a hierarchy of control problems with communication delay and

provides a proof that the decentralized supervisor synthesis problem with unbounded communication delay is undecidable, while the same problem becomes decidable with bounded-delay assumption.

Sengupta and Tripakis [30] consider the problem of distributed fault diagnosis with unbounded delay and prove it to be undecidable. Qui *et al.* [31] investigates the same problem of distributed diagnosis with the assumption of unbounded delay, argues that the proposed property called *decentralized diagnosability* [30] does not completely capture distributed diagnosis problem, and proposes another property called $joint_{\infty}$*-diagnosability* which proves to be polynomially decidable.

In a different context, Park and Cho [32; 33; 34] investigate centralized (resp. decentralized) supervisor synthesis problem when delay can occur in sensor and actuators and propose a new property called *delay observability* (resp. *delay coobservability*) which is required for the existence of a solution. In their work, delay in sensors and actuators will cause some uncontrollable events to occur before proper control action is applied to the plant.

1.1.4 Robust Supervisory Control

Robust supervisory control has been studied in different frameworks. Cury and Krogh [35; 36] measure the performance in term of the largest possible language within specification. [36] assumes complete observation for the events and makes no specific assumption about the specification. [36] formulates robustness problem so that by maximizing the family of plants for which the system under supervision is within specification, one can achieve maximum robustness. [36] shows that this problem in general does not have a solution and only by restricting the specification such a maximum solution could be found and also proposes an algorithm for that. Takai [37; 38] extends the results of [36] to the partial observation case, and also removes the

restriction on the specification. [38] presents its results based on the permissiveness of each of the members of the family of plants and shows that under partial observation robustness could be maximized. [38] also proves that if every controllable event is observable, then the previous result also maximizes the permissiveness. Park and Lim [39] associate uncertainty with internal unobservable events of the system, and find necessary and sufficient conditions for the existence of a robust nonblocking supervisor, including the non-deterministic solutions.

The work in this thesis is based on the framework presented by Lin [4] that can be regarded as the "most natural" [40]. Lin assumes that the precise model of the plant is not known, but it is known that it is among a finite set of plant models. [4] formulates the robust problem considering partial observation and satisfying nonblocking condition for the resulting supervisor, but assumes a single design specification for all possible plant models. [4] further assumes that this specification is a subset of any of the marked languages generated by possible models, and forms the union of all possible behaviors to turn the robustness problem to a conventional supervisory control problem. The author then proves that the robust problem has a solution if and only if the conventional supervisory control problem regarding the union behavior and the specification has a solution. [41] extends this work by relaxing the assumption on specification only requiring it to be prefix-closed, and shows that necessary and sufficient conditions presented in [4] are still valid.

Bourdon *et al.* [5] extends the results of [4] by assuming a non-unique design specification, i.e. for every possible plant model there could be a separate specification. Nonblocking property is also addressed, but the partial observation assumption of [4] is changed to the full observation case. [5] synthesizes global specification based on the local specifications and proves that to guarantee a nonblocking solution for the robust problem a new property should be satisfied which they call it *nonconflicting* property.

The necessary and sufficient conditions for the existence of a robust nonblocking supervisor in an untimed DES are then presented and extended to the timed discrete-event systems (TDES).

Saboori and Hashtrudi Zad [9] show that the notion of nonconflicting in [40] is only necessary but not sufficient to guarantee a nonblocking solution to the robust problem. [9] also extends the work of Bourdon *et al.* to the partial observation case and presents another property, called *G-nonblocking* to replace nonconflicting property to ensure sufficiency. The authors in [9] present necessary and sufficient conditions for the existence of a solution to RNSC-PO, for *Robust Nonblocking Supervisory Control under Partial Observation*, and provides a formula to compute the maximally permissive solution of this problem in [42].

1.2 Thesis Contributions and Organization

In Chapter 2, we present mathematical background for our work, covering set theory, automata theory and supervisory control of discrete-event systems. We will define many of the notions that we have introduced throughout this chapter and use them in our main work.

In Chapter 3, we extend the existing solved robust centralized supervisory control problem to a decentralized setting and show that necessary and sufficient conditions for the existence of a solution to this new problem could be found. We show under which conditions a solution to *Robust Decentralized Supervisory Control* (RDSC for short) problem exists, and provide some insights as how to find this solution, although not specifically providing an algorithm for that. We provide an example that explains the difficulties in solving RDSC problems and another example to show that adding communication, without going into the details of using an specific framework, by itself

provides a much less restrictive (or larger in terms of languages) solution.

In Chapter 4, we assume that we have already incorporated communication between our local supervisors in its primitive form, not to minimize the communication either in terms of bits or the times that we communicate, but to investigate a practical hindrance: delay in communication channels. Much work has been done on finding different frameworks that a minimal communication could solve a particular problem, and we have reviewed some of them here, but most of them assume zero-delay communication channels. Tripakis [29] explicitly announces that with unbounded communication delay, decentralized supervisor synthesis problem is an undecidable problem. We believe that there is another approach into solving this problem. What Tripakis expects from a controller is not using a controller to its full potential, as it becomes clear when comparing the class of jointly observable languages with the class of coobservable languages, the two are incomparable. But the class of coobservable languages is commonly known for being the class of languages that can successfully be implemented by a set of local supervisors without communicating with each other. The question is why such a language can not be implemented by a set of supervisors that communicate with each other but through a channel that has unbounded delay; meaning that the messages will be delivered eventually but could take arbitrarily long before doing so. This question was our motivation for Chapter 4 and we show that indeed there is another class of languages that can be implemented in the presence of unbounded delay. We call these languages *UCDR* languages, for *Unbounded Communication Delay Robust*, and show that this class is strictly larger than the class of coobservable languages. We also show that the class of UCDR languages is incomparable with the class of jointly observable languages, and so the undecidability results of [29] can not be applied to that. Unfortunately we were not able to provide an algorithm that checks UCDR property, so at this point this property is not known

to be decidable, but the reduction to well known undecidable problems such as PCP [28] has also failed. It is left as a rewarding future work to look for an algorithm that checks UCDR property, or to prove it to be undecidable. We also redefine the supervisor's decision making patterns to allow anticipation for future communication in them, and call them UCDR supervisors. We show that under certain conditions a set of UCDR supervisors can be found that can achieve a given specification in the presence of unbounded delay, and prove that these conditions are necessary and sufficient.

In Chapter 5, we conclude this thesis and provide some directions for future work.

Chapter 2

Backgrounds and Preliminaries

2.1 Introduction

In this chapter we will briefly review some of the definitions and theories we will need in the following chapters.

2.2 Discrete-Event System (DES)

Individually, a *system* is best defined as a combination of components that act together to perform a function not possible with any of the individual parts [43]; *discrete* as individually distinct entities, and *event* as something that happens. "Discrete-event system" (DES for brevity) is an event-driven, in contrast to time-driven, discrete state space system in which the occurrence of events lead to state transitions. The behavior of such a system can be seen as a sequence of those discrete events that will cause its state transitions, so, if one thinks of a set of events as *alphabet* and a sequences of such events as *words*, then we say that the behavior of a DES is a *language*, the set of all sequences of events the DES can generate. *Automata*

theory will be used throughout this work, and although it existed from the viewpoint of theory of computation [44], its application in control systems originates with the doctorate thesis of P.J. Ramadge [1] under W.M. Wonham. Automata are intuitive, easy to use, and form a basic class of DES models, but lack structure and for this reason might lead to a very large state space [43]. Other modeling formalism, *Petri nets*, have more structure than automata models, but not with that much analytical power.

We proceed as follows. Section 2.3 presents some of the mathematical preliminaries which will be used throughout this work, and includes topics such as set theory and automata theory. Section 2.4 covers the theory of supervisory control of discrete-event systems. Section 2.5 introduces robust control theory in discrete-event systems, and includes some of the theorems that will be used in Chapter 3 for proving our main result.

2.3 Linguistic Preliminaries

2.3.1 Languages

Let Σ be a set of distinct symbols $\alpha, \beta, \ldots$ called an *alphabet*. Let Σ^+ denote the set of all finite sequences of symbols, $\sigma_1\sigma_2\ldots\sigma_k$ for $k \geq 1$ and $\sigma_i \in \Sigma$ $(i \in \mathcal{I} = \{1, \ldots, k\})$. Let ϵ represents the sequence with no symbol. Now,

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

We call each element of Σ^* a *string* over Σ, and ϵ the *empty string*. For $s \in \Sigma^*$, $|s|$ denotes the *length* of string s, and is defined according to

$$|s| = \begin{cases} 0 & \text{if } s = \epsilon \\ k & \text{if } s = \sigma_1 \ldots \sigma_k \in \Sigma^+ \end{cases}$$

Definition 2.3.1. *A* language *is any subset of* Σ^*.

Thus, $\emptyset$ (the empty set) and Σ are both languages.

Nerode equivalence [45; 44]: Let $L \subseteq \Sigma^*$ be an arbitrary language. The *Nerode equivalence relation on* Σ^* *with respect to* L is defined as follows. For $s, t \in \Sigma^*$: $s \equiv_L t$ if and only if

$$\forall u \in \Sigma^* : \quad su \in L \Leftrightarrow tu \in L$$

We write $||L||$ for the *index* of nerode equivalence relation $\equiv_L$.

Definition 2.3.2. *If* $||L|| < \infty$, *the language* L *is said to be* regular.

2.3.2 Operation on Languages

Concatenation [44]:

$$cat \ : \ \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

is defined according to (i) for $s \in \Sigma^*$, $\text{cat}(\epsilon, s) = \text{cat}(s, \epsilon) = s$, and (ii) for $s, t \in \Sigma^+$, $\text{cat}(s, t) = st$. Clearly ϵ is the unit element of concatenation. Also $|\text{cat}(s, t)| = |s| + |t|$.

If $tuv = s$ with $t, u, v \in \Sigma^*$, then

- t is called a *prefix* of s,
- u is called a *substring* of s, and
- v is called a *suffix* of s.

Prefix-closure [44]: Let $L \subseteq \Sigma^*$, then

$$\overline{L} := \{s \in \Sigma | \exists t \in \Sigma^*, st \in L\}$$

in other words, the prefix-closure of L is a language that contains all of the prefixes of the strings in L. If $L = \overline{L}$, we call L *prefix-closed.*

Prefix-closure of a language L keeps track of words in L, and so it is closely related to control problems. Notice that if $s, t \notin \overline{L}$ (i.e. $s, t \in \Sigma^* - \overline{L}$), then

$$(\forall \omega \in \Sigma^*)\ s\omega \notin L \quad \& \quad t\omega \notin L$$

Which means $\Sigma^* - \overline{L}$, if nonempty, is a single nerode cell, which we call the *dump cell.*

Projection [45]: Let $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, with $\Sigma_1^* \cap \Sigma_2^* \neq \emptyset$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, we can define *natural projection* $P_i : \Sigma^* \to \Sigma_i^*$ $(i = 1, 2)$, as follows,

$$P_i(\epsilon) := \epsilon,$$

$$P_i(\sigma) := \begin{cases} \epsilon & \text{if } \sigma \notin \Sigma_i \\ \sigma & \text{if } \sigma \in \Sigma_i \end{cases}$$

$$P_i(s\sigma) := P_i(s)P_i(\sigma) \quad \forall s \in \Sigma^*, \forall \sigma \in \Sigma$$

Natural projection of a string $P_i(s)$ erases all the occurrences of events σ not in Σ_i. Inverse image function of P_i, $P_i^{-1} : \mathcal{P}(\Sigma_i^*) \to \mathcal{P}(\Sigma^*)$, is

$$P_i^{-1}(H) := \{s \in \Sigma^* | P_i(s) \in H\} \quad H \subseteq \Sigma_i^*$$

when $\mathcal{P}(\Sigma^*)$ is the set of all the subsets of Σ^*, called *power set* of Σ^*.

Synchronous product [45; 44]: Let $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, and $\Sigma = \Sigma_1 \cup \Sigma_2$. Define the *synchronous product* $L_1 \| L_2 \subseteq \Sigma^*$ according to

$$L_1 \| L_2 := P_1^{-1} L_1 \cap P_2^{-1} L_2$$

Thus, a string $s \in L_1 \| L_2$ if and only if its projection on event set Σ_1 is in L_1, and its projection on event set Σ_2 is in L_2; i.e. $P_1(s) \in L_1$ and $P_2(s) \in L_2$.

2.3.3 Automata and Generators

Consider the following 5-tuple

$$\mathbf{A} = (X, \Sigma, \xi, x_0, X_m)$$

with Σ being the alphabet, X a nonempty set, $x_0 \in X$, $X_m \subseteq X$, and ξ a function,

$$\xi \ : \ X \times \Sigma \rightarrow X$$

$\mathbf{A}$ is then called a *deterministic automaton over the alphabet* Σ. X is the set of states, x_0 is the *initial state*, X_m is the set of *marker states*, and ξ is the *transition function.* For convenience, we extend ξ from domain $X \times \Sigma$ to $X \times \Sigma^*$ according to,

$$\begin{aligned} \xi(x, \epsilon) &= x, && x \in X \\ \xi(x, s\sigma) &= \xi(\xi(x, s), \sigma)). && x \in X, s \in \Sigma^*, \sigma \in \Sigma \end{aligned}$$

Given an automaton $\mathbf{A}$, the language $L \subseteq \Sigma^*$ *recognized* by $\mathbf{A}$ is

$$L := \{s \in \Sigma^* | \xi(x_0, s) \in X_m\}$$

$\mathbf{A}$ is also called a *recognizer* for L.

Generator [45]: *Generator* is an automaton, in which at each state only a subset of all events can occur, therefore the transition function of a generator is a *partial function*, whereas the transition function of an automaton is a complete function. Consider the following 5-tuple

$$\mathbf{G} = (Y, \Sigma, \delta, y_0, Y_m)$$

with $\delta \ : \ Y \times \Sigma \rightarrow Y$. δ is defined at each state $y \in Y$, only for a subset of elements $\sigma \in \Sigma$. We write $\delta(y, \sigma)!$ to state δ is defined at y for σ. Obviously δ is a partial

function. Extension of δ to domain $Y \times \Sigma^*$ is done by,

$$\delta(y, \epsilon) = y,$$
$$\delta(y, s\sigma) = \delta(\delta(y, s), \sigma)), \quad \text{if} y' := \delta(y, s)!, \delta(y', \sigma)!$$

Closed behavior [45]: The set of strings $s \in \Sigma^*$ for which $\delta(y_0, s)!$, is called *closed behavior*, $L(\mathbf{G}) = \{s \in \Sigma^* | \delta(y_0, s)!\}$. Closed behavior represents all the possible paths, starting from initial state, following along the state transition graph, and is prefix-closed by definition.

Marked behavior [45]: The set of strings $s \in L(\mathbf{G})$ for which $y = \delta(y_0, s) \in Y_m$, is called *marked behavior*, $L_m(G) = \{s \in L(\mathbf{G}) | \delta(y_0, s) \in Y_m\}$. Marked behavior represents the subset of all those paths which end at a marker state in the state transition graph, and need not be prefix-closed. By definition, $L_m(\mathbf{G}) \subseteq L(\mathbf{G})$.

Reachability [45; 43]: A state $y \in Y$ is called *reachable* if there is a path from the initial state ending at it; i.e. there exists $s \in L(\mathbf{G})$ such that $y = \delta(y_0, s)$ provided $\delta(y_0, s)!$. The *reachable* subset of $\mathbf{G}$ is

$$Y_r = \{y \in Y | (\exists s \in \Sigma^*) \delta(y_0, s) = y\}.$$

and $\mathbf{G}$ is reachable if every state $y \in Y$ is reachable; i.e. $Y_r = Y$.

Coreachability [45; 43]: A state $y \in Y$ is called *coreachable* if there is a path starting from it, and ending at a marker state; i.e. there exists $s \in L(\mathbf{G})$ such that $y' = \delta(y, s) \in Y_m$. The *coreachable* subset of $\mathbf{G}$ is

$$Y_{cr} = \{y \in Y | (\exists s \in \Sigma^*) \delta(y, s) \in Y_m\}.$$

and $\mathbf{G}$ is coreachable if every state $y \in Y$ is coreachable; i.e. $Y_{cr} = Y$.

Nonblocking [45; 43]: Generator $\mathbf{G}$ is said to be *nonblocking* if every reachable state is also coreachable, i.e.

$$L(\mathbf{G}) = \overline{L_m(\mathbf{G})}$$

which means that, any string in $\mathbf{G}$ is a prefix of a marker string of $\mathbf{G}$. $\mathbf{G}$ is said to be *blocking* if $\overline{L_m(\mathbf{G})} \subset L(\mathbf{G})$.

When a generator is blocking, it has a (reachable) *deadlock* or a *livelock*. Deadlock can happen where there exists a non-marker state $y \in Y$ in generator $\mathbf{G}$ with no transition out of y. Livelock can happen where there exists a set of non-marker states in $\mathbf{G}$ which are strongly connected to each other, but there is no transition going out of the set.

Trim [45; 43]: $\mathbf{G}$ is said to be *trim* if it is both reachable and coreachable. It is important to note that $\mathbf{G}$ being trim implies $\mathbf{G}$ being nonblocking, but the converse is not true: a nonblocking generator with some non-reachable states is an example.

2.3.4 Operation on Automata

Consider two generators $\mathbf{G_1} = (Y_1, \Sigma_1, \delta_1, y_{01}, Y_{m1})$, and $\mathbf{G_2} = (Y_2, \Sigma_2, \delta_2, y_{02}, Y_{m2})$, and further assume $\mathbf{G_1}$ and $\mathbf{G_2}$ are reachable, but not necessarily coreachable. We impose no restriction on Σ_1 and Σ_2.

Synchronous product [45] (Parallel composition [43]): In the definition of synchronous product in Section 2.3.2, let $L_1 = L_m(\mathbf{G_1})$ and $L_2 = L_m(\mathbf{G_2})$. Then it is easy to see synchronous product forces two generators to 'agree upon' executing common events, i.e. those events in $\Sigma_1 \cap \Sigma_2$, while it puts no constraint on 'private events' that can be executed whenever possible. In this case, the synchronous product of $\mathbf{G_1}$ and $\mathbf{G_2}$, $\mathbf{G_1} \| \mathbf{G_2}$, is the reachable part of generator $(Y', \Sigma', \delta', y_0', Y_m')$, where

$Y' = Y_1 \times Y_2$, $\Sigma' = \Sigma_1 \cup \Sigma_2$, $y'_0 = (y_{01}, y_{02})$, $Y'_m = Y_{m1} \times Y_{m2}$, with

$$\delta'((y_1, y_2), \sigma) = \begin{cases} (\delta_1(y_1, \sigma), \delta_2(y_2, \sigma)) & \sigma \in \Sigma_1 \cup \Sigma_2, \delta_1(y_1, \sigma)!, \text{and } \delta_1(y_1, \sigma)! \\ (\delta_1(y_1, \sigma), y_2) & \sigma \in \Sigma_1 \backslash \Sigma_2, \delta_1(y_1, \sigma)! \\ (y_1, \delta_2(y_2, \sigma)) & \sigma \in \Sigma_2 \backslash \Sigma_1, \delta_2(y_2, \sigma)! \\ \text{undefined} & \text{otherwise} \end{cases}$$

And therefore, the languages resulting from the synchronous product are

$$L(\mathbf{G_1} \| \mathbf{G_2}) = P_1^{-1}(L(\mathbf{G_1})) \cap P_2^{-1}(L(\mathbf{G_2})),$$
$$L_m(\mathbf{G_1} \| \mathbf{G_2}) = P_1^{-1}(L_m(\mathbf{G_1})) \cap P_2^{-1}(L_m(\mathbf{G_2})).$$

Shuffle product [45]: In the special case where $\Sigma_1 \cap \Sigma_2 = \emptyset$, we use the term *Shuffle product* in place of synchronous product to emphasize the disjointness of alphabets. Therefore, one can construct the shuffle product of $\mathbf{G_1}$ and $\mathbf{G_2}$ by asynchronously generating $\mathbf{G_1}$ and $\mathbf{G_2}$.

Meet [45] (Product [43]): The *Meet* of $\mathbf{G_1}$ and $\mathbf{G_2}$, $\mathbf{G_1} \times \mathbf{G_2}$, is the reachable part of generator $(Y'', \Sigma'', \delta'', y''_0, Y''_m)$, where $Y'' = Y_1 \times Y_2$, $\Sigma'' = \Sigma_1 \cap \Sigma_2$, $y''_0 = (y_{01}, y_{02})$, $Y''_m = Y_{m1} \times Y_{m2}$, with

$$\delta''((y_1, y_2), \sigma) = \begin{cases} (\delta_1(y_1, \sigma), \delta_2(y_2, \sigma)) & \delta_1(y_1, \sigma)!, \text{and } \delta_1(y_1, \sigma)! \\ \text{undefined} & \text{otherwise} \end{cases}$$

$\mathbf{G_1} \times \mathbf{G_2}$ keeps track of all the strings that can be generated by $\mathbf{G_1}$ and $\mathbf{G_2}$ in common. The following can be readily verified.

$$L(\mathbf{G_1} \times \mathbf{G_2}) = L(\mathbf{G_1}) \cap L(\mathbf{G_2}),$$
$$L_m(\mathbf{G_1} \times \mathbf{G_2}) = L_m(\mathbf{G_1}) \cap L_m(\mathbf{G_2}).$$

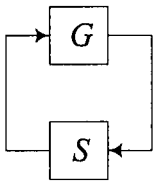

Figure 2.1: System under supervision

2.4 Supervisory Control Theory of DES

What we have seen so far is a DES modelled by a generator, $\mathbf{G}$, that executes its events according to its transition function. This "uncontrolled behavior" is not usually satisfactory, and thus a means to contain the behavior of $\mathbf{G}$ in the "acceptable range" must be presented. In order to modify this behavior, we introduce a *supervisor*; we denote supervisor by S or V. Figure 2.1 shows the schematic of a system under supervision. "Acceptable range" will be called *specification*, and is always a subset of $L(\mathbf{G})$. There are different reasons why we might need to restrict the behavior of $\mathbf{G}$: It might contain strings that violate some physical constraints, like using shared resources; it might contain some states that are not desirable, for instance states where $\mathbf{G}$ blocks; or it might even be some conditions that we impose on the DES, such as generating events in some specific order only. In this work E or K will be used to denote the language of specification. Then the control problem will be how V interacts with $\mathbf{G}$: V observes some, possibly all, of the events that $\mathbf{G}$ executes. It then informs $\mathbf{G}$ of the set of events that are allowed to happen at the current state. This set of events is a subset or equal to the set of events that are feasible at that state. Note that control exertion of a supervisor V is limited in two ways: by the set of events V can observe, and by the the set of events V can disable.

2.4.1 Controllability and Supervision

Let the structure of a DES to be controlled be that of a (nonempty) generator,

$$\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$$

with Σ as a finite alphabet or a set of *event labels*, Q as the state set (at most countable), δ as the (partial) transition function, q_0 as the initial state, and $Q_m \subseteq Q$ as the set of marker states, as usual. The pair of languages $L(\mathbf{G})$ and $L_m(\mathbf{G})$ represent the closed behavior and the marked behavior of $\mathbf{G}$, respectively. Let $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where

- Σ_c is the set of *controllable* events, and
- Σ_{uc} is the set of *uncontrollable* events

and the two are disjoint from one another. The distinction between controllable and uncontrollable events arises from physical limitation, modeling limitation, or simply choice. In any case, the supervisor can only exert control over the set of controllable events, and those events in the uncontrollable event set should be allowed to be executed whenever possible. Thus it is convenient to adjoin all the uncontrollable events with the subset of controllable events to be enabled at each state. We call each such subset of events, a *control pattern*. Thus, a supervisor is an *agent* that specifies the control pattern at each state. The set of all control patterns will then be

$$\Gamma = \{\gamma \in \mathcal{P}(\Sigma) | \gamma \supseteq \Sigma_{uc}\}$$

A *supervisory control for* $\mathbf{G}$ is any map $V : L(\mathbf{G}) \rightarrow \Gamma$. We write $V/\mathbf{G}$ to denote "$\mathbf{G}$ under the supervision of V". The closed behavior of $V/\mathbf{G}$ is defined recursively as,

1. $\epsilon \in L(V/\mathbf{G})$

2. $(s \in L(V/\mathbf{G}))$, $(\sigma \in V(s))$, and $(s\sigma \in L(\mathbf{G})) \Leftrightarrow (s\sigma \in L(V/\mathbf{G}))$

3. No other strings belong to $L(V/\mathbf{G})$.

From the definition $L(V/\mathbf{G})$ is nonempty and close, and always $\{\epsilon\} \subseteq L(V/\mathbf{G}) \subseteq L(\mathbf{G})$. The marked behavior of $V/\mathbf{G}$ comprises of all the marker states of $L_m(\mathbf{G})$ that 'survive' under supervision, or formally

$$L_m(V/\mathbf{G}) = L(V/\mathbf{G}) \cap L_m(\mathbf{G})$$

Marked behavior could be empty and thus always $\emptyset \subseteq L_m(V/\mathbf{G}) \subseteq L_m(\mathbf{G})$. We say V is nonblocking for $\mathbf{G}$ if

$$\overline{L_m(V/\mathbf{G})} = L(V/\mathbf{G}).$$

For practical reasons we might prefer a representation of a supervisor in the form of a generator. Let V be a nonblocking supervisor with $L_m(V/\mathbf{G}) = K$, and $L(V/\mathbf{G}) = \overline{K}$ where $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$. Let S be an arbitrary generator. We say S implements V where

$$K = L_m(S) \cap L_m(\mathbf{G}), \quad \overline{K} = L(S) \cap L(\mathbf{G})$$

In this fashion, the closed-loop system is represented by $S/\mathbf{G} = meet(S, \mathbf{G})$ as specified in Section 2.3.4, and the languages resulting from this construction have the following property.

$$L_m(S/\mathbf{G}) = L_m(V/\mathbf{G}), \quad L(S/\mathbf{G}) = L(V/\mathbf{G})$$

Some of the terminologies associated with 'supervisor' are as follows.

- *Admissible*: A supervisor V is admissible if it never disables a feasible uncontrollable event. Since our definition of supervision allows uncontrollable events to be executed whenever possible, our supervisors are always admissible.

- *Proper*: A supervisor V is proper for $\mathbf{G}$ if it is nonblocking.

As discussed earlier, control problems require construction of a (preferably proper) supervisor V for $\mathbf{G}$ such that either $L(V/\mathbf{G})$ or $L_m(V/\mathbf{G})$ exhibits some desirable behavior. This desirable behavior, or specification, is also called *legal language.*

Definition 2.4.1. *[45] Nonblocking Supervisory Control problem [NSC]: Given the plant* $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$*, and nonempty legal language* $E \subseteq L_m(\mathbf{G})$*, find a nonblocking supervisor* V *such that*

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

Note that, in general there are many solutions to the NSC problem; thus we form

$$\mathbb{V} = \{V \mid V \text{ solves the NSC problem.}\}$$

and say that a supervisor $V \in \mathbb{V}$ is *maximally permissive* if and only if

$$\forall V' \in \mathbb{V}. \quad L_m(V'/\mathbf{G}) \subseteq L_m(V/\mathbf{G}),\ L(V'/\mathbf{G}) \subseteq L(V/\mathbf{G}).$$

We need two more definitions in order to characterize the solution to NSC problem.

Definition 2.4.2. *[46]* ***Controllability****: A language* $K \subseteq \Sigma^*$ *is controllable with respect to* $\mathbf{G}$ *if*

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

In other words, K is controllable if the continuation of its prefixes by an uncontrollable event stays in prefix-closure of K.

Definition 2.4.3. *[46]* $L_m(\mathbf{G})$***-closedness****: Let* $K \subseteq L_m(\mathbf{G}) \subseteq \Sigma^*$*. The language* K *is* $L_m(\mathbf{G})$*-closed if*

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

In other words, K is $L_m(\mathbf{G})$-closed if it contains every one of its prefixes that also belong to $L_m(\mathbf{G})$.

Theorem 2.4.4. *[46] Let $K \subseteq E$ be a nonempty language, $\mathbf{G}$ and E be the same as in NSC problem. There exists a nonblocking supervisor V for $\mathbf{G}$, such that $L_m(V/\mathbf{G}) = K$ if and only if*

1. *K is controllable with respect to $\mathbf{G}$, and*

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

Theorem 2.4.4 shows that any subset of E that is controllable and $L_m(\mathbf{G})$-closed is a solution to NSC problem, and so the maximally permissive solution for the NSC problem is the 'largest' subset of E with those properties. However, this theorem does not specify how to find that 'largest' subset. As *lattice theory* and its application in DES is out of the scope of this work, we will only present the following propositions without going into the details.

Proposition 2.4.5. *[46] $\mathcal{C}(E)$ (the set of all controllable sublanguages of E) is nonempty and closed under arbitrary unions. In particular, $\mathcal{C}(E)$ contains a (unique) supremal element, which we denote by $sup\mathcal{C}(E)$.*

Proposition 2.4.6. *[46] Let $E \subseteq \Sigma^*$ be $L_m(\mathbf{G})$-closed. Then $sup\mathcal{C}(E)$ is $L_m(\mathbf{G})$-closed.*

2.4.2 Observability and Supervision

So far we have assumed that the supervisor can 'see' or 'observe' all the events that the plant $\mathbf{G}$ executes. However, it is more realistic to assume only a subset of all event labels can actually be observed by the supervisor. Limitation in the number of sensors in the plant or the distributed nature of the plant that causes some events not

to be available at every location, are among the reasons one might consider *partial observation* versus *full observation.* In general, this subset of events has no relation to the subset of controllable events, and as such an unobservable event could be controllable, or an uncontrollable event could be observable, and so on. Formally, Σ is partitioned into two disjoint subsets,

$$\Sigma = \Sigma_o \cup \Sigma_{uo}$$

where,

- Σ_o is the set of *observable* events
- Σ_{uo} is the set of *unobservable* events

In this setting, a supervisor is an agent that observes only a subset of events generated by the plant and then specifies the control pattern. A supervisor's view of the string can be captured by applying natural projection onto the set Σ_o, where P effectively erases all unobservable events. We extend the domain of natural projection to accept languages in the obvious fashion, in particular, we write $P(L(\mathbf{G}))$ to denote the supervisor's view of the sequences of events generated by the plant $\mathbf{G}$. Note that due to the presence of P, it is possible for two strings s_1 and s_2 to have the same projection, i.e. $P(s_1) = P(s_2)$, and the supervisor should be designed in a way that it issues the same control pattern; thus the following terminology for feasible supervisors

- *Feasible* [13]: Supervisor V is feasible if $\forall s, s' \in L(\mathbf{G})$:

$$Ps = Ps' \Rightarrow V(s) = V(s')$$

Informally, feasibility ensures that the supervisor only changes its control action after the occurrence of an observable event. From now on, we restrict ourselves to feasible supervisors even if it is not mentioned specifically. It is also a common assumption

that when an observable event occurs, the control pattern update is *instantaneous.* It is important that this update occurs before any unobservable event could happen, as we may wish to change the controlling action for some of these unobservable events. A control problem involving partial observation is as follows.

Definition 2.4.7. *[10] Nonblocking Supervisory Control and Observation Problem [NSCO]: Given the plant* $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$*, subsets* $\Sigma_c, \Sigma_o \subseteq \Sigma$*, and nonempty legal language* $E \subseteq L_m(\mathbf{G})$*, find a nonblocking supervisor* V *such that*

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

To characterize the solutions to NSCO problem we need to define the following notion.

Definition 2.4.8. *[10; 45]* ***Observability****: Let* $P : \Sigma^* \to \Sigma_o^*$ *be the natural projection. A language* K *is* $(\mathbf{G}, P)$*-observable if*

$$\forall s, s' \in \Sigma^* \; \forall \sigma \in \Sigma.\; s\sigma \in \overline{K} \wedge s' \in \overline{K} \wedge s'\sigma \in L(\mathbf{G}) \wedge Ps = Ps' \Rightarrow s'\sigma \in \overline{K}$$

Informally, observability ensures that P preserves enough information to prevent disputed membership of strings in $\overline{K}$ after occurrence of an event σ. So, if two strings 'looks the same', any control pattern that will be applied to one should also be applied to the other.

Theorem 2.4.9. *[10; 11] Let* $K \subseteq E$ *be a nonempty language,* $\mathbf{G}$ *and* E *be the same as in NSCO problem. There exists a nonblocking supervisor* V *for* $\mathbf{G}$*, such that* $L_m(V/\mathbf{G}) = K$ *if and only if*

1. K *is controllable with respect to* $\mathbf{G}$*,*

2. *K is $(\mathbf{G}, P)$-observable, and*

3. *K is $L_m(\mathbf{G})$-closed.*

Theorem 2.4.9 shows that any subset of E that is controllable, observable, and $L_m(\mathbf{G})$-closed, is a nominee for being the maximally permissive solution to NSCO, but again does not provide a means to find this largest subset. Following the method of the previous section, define

$$\mathcal{O}(E) = \{K | K \subseteq E \wedge K \text{ is } (\mathbf{G}, P)\text{-observable}\}$$

The difficulty of dealing with observability is that it is not preserved under union, and thus $\mathcal{O}(E)$ does not have a (unique) supremal element. As a suboptimal solution to NSCO problem, we define another property called *normality.*

Definition 2.4.10. *[12]* ***Normality****: A language $K \subseteq M \subseteq \Sigma^*$ is said to be (M, P)-normal if*

$$K = M \cap P^{-1}(P(K))$$

Note that inclusion $K \subseteq M \cap P^{-1}(P(K))$ is automatic, but the converse is not. In other words, a normal language can be constructed by the knowledge of its projection and the structure of the plant, while in general this process yields a larger language than K. Let $E \subseteq M$. Define

$$\mathcal{N}(E; M) = \{K | K \subseteq E \wedge K \text{ is } (M, P)\text{-normal}\}$$

$\mathcal{N}(E; M)$ enjoys the following property.

Proposition 2.4.11. *[45] The class of languages $\mathcal{N}(E; M)$ is nonempty and closed under arbitrary unions and intersections. In particular $\mathcal{N}(E; M)$ contains a (unique) supremal element, denoted by $sup\mathcal{N}(E; M)$, and defined as follows:*

$$sup\mathcal{N}(E; M) = E - P^{-1}P(M - E)$$

From the viewpoint of a supervisor, the fundamental difference between a $(\mathbf{G}, P)$-observable languages and a closed $(L(\mathbf{G}), P)$-normal, is that with a normal language there would be no controllable unobservable event that causes a sting to exit from legal language, while in general, this is not the case for an observable language. It is simply enough to look at the projection of an evolving string s, in a normal language, to tell if and when the string exists the legal language. Normality is obviously a stronger property than observability. Interestingly, there is an intermediate condition between observability and normality, called *strong observability*, but it is not in the scope of this work and will not be discussed here (see [47]). Let

$$\mathcal{C}(E) = \{K \mid K \subseteq E \wedge K \text{ controllable w.r.t. } \mathbf{G}\}$$
$$\overline{\mathcal{N}}(E; L(\mathbf{G})) = \{K \mid K \subseteq E \wedge \overline{K} \text{ is } (L(\mathbf{G}), P)\text{-normal}\}$$
$$\mathcal{R}(E) = \{K \mid K \subseteq E \wedge K \text{ is } L_m(\mathbf{G})\text{-closed}\}$$

To characterize the solution of NSCO problem we present the following class of languages,

$$\mathcal{S}(E) = \mathcal{C}(E) \cap \overline{\mathcal{N}}(E; L(\mathbf{G})) \cap \mathcal{R}(E)$$

$\mathcal{S}(E)$ enjoys the following closure property.

Proposition 2.4.12. *[12; 47] $\mathcal{S}(E)$ is nonempty and closed under arbitrary unions, so that sup$\mathcal{S}(E)$ exists in $\mathcal{S}(E)$.*

sup$\mathcal{S}(E)$ gives a suboptimal solution to NSCO problem.

2.4.3 Coobservability and Supervision

Up until now, we have discussed only those control problems where a 'single' supervisor should take actions to achieve the given specification. While in some systems this assumption works perfectly, in some other systems, most notably communication and

computer networks, no single supervisor is likely to be considered a useful solution, and therefore, we rather require a set of supervisors, which will be called hereinafter *decentralized supervisors*, each of which only controls and observes a subset of the system. We note that the difference between this control architecture called *decentralized control*, and *modular control* [48], lies in the fact that in the decentralized control architecture, individual supervisors may be partial observers and their respective sets of controllable events are not all the same.

In decentralized control, as in modular control, the net control action is the intersection of the control actions taken by each supervisors. So we first have to define the control action of each supervisor. We have a set of n supervisors, and associated with them, are the sets $\Sigma_{i,c}$, $\Sigma_{i,o}$, the controllable and the observable event sets corresponding to supervisor V_i, respectively. Natural projection corresponding to supervisor V_i is

$$\forall i \in \mathcal{I} = \{1, \ldots, n\}. \;\; P_i : \Sigma^* \rightarrow \Sigma_{i,o}^*$$

Then, the sets of controllable events and observable events are

$$\Sigma_c = \bigcup_{i \in \mathcal{I}} \Sigma_{i,c},$$

$$\Sigma_o = \bigcup_{i \in \mathcal{I}} \Sigma_{i,o}$$

Given the supervisor V_i whose domain is $P_i(L(\mathbf{G}))$, $\widetilde{V}_i$ denotes the supervisor whose domain is $L(\mathbf{G})$. It takes the same control action as V_i for all $\sigma \in \Sigma_{i,c}$, and enables all $\sigma \in \Sigma \backslash \Sigma_{i,c}$. The supervisor V_i is called a *local* supervisor, while $\widetilde{V}_i$ is its *global extension.* With a slight abuse of notation we write V_i in place of $\widetilde{V}_i$ where the extension of domain is obvious.

Let V_1 and V_2 be proper supervisors for $\mathbf{G}$. To exercise control over the plant, the supervisors are fused together in a *conjunctive* fashion, written $V_1 \wedge V_2$, as the

generator of the product

$$V_1 \wedge V_2 = V_1 \times V_2 = meet(V_1, V_2)$$

The resulting behavior is described by the languages $L(V_1 \wedge V_2/\mathbf{G})$ and $L_m(V_1 \wedge V_2/\mathbf{G})$ as follows.

$$L(V_1 \wedge V_2/\mathbf{G}) = L(V_1/\mathbf{G}) \cap L(V_2/\mathbf{G}),$$
$$L_m(V_1 \wedge V_2/\mathbf{G}) = L_m(V_1/\mathbf{G}) \cap L_m(V_2/\mathbf{G}).$$

It is easy to verify that $V_1 \wedge V_2$ only enables an event σ if it is enabled by both V_1 and V_2. Note that the choice of *two* supervisors was for simplicity, and all the resulting properties can be extended to any fixed number of supervisors.

Definition 2.4.13. *[13] Nonblocking Decentralized Supervisory Control Problem [NDSC]: Given the plant* $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, *subsets* $i \in \mathcal{I} = \{1, \ldots, n\}$. $\Sigma_{i,c}, \Sigma_{i,o} \subseteq \Sigma$, *and nonempty legal language* $E \subseteq L_m(\mathbf{G})$, *find a set of nonblocking supervisors* V_i $(\forall i \in \mathcal{I})$ *such that*

1. $L_m(\wedge_{i\in\mathcal{I}} V_i/\mathbf{G}) \subseteq E$
2. $L(\wedge_{i\in\mathcal{I}} V_i/\mathbf{G}) = \overline{L_m(\wedge_{i\in\mathcal{I}} V_i/\mathbf{G})}$.

We define the decentralized version of observability property in the following manner.

Definition 2.4.14. *[13]* ***Coobservability****: Let* $P : \Sigma^* \to \Sigma_{i,o}^*$ $(\forall i \in \mathcal{I})$ *be the natural projection for each supervisor. A language* K *is* $(\mathbf{G}, P_1, \ldots, P_n)$*-observable if*

$\forall s, s', s'' \in \Sigma^*$ *such that* $P_i s = P_i s' \wedge P_j s = P_j s''$ $(i \neq j)$, *we have:*

$$\forall \sigma \in \Sigma_{i,c} \cap \Sigma_{j,c}. \quad s'\sigma \in \overline{K} \wedge s \in \overline{K} \wedge s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{K}$$

$$\textit{or} \quad s''\sigma \in \overline{K} \wedge s \in \overline{K} \wedge s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{K} \qquad \text{conjunct 1}$$

$$\forall \sigma \in \Sigma_{i,c}/\Sigma_{j,c}. \quad s'\sigma \in \overline{K} \wedge s \in \overline{K} \wedge s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{K} \qquad \text{conjunct 2}$$

$$\forall \sigma \in \Sigma_{j,c}/\Sigma_{i,c}. \quad s''\sigma \in \overline{K} \wedge s \in \overline{K} \wedge s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in \overline{K} \qquad \text{conjunct 3}$$

Informally, it is not enough to require any one of the supervisors knows what action to take in all instances; and it is too much to require that all of the supervisors always know when to disable an event. Instead, coobservability employs the policy of "pass the buck", in the sense that if continuing membership of a string in $\overline{K}$ after occurrence of an event σ is disputed for a supervisor, then there is at least one supervisor which can control σ and also tell without ambiguity whether its occurrence is legal or not. In such an event, the former supervisor simply 'enables' the event and "passes the buck" to the other supervisor. Coobservability guarantees that this policy does not allow the occurrence of an illegal string. The following theorem characterizes the solution of NDSC problem in the case of $n = 2$.

Theorem 2.4.15. *[13] Let* $K \subseteq E$ *be a nonempty language,* $\mathbf{G}$ *and* E *be the same as in NDSC problem. There exist nonblocking supervisors* V_1 *and* V_2 *for* $\mathbf{G}$, *such that* $L_m(V_1 \wedge V_2/\mathbf{G}) = K$ *if and only if*

1. K *is controllable with respect to* $\mathbf{G}$,
2. K *is* $(\mathbf{G}, P_1, P_2)$*-coobservable, and*
3. K *is* $L_m(\mathbf{G})$*-closed.*

Theorem 2.4.15 shows that any controllable, coobservable, and $L_m(\mathbf{G})$-closed subset of E is a solution for NDSC problem. However, when the legal language E itself does not fall into this category, its largest subset with the aforementioned properties must be found if one desires to construct the maximally permissive set of supervisors. This proves to be almost an impossible job. Coobservability, similar its centralized counterpart observability, is not *well-behaved.* Consider the following set:

$$\mathcal{C}o(E) = \{K | K \subseteq E, K \text{ is prefix-closed, and coobservable.}\}$$

It has been shown that $\mathcal{C}o(E)$ is nonempty and closed under arbitrary intersections [49]. Also, the set of prefix-closed and controllable languages containing a given language is nonempty and closed under arbitrary intersections [10]. Therefore, the following class of languages

$$\mathcal{CC}o(M) = \{K | K \supseteq M, K \text{ is prefix-closed, controllable, and coobservable.}\}$$

is nonempty and closed under arbitrary intersections, and more importantly it contains a (unique) infimal element [13]. If the given specification is a range instead of one language, i.e. $A \subseteq L(V_1 \wedge V_2/\mathbf{G}) \subseteq E$, then one could form $\mathcal{CC}o(A)$ and check if its infimal is still in E. Therefore the following theorem for the case when specification is given as a range states,

Theorem 2.4.16. *If A is nonempty then NDSC is solvable if and only if*

$$inf\mathcal{CC}o(A) \subseteq E.$$

The actual formula for computing this infimal element is given in [13]. However, this method does not yield a maximally permissive solution; in fact, it can be shown that no largest controllable and coobservable language exists within a given range, so no unique maximally permissive solution exists. Also while Theorem 2.4.16 provides

the condition for the existence of solution to NDSC problem, it requires different form of specification than our original problem.

It is worthwhile to note that coobservability can be tested in polynomial time [15], but the construction of such supervisors cannot be done in polynomial time. Also these results only hold for the case of a single language specification, and not for the case when the specification is given as a range.

2.5 Robust Supervisory Control

Up until now, one of our main assumptions was the determinism, in the different aspects of the particular control problem we are facing. Of these aspects, the plant model uncertainties are among the ones worth noting. Dealing with modeling uncertainties can be regarded as a *robust control problem.* With the family of possible models given, one might try to construct a controller to achieve the given specification, without trying to identify the real plant model. Another approach to deal with modeling uncertainties is *adaptive control,* which mainly is an attempt to resolve the uncertainty and to construct an appropriate controller afterward. Modeling uncertainties can be encountered in fault recovery procedures, when the controller is just attached to the system, and it is not known, for the controller at least, whether the system has undergone some failures or not. In these cases, normal and faulty operation of the system usually are modelled differently, and their design specifications are also generally different. For the purpose of this section, we review the work that has been done by Saboori and Hashtrudi Zad [50; 9], as our work in Chapter 3 is mainly an extension of that.

2.5.1 Robust Nonblocking Supervisory Control - Full Observation

The framework of this section has been first proposed by F. Lin [4]. It is assumed that the exact model of the system is unknown, but it is assumed that it belongs to a finite set of possibilities. We designate every possibility with its appropriate *superscript*, i.e. $\mathbf{G^i}$ ($i \in \mathcal{I} = \{1, \ldots, n\}$). Each of these possibilities has their own sets of alphabets, Σ^i, as well as their own design specification, E^i [5]. For now, we assume every event is observable, and further assume that the system models agree on controllability of any common event, i.e. if one plant, say model i, regards an event $\sigma \in \Sigma^i$ as (un)controllable, then any other model for which we have $\sigma \in \Sigma^j$ ($j \neq i$) also regards σ as (un)controllable. As before, a supervisor is a map

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

where

$$\Sigma = \bigcup_{i \in \mathcal{I}} \Sigma^i$$

In general $\Sigma^i \neq \Sigma$. A supervisor S^i for $\mathbf{G^i}$ is a map $S^i : \Sigma^i \rightarrow \Gamma_{\Sigma^i}$, which can be obtained from S by

$$S^i(s) = S(s) \cap \Sigma^i, \quad s \in \Sigma^{i*}$$

Definition 2.5.1. *[5] Robust Nonblocking Supervisor Control problem [RNSC]: Given a set of plant models $\mathbf{G^i}$, with $L(\mathbf{G^i}) \subseteq \Sigma^{i*}$, and a set of legal languages E^i, with $E^i \subseteq L_m(\mathbf{G^i})$ for all $i \in \mathcal{I} = \{1, \ldots, n\}$. Synthesize a supervisor $S : \Sigma \rightarrow \Gamma_\Sigma$, such that $\forall i \in \mathcal{I}$*

1. $L_m(S/\mathbf{G^i}) \subseteq E^i$,
2. $\overline{L_m(S/\mathbf{G^i})} = L(S/\mathbf{G^i})$.

As before, we are interested in the maximally permissive solutions to RNSC problem. Let

$$\mathbb{S} = \{S | S \text{ solves the RNSC problem.}\}$$

We call the supervisor S, a maximally permissive solution if and only if for all $S' \in \mathbb{S}$ and all $i \in \mathcal{I}$ we have

- $L_m(S'/\mathbf{G^i}) \subseteq L_m(S/\mathbf{G^i})$
- $L(S'/\mathbf{G^i}) \subseteq L_m(S/\mathbf{G^i})$

To characterize the solution of RNSC problem we need the following definition.

Definition 2.5.2. *[45] **Nonconflicting**: Two languages L and M are* nonconflicting *if and only if*

$$\overline{L \cap M} = \overline{L} \cap \overline{M}$$

Define $\mathbf{G}$ and E as follows

$$L(\mathbf{G}) = \bigcup_{i \in \mathcal{I}} L(\mathbf{G}^i), \quad L_m(\mathbf{G}) = \bigcup_{i \in \mathcal{I}} L_m(\mathbf{G}^i)$$

and,

$$E = \bigcap_{i \in \mathcal{I}} (E^i \cup (\Sigma^* - L_m(\mathbf{G}^i))) \cap L_m(\mathbf{G})$$

The solution to RNSC problem will be as follows.

Theorem 2.5.3. *[5] Let $K \subseteq E$ be a nonempty language, $\mathbf{G^i}$s and E^is as in RNSC problem. There exists a supervisor S for solving RNSC only if*

1. *K is controllable with respect to $\mathbf{G}$,*
2. *K and $L_m(\mathbf{G^i})$ are nonconflicting, and*

3. K is $L_m(\mathbf{G})$-closed.

Remark 2.5.4. *Conditions in Theorem 2.5.3 were given in [5] as necessary and sufficient, but later [9] showed that these conditions are only necessary and not sufficient in general.*

The following two lemmas are useful in proving our main result in Chapter 3.

Lemma 2.5.5. *[36] Suppose $\mathbf{G^1}$ and $\mathbf{G^2}$ are two DESs over some alphabet Σ. If we have*

$$L(\mathbf{G^1}) \subseteq L(\mathbf{G^2})$$

then for any supervisor $S : \Sigma^* \rightarrow \Gamma_\Sigma$, we will have

$$L(S/\mathbf{G^1}) = L(S/\mathbf{G^2}) \cap L(\mathbf{G^1}).$$

Lemma 2.5.6. *[5] Suppose $\mathbf{G^1}$ and $\mathbf{G^2}$ are two DESs over some alphabet Σ. If we have*

$$L(\mathbf{G^1}) \subseteq L(\mathbf{G^2}) \ \wedge \ L_m(\mathbf{G^1}) \subseteq L_m(\mathbf{G^2})$$

then for any supervisor $S : \Sigma^* \rightarrow \Gamma_\Sigma$, we will have

$$L_m(S/\mathbf{G^1}) = L_m(S/\mathbf{G^2}) \cap L_m(\mathbf{G^1}).$$

2.5.2 Robust Nonblocking Supervisor Control - Partial Observation

[9] extended the results of [5] to include the case of partial observation. It is assumed that exact plant model in unknown but it is among a set of possible models, $\mathbf{G^i}$ ($i \in \mathcal{I} = \{1, \ldots, n\}$). Associated with each model is a alphabet, Σ^i, and a design specification, E^i. It is assumed that different plant models agree on controllability

and observability of common events. So the sets of controllable and observable events are, respectively

$$\Sigma_o = \bigcup_{i\in\mathcal{I}} \Sigma_o^i \ \wedge \ \Sigma_c = \bigcup_{i\in\mathcal{I}} \Sigma_c^i$$

with $\Sigma_c^i, \Sigma_o^i \subseteq \Sigma^i$ as, controllable and observable event set of plant model i, respectively. Also let Σ represent the set of all possible event labels

$$\Sigma = \bigcup_{i\in\mathcal{I}} \Sigma^i$$

Definition 2.5.7. *[50; 9] Robust Nonblocking Supervisor Control under Partial Observation problem [RNSC-PO]: Given a set of plant models* $\mathbf{G^i}$*, with* $L(\mathbf{G^i}) \subseteq \Sigma^{i*}$*, and a set of legal languages* E^i*, with* $E^i \subseteq L_m(\mathbf{G^i})$ *for all* $i \in \mathcal{I} = \{1, \ldots, n\}$*. Synthesize a supervisor* $S : \Sigma \rightarrow \Gamma_\Sigma$*, such that* $\forall i \in \mathcal{I}$

1. $L_m(S/\mathbf{G^i}) \subseteq E^i$,
2. $\overline{L_m(S/\mathbf{G^i})} = L(S/\mathbf{G^i})$.

In order to characterize the solution to RNSC-PO problem [9] defined a new property called **G**-*nonblocking.*

Definition 2.5.8. *[9]* **G-*nonblocking*:** *Let* **G** *be a generator over* Σ*.* $K \subseteq \Sigma^*$ *is called* **G**-nonblocking *if and only if*

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

[9] showed that **G**-nonblocking is a stronger property than (K, $L_m(\mathbf{G})$) nonconflicting, and that the class of **G**-nonblocking sublanguages of a given language is nonempty and closed under arbitrary unions.

Theorem 2.5.9. *[9] Define* **G** *as*

$$L(\mathbf{G}) = \bigcup_{i\in\mathcal{I}} L(\mathbf{G}^i), \quad L_m(\mathbf{G}) = \bigcup_{i\in\mathcal{I}} L_m(\mathbf{G}^i)$$

and E as

$$E = \bigcap_{i \in \mathcal{I}} (E^i \cup (\Sigma^* - L_m(\mathbf{G}^i))) \cap L_m(\mathbf{G}).$$

There exists a supervisor S that solves RNSC-PO if and only if there exists a nonempty language $K \subseteq E$ such that

1. *K is controllable with respect to $\mathbf{G}$,*
2. *K is $\mathbf{G^i}$-nonblocking for all $i \in \mathcal{I}$,*
3. *K is $L_m(\mathbf{G})$-closed, and*
4. *K is $(L(\mathbf{G}), P)$-observable.*

Remark 2.5.10. *Conditions 1.-3. in Theorem 2.5.9 provide a set of necessary and sufficient conditions for Theorem 2.5.3.*

Chapter 3

Robust Supervisory Control Problem with respect to System Model

Like many other areas of study, there are more than one framework in which robust supervisory control has been studied. Robust supervisory control has been studied in three different frameworks, and we reviewed Lin's work and its extension by Saboori and Hashtrudi Zad [50; 9] in Chapter 2. While we did not discuss the remaining two robustness paradigm, it is safe to say that Lin's work "seems most natural in the context of DES" [40]. In this chapter our aim is to extend the results of [9] to include the case where a set of decentralized supervisors are required to solve the given control problem.

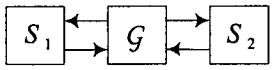

Figure 3.1: System schematic of the RDSC problem, $\mathcal{G} \in \{\mathbf{G_1}, \ldots, \mathbf{G_n}\}$

3.1 RDSC Problem Formulation

Assume a set of plant models, $\mathbf{G^i}$ ($i \in \mathcal{I} = \{1, \ldots, n\}$), over their respective alphabets, Σ^i. We are given a set of specifications, one for each possible plant model, E^i, a set of alphabets representing each decentralized supervisor local alphabet, Σ_j ($j \in \mathcal{I} = \{1, \ldots, m\}$), and are required to synthesize a set of decentralized supervisors such that for each plant model the resulting 'plant under supervision' is nonblocking and stays inside its corresponding specification. Formally,

Definition 3.1.1. *Robust Decentralized Supervisor Control problem [RDSC]: Given a set of possible plant models* $\{\mathbf{G}^1, \ldots, \mathbf{G}^n\}$ *respectively over the sets of alphabets* Σ^i, *with* $i \in \mathcal{I} = \{1, \ldots, n\}$, *and a set of languages* $\{E^1, \ldots, E^n\}$ *describing the legal behavior of each plant's possible models* ($\forall i \in \mathcal{I}.\ E^i \subseteq L_m(\mathbf{G^i})$), *synthesize two feasible supervisors,*

$$S_1 : {\Sigma_1}^* \rightarrow \Gamma_{\Sigma_1}, \text{ and}$$

$$S_2 : {\Sigma_2}^* \rightarrow \Gamma_{\Sigma_2}$$

such that $\forall i \in \mathcal{I}$,

1. $L_m(S_1 \wedge S_2/\mathbf{G^i}) \subseteq E^i$,
2. $\overline{L_m(S_1 \wedge S_2/\mathbf{G^i})} = L(S_1 \wedge S_2/\mathbf{G^i})$.

The solution to the RDSC problem is similar to that of RNSC-PO presented by Theorem 2.5.9 [9]. The only difference between the two is in the number of agents (i.e. supervisors) which have to control the given plant. Note that while we

have restricted the problem to the case where only two decentralized supervisors are required, this restriction is only for convenience and the generalization of the results to any number of supervisors is straightforward. Figure 3.1 shows the schematic of the RDSC problem. So, let Σ be the set of all event labels,

$$\Sigma = \bigcup_{i \in \mathscr{I}} \Sigma^i$$

Thus,

$$\Sigma_1, \Sigma_2 \subseteq \Sigma$$

Define $\mathbf{G}$ and E as follows,

$$L(\mathbf{G}) = \bigcup_{i \in \mathcal{I}} L(\mathbf{G^i}), \qquad L_m(\mathbf{G}) = \bigcup_{i \in \mathcal{I}} L_m(\mathbf{G^i})$$

and,

$$E = \bigcap_{i \in \mathcal{I}} (E^i \cup (\Sigma^* - L_m(\mathbf{G^i}))) \cap L_m(\mathbf{G})$$

We need the following lemmas.

Lemma 3.1.2. *Assume S_1 and S_2 are two supervisors solving RDSC problem. Let $K^i = L_m(S_1 \wedge S_2/\mathbf{G^i})$ and $K = L_m(S_1 \wedge S_2/\mathbf{G})$, then*

$$K = \bigcup_{i \in \mathcal{I}} K^i.$$

Proof.

$$
\begin{aligned}
K &= L_m(S_1 \wedge S_2/\mathbf{G}) \\
&= L_m(S_1 \wedge S_2/\mathbf{G}) \cap L_m(\mathbf{G}) \qquad (K \subseteq L_m(\mathbf{G})) \\
&= L_m(S_1 \wedge S_2/\mathbf{G}) \cap (\bigcup_{i\in\mathcal{I}} L_m(\mathbf{G^i})) \\
&= \bigcup_{i\in\mathcal{I}} (L_m(S_1 \wedge S_2/\mathbf{G}) \cap L_m(\mathbf{G^i})) \\
&= \bigcup_{i\in\mathcal{I}} L_m(S_1 \wedge S_2/\mathbf{G^i}) \qquad \text{(by Lemma 2.5.6)} \\
&= \bigcup_{i\in\mathcal{I}} K^i.
\end{aligned}
$$

Lemma 3.1.3. *Assume S_1 and S_2 are two supervisors solving RDSC problem. Then*

$$\overline{K} = \overline{L_m(S_1 \wedge S_2/\mathbf{G})} = L(S_1 \wedge S_2/\mathbf{G}).$$

Proof. Let $K^i = L_m(S_1 \wedge S_2/\mathbf{G^i})$.

$$
\begin{aligned}
\overline{K} &= \bigcup_{i\in\mathcal{I}} \overline{K^i} \qquad \text{(by Lemma 3.1.2)} \\
&= \bigcup_{i\in\mathcal{I}} \overline{L_m(S_1 \wedge S_2/\mathbf{G^i})} \\
&= \bigcup_{i\in\mathcal{I}} L(S_1 \wedge S_2/\mathbf{G^i}) \qquad (S_1 \wedge S_2 \text{ solves RDSC}) \\
&= \bigcup_{i\in\mathcal{I}} (L(S_1 \wedge S_2/\mathbf{G}) \cap L(\mathbf{G^i})) \qquad \text{(by Lemma 2.5.5)} \\
&= L(S_1 \wedge S_2/\mathbf{G}) \cap (\bigcup_{i\in\mathcal{I}} L(\mathbf{G^i})) \\
&= L(S_1 \wedge S_2/\mathbf{G}) \cap L(\mathbf{G}) \qquad \text{(by definition)} \\
&= L(S_1 \wedge S_2/\mathbf{G}).
\end{aligned}
$$

Lemma 3.1.4. *Assume S_1 and S_2 are two supervisors solving RDSC problem. Let $K^i = L_m(S_1 \wedge S_2/\mathbf{G^i})$, then for $i, j \in \mathcal{I}$*

$$\overline{K^j} \cap L(\mathbf{G^i}) \subseteq \overline{K^i} \cap L(\mathbf{G^i}).$$

Proof.

$$\begin{aligned}
\overline{K^j} \cap L(\mathbf{G^i}) &= L(S_1 \wedge S_2/\mathbf{G^j}) \cap L(\mathbf{G^i}) \qquad (S_1 \wedge S_2 \text{ solves RDSC}) \\
&= L(S_1 \wedge S_2/\mathbf{G}) \cap L(\mathbf{G^j}) \cap L(\mathbf{G^i}) \qquad \text{(by Lemma 2.5.5)} \\
&= L(S_1 \wedge S_2/\mathbf{G^i}) \cap L(\mathbf{G^j})) \\
&\subseteq L(S_1 \wedge S_2/\mathbf{G^i}) \\
&= \overline{K^i} \qquad (S_1 \wedge S_2 \text{ solve RDSC}) \\
&= \overline{K^i} \cap L(\mathbf{G^i}) \qquad (K^i \subseteq L(\mathbf{G^i}))
\end{aligned}$$

Remark 3.1.5. *Lemma 3.1.4 specially implies,*

1. $\overline{K^j} \cap L_m(\mathbf{G^i}) \subseteq \overline{K^i} \cap L_m(\mathbf{G^i})$, *and*
2. $\overline{K^j}\Sigma_{uc}^* \cap L(\mathbf{G^i}) \subseteq \overline{K^i}\Sigma_{uc}^* \cap L(\mathbf{G^i})$. *(from Lemma 3.1.4, and* $(A \cap B)\Sigma_{uc}^* = A\Sigma_{uc}^* \cap B\Sigma_{uc}^*$ *if* A *and* B *are closed.)*

3.2 Solution to RDSC Problem

Theorem 3.2.1. *There exist two supervisors* S_1 *and* S_2 *solving RDSC if and only if there exists a nonempty sublanguage* $K \subseteq E$ *such that*

1. K *is controllable with respect to* $\mathbf{G}$,
2. K *is* $(L_m(\mathbf{G}), P_1, P_2)$*-coobservable,*
3. K *is* $\mathbf{G^i}$*-nonblocking for all* $i \in \mathcal{I}$, *and*
4. K *is* $L_m(\mathbf{G})$*-closed.*

Remark 3.2.2. *Before we proceed to the proof of Theorem 3.2.1, we like to comment on the computation of* K. *It turns out that finding a sublanguage of* E *that satisfies*

the condition 1.-4. is not easy. In Chapter 2, in the solution to Theorem 2.4.15 we discussed that coobservability is not closed under arbitrary unions and thus a supremal sublanguage of a given language can not be found. The only remedy here is to check all of the conditions for a subset of E, starting from E itself, and upon failing remove one or more transitions, thus obtaining another (smaller) subset of E. This procedure converges in general to the empty language, but does not necessarily provides the maximally permissive solution.

Proof.(if) By Theorem 2.4.15, with K being (1) Controllable, (2) ($L_m(\mathbf{G})$, P_1, P_2)-coobservable, and (3) $L_m(\mathbf{G})$-closed, we can find two supervisors, S_1 and S_2, such that $L_m(S_1 \wedge S_2/\mathbf{G}) = K$ and $L(S_1 \wedge S_2/\mathbf{G}) = \overline{K}$. We have to show $S_1 \wedge S_2$ also solves *RDSC* problem.

$$
\begin{aligned}
L_m(S_1 \wedge S_2/\mathbf{G^i}) &= L_m(S_1 \wedge S_2/\mathbf{G}) \cap L_m(\mathbf{G^i}) \qquad \text{(by Lemma 2.5.6)} \\
&= K \cap L_m(\mathbf{G^i}) \\
&\subseteq E \cap L_m(\mathbf{G^i}) \qquad (K \subseteq E) \\
&= \bigcap_{i \in \mathcal{I}} (E^i \cup (\Sigma^* - L_m(\mathbf{G^i}))) \cap L_m(\mathbf{G^i}) \\
&= \bigcap_{i \in \mathcal{I}} (E^i \cap L_m(\mathbf{G^i})) \\
&\subseteq E^i.
\end{aligned}
$$

Also for nonblocking property,

$$\begin{aligned}
\overline{L_m(S_1 \wedge S_2/\mathbf{G^i})} &= \overline{L_m(S_1 \wedge S_2/\mathbf{G}) \cap L_m(\mathbf{G^i})} \qquad \text{(by Lemma 2.5.6)} \\
&= \overline{K \cap L_m(\mathbf{G^i})} \\
&= \overline{K} \cap L(\mathbf{G^i}) \qquad (K \text{ being } \mathbf{G^i}\text{-nonblocking}) \\
&= L(S_1 \wedge S_2/\mathbf{G}) \cap L(\mathbf{G^i}) \\
&= L(S_1 \wedge S_2/\mathbf{G^i}). \qquad \text{(by Lemma 2.5.5)}
\end{aligned}$$

(Only-if) Assume two supervisors S_1 and S_2 exist such that

$$\begin{aligned}
L_m(S_1 \wedge S_2/\mathbf{G}^i) &= K^i \subseteq E^i, \text{ and} \\
\overline{L_m(S_1 \wedge S_2/\mathbf{G^i})} &= L(S_1 \wedge S_2/\mathbf{G^i})
\end{aligned}$$

Define the language $K = \bigcup_{i \in \mathcal{I}} K^i$, with $K^i = L_m(S_1 \wedge S_2/\mathbf{G}^i)$. We show that $K \subseteq E$ and also has the quadruple properties.

$$\begin{aligned}
K^i \subseteq E^i &\Rightarrow L_m(S_1 \wedge S_2/\mathbf{G^i}) \subseteq E^i \\
&\Rightarrow L_m(S_1 \wedge S_2/\mathbf{G}) \cap L_m(\mathbf{G^i}) \subseteq E^i \qquad \text{(by Lemma 2.5.6)} \\
&\Rightarrow L_m(S_1 \wedge S_2/\mathbf{G}) \subseteq E^i \cup (\Sigma^* - L_m(\mathbf{G^i})) \\
&\Rightarrow L_m(S_1 \wedge S_2/\mathbf{G}) \subseteq \bigcap_{i \in \mathcal{I}} \Big(E^i \cup (\Sigma^* - L_m(\mathbf{G^i})) \Big) \\
&\Rightarrow L_m(S_1 \wedge S_2/\mathbf{G}) \subseteq \bigcap_{i \in \mathcal{I}} \Big(E^i \cup (\Sigma^* - L_m(\mathbf{G^i})) \Big) \cap L_m(\mathbf{G}) \\
&\qquad (L_m(S_1 \wedge S_2/\mathbf{G}) \subseteq L_m(\mathbf{G})) \\
&\Rightarrow L_m(S_1 \wedge S_2/\mathbf{G}) \subseteq E.
\end{aligned}$$

By Theorem 2.4.15 and the fact that $S_1 \wedge S_2$ solves RDSC problem, we can deduce that K^i $(i \in \mathcal{I})$ is (1) Controllable wrt $\mathbf{G^i}$, (2) $(L_m(\mathbf{G^i}), P_1, P_2)$-coobservable, and (3) $L_m(\mathbf{G^i})$-closed. Now, we use these properties to prove the quadruple properties of K.

Controllability.

$$\begin{aligned}
\overline{K}\Sigma_{uc} \cap L(\mathbf{G}) &= \Big(\overline{\bigcup_{i\in\mathcal{I}} K^i}\Sigma_{uc}\Big) \cap \Big(\bigcup_{j\in\mathcal{I}} L(\mathbf{G^j})\Big) \qquad \text{(by Lemma 3.1.2)} \\
&= \Big(\bigcup_{i\in\mathcal{I}} \overline{K^i}\Sigma_{uc}\Big) \cap \Big(\bigcup_{j\in\mathcal{I}} L(\mathbf{G^j})\Big) \\
&= \bigcup_{i\in\mathcal{I}}\bigcup_{j\in\mathcal{I}} (\overline{K^i}\Sigma_{uc} \cap L(\mathbf{G^j})) \\
&= \bigcup_{i\in\mathcal{I}} (\overline{K^i}\Sigma_{uc} \cap L(\mathbf{G^i})) \qquad \text{(by Remark 3.1.5)} \\
&\subseteq \bigcup_{i\in\mathcal{I}} \overline{K^i} \qquad (K^i\text{s are controllable w.r.t. } \mathbf{G^i}) \\
&= \overline{K}. \qquad \text{(by Lemma 3.1.2)}
\end{aligned}$$

Coobservability. We need to show that the three conjuncts of coobservability cannot fail. Note that for $i \in \{1,2\}$ $P_i : \Sigma^* \to \Sigma_{i,o}^*$ is well-defined. So, $\forall s, s', s'' \in \Sigma^*$ such that $P_1(s) = P_1(s') \wedge P_2(s) = P_2(s'')$ we have:

Case 1, conjunct 1 fails: $\exists \sigma \in \Sigma_{1,c} \cap \Sigma_{2,c}.\ s'\sigma, s''\sigma \in \overline{K} \wedge s \in \overline{K} \wedge s\sigma \in L(\mathbf{G}) \wedge s\sigma \notin \overline{K}$

So there exists $i, j, l \in \mathcal{I}$ such that, $s\sigma \in L(\mathbf{G^i}) \wedge s'\sigma \in \overline{K^j} \wedge s''\sigma \in \overline{K^l}$ and we have,

$$\left.\begin{aligned} s \in \overline{K} &\Rightarrow s \in L(\mathbf{G^i}) \\ s \in \overline{K} \Rightarrow s \in L(S_1 \wedge S_2/\mathbf{G}) &\Rightarrow s \in L(S_1/\mathbf{G}) \wedge s \in L(S_2/\mathbf{G}) \end{aligned}\right\}$$

From the above two equations we can deduce

$$s \in L(S_1/\mathbf{G^i}) \wedge s \in L(S_2/\mathbf{G^i}) \tag{3.1}$$

Also,

$$\left.\begin{aligned} P_1(s) = P_1(s') &\Rightarrow S_1(s) = S_1(s') \\ s'\sigma \in \overline{K^j} \Rightarrow s'\sigma \in L(S_1/\mathbf{G^j}) &\Rightarrow \sigma \in S_1(s') \end{aligned}\right\}$$

Thus

$$\sigma \in S_1(s) \tag{3.2}$$

and,

$$\left.\begin{aligned} P_2(s) = P_2(s'') \Rightarrow S_2(s) = S_2(s'') \\ s''\sigma \in \overline{K^l} \Rightarrow s''\sigma \in L(S_2/\mathbf{G}^l) \Rightarrow \sigma \in S_2(s'') \end{aligned}\right\}$$

Therefore

$$\sigma \in S_2(s) \tag{3.3}$$

From (3.1), (3.2), and (3.3) we have,

$$s\sigma \in L(S_1 \wedge S_2/\mathbf{G}^{\mathbf{i}}) \subseteq L(S_1 \wedge S_2/\mathbf{G}) = \overline{K}$$

A contradiction, which shows that conjunct 1 in coobservability cannot fail.

Case 2, conjunct 2 fails: $\exists \sigma \in \Sigma_{1,c} \backslash \Sigma_{2,c}.\ s'\sigma \in \overline{K} \wedge s \in \overline{K} \wedge s\sigma \in L(\mathbf{G}) \wedge s\sigma \notin \overline{K}$

We have

$$\sigma \notin \Sigma_{2,c} \Rightarrow \sigma \in S_2(s)$$

which together with (3.2) yields,

$$s\sigma \in L(S_1 \wedge S_2/\mathbf{G}^{\mathbf{i}}) \subseteq \overline{K}$$

A contradiction. Therefore conjunct 2 can't fail either.

Case 3, conjunct 3 fails: Similar to conjunct 2.

So none of the three conjuncts of coobservability could fail and therefore K is coobservable.

$L_m(\mathbf{G})$-closure.

$$\begin{aligned}
\overline{K} \cap L_m(\mathbf{G}) &= \Big(\bigcup_{i\in\mathcal{I}} \overline{K^i}\Big) \cap \Big(\bigcup_{j\in\mathcal{I}} L_m(\mathbf{G^j})\Big) \qquad \text{(by Lemma 3.1.2)} \\
&= \bigcup_{i\in\mathcal{I}}\bigcup_{j\in\mathcal{I}} (\overline{K^i} \cap L_m(\mathbf{G^j})) \\
&= \bigcup_{i\in\mathcal{I}} (\overline{K^i} \cap L_m(\mathbf{G^i})) \qquad \text{(by Remark 3.1.5)} \\
&= \bigcup_{i\in\mathcal{I}} K^i \qquad (K^i\text{s are } L_m(\mathbf{G^i})\text{-closed}) \\
&= K
\end{aligned}$$

$\mathbf{G^i}$-nonblocking.

$$\begin{aligned}
\overline{K \cap L_m(\mathbf{G^i})} &= \overline{L_m(S_1 \wedge S_2/\mathbf{G^i})} \qquad \text{(by Lemma 2.5.6 and } K = L_m(S_1 \wedge S_2/\mathbf{G})) \\
&= L(S_1 \wedge S_2/\mathbf{G^i}) \qquad (S_1 \wedge S_2 \text{ is nonblocking}) \\
&= L(S_1 \wedge S_2/\mathbf{G}) \cap L(\mathbf{G^i}) \qquad \text{(by Lemma 2.5.5)} \\
&= \overline{K} \cap L(\mathbf{G^i}) \qquad \text{(by Lemma 3.1.3)}
\end{aligned}$$

This concludes our proof of Theorem 3.2.1. ■

3.3 Two Examples

We examine each of the quadruple properties of the solution to RDSC problem by the following example, to show how each one is of importance and what happens when each is violated.

Example 1. In the following example $\Sigma = \{a_1, b_2, c, d, f\}$, $\Sigma_{1,c} = \Sigma_{1,o} = \{a_1, c, d\}$, and $\Sigma_{2,c} = \Sigma_{2,o} = \{b_2, c, d\}$ which leaves 'f' as an unobservable uncontrollable event for both supervisors. Figure 3.2 and Figure 3.3 show a possible model, $\mathbf{G^1}$, for a plant and its specification. This plant model does not require any kind of control

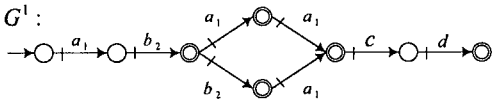

Figure 3.2: Plant model 1

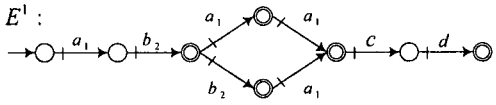

Figure 3.3: Plant specification 1

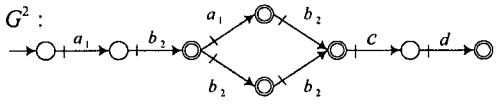

Figure 3.4: Plant model 2

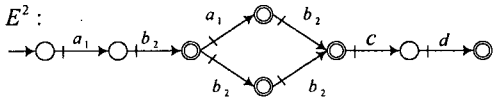

Figure 3.5: Plant specification 2

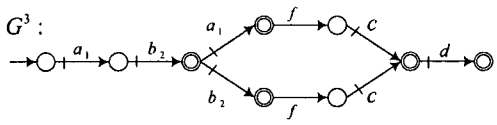

Figure 3.6: Plant model 3

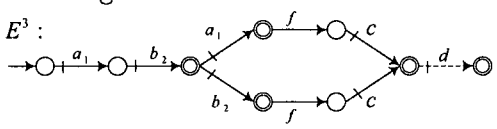

Figure 3.7: Plant specification 3

as the specification is equal to the plant model. Figure 3.4 and Figure 3.5 show another possible model for the plant and its associated specification. Again this plant model does not require any control action. Figure 3.6 and Figure 3.7 show the last possible model for the plant and its associated specification. This plant model requires disablement of event 'd' after occurrence of 'c'. Note that event 'd' in Figure 3.7 is not part of the specification. It is shown as dashed event to indicate that it has to be removed from plant behavior. Also note that the states after the occurrence of 'f' along with the initial state and the state after string $s = a_1$ are not marker states. Now, the problem is to find two supervisors which could control the plant according to the specification that is provided for each of the possible plant models. As the plant model is uncertain our only option is to design two robust supervisors such that their cooperative action would not allow any illegal event to be executed regardless of which plant model is actually running at the time. For this, we construct the closed and marked behaviors of plant $\mathbf{G}$, Figure 3.8, and its associated specification, Figure 4.6, according to,

$$L(\mathbf{G}) = \bigcup_{i \in \mathcal{I}} L(\mathbf{G^i}), \qquad L_m(\mathbf{G}) = \bigcup_{i \in \mathcal{I}} L_m(\mathbf{G^i})$$

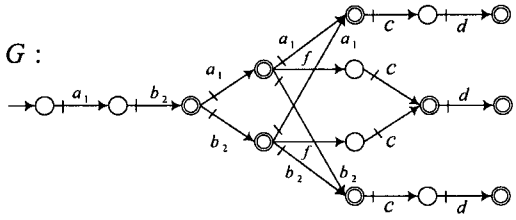

Figure 3.8: Plant model

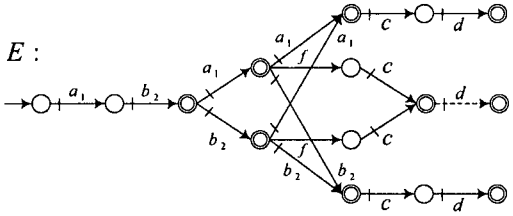

Figure 3.9: Specification

and,

$$E = \bigcap_{i \in \mathcal{I}} (E^i \cup (\Sigma^* - L_m(\mathbf{G^i}))) \cap L_m(\mathbf{G})$$

Now, we only have to find K a sublanguage of E which has the quadruple properties in Theorem 3.2.1, and construct two supervisors S_1 and S_2 such that the plant under supervision, $(S_1 \wedge S_2)/\mathbf{G}$, recognizes K. Naturally, our first guess would be E itself, so we take $K_1 = E$ as our starting point and check if it satisfies all of the four properties: K_1 is not coobservable, as taking $s = a_1 b_2 a_1 f c$, $s' = a_1 b_2 a_1 b_2 c$, $s'' = a_1 b_2 a_1 a_1 c$, and $\sigma = d$ gives,

$$s'\sigma, s''\sigma \in \overline{K_1} \wedge s \in \overline{K_1} \wedge s\sigma \in L(\mathbf{G}) \wedge s\sigma \notin \overline{K_1}$$

So we disable 'd' in plant models $\mathbf{G^1}$ and $\mathbf{G^2}$ to avoid the confusion for supervisors, which in turn gives us K_2, Figure 3.10. Turns out that K_2 is not $\mathbf{G^i}$-nonblocking. Comparing with $\mathbf{G^1}$ it can be easily seen that $s = a_1 b_2 a_1 a_1 c \in (\overline{K_2} \cap L(\mathbf{G^1}))$ but $s \notin \overline{K_2 \cap L_m(\mathbf{G^1})}$, and as such,

$$\overline{K_2} \cap L(\mathbf{G^1}) \neq \overline{K_2 \cap L_m(\mathbf{G^1})}$$

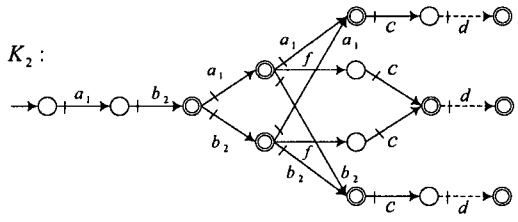

Figure 3.10: $K_2 \subseteq E$

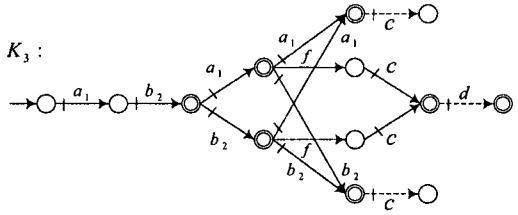

Figure 3.11: $K_3 \subseteq E$

To remove this blocking we have to disable event 'c' in plant models $\mathbf{G^1}$ and $\mathbf{G^2}$ which leads to the non-marker states, and thus we obtain K_3, Figure 3.11. K_3 is not coobservable, as taking $s = a_1b_2a_1b_2$, $s' = a_1b_2a_1f$, $s'' = a_1b_2b_2f$, and $\sigma = c$ gives

$$s'\sigma, s''\sigma \in \overline{K_3} \wedge s \in \overline{K_3} \wedge s\sigma \in L(\mathbf{G}) \wedge s\sigma \notin \overline{K_3}$$

So we need to disable 'c' in plant model $\mathbf{G^3}$ to ensure consistent decision making of our robust supervisors, namely to disable 'c' anywhere, which in turn gives us K_4, Figure 3.12. K_4 is not $\mathbf{G^i}$-nonblocking as taking $s = a_1b_2a_1f$ gives

$$s \in \overline{K_4} \cap L(\mathbf{G^3}) \qquad \wedge \qquad s \notin \overline{K_4 \cap L_m(\mathbf{G^3})}$$

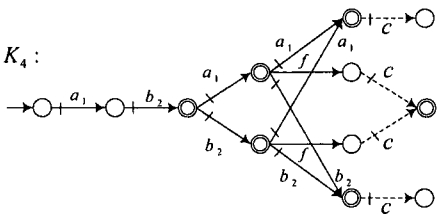

Figure 3.12: $K_4 \subseteq E$

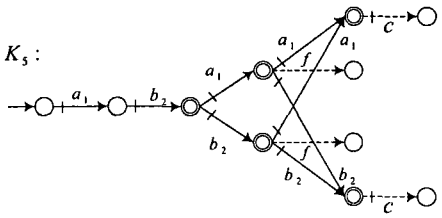

Figure 3.13: $K_5 \subseteq E$

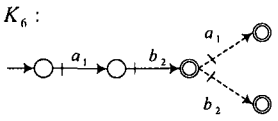

Figure 3.14: $K_6 \subseteq E$

In other words,

$$\overline{K_4} \cap L(\mathbf{G^3}) \neq \overline{K_4 \cap L_m(\mathbf{G^3})}$$

As before we have to disable some events to remove the blocking states, this time event 'f'. This choice gives us yet another smaller sublanguage $K_5 \subseteq E$, Figure 3.13. K_5 violates another property in the quadruple set of properties; controllability. 'f' is not controllable by any of the two supervisors and thus it can not be disabled, but it is possible to disable any event leading to where 'f' can be executed if those events are controllable. It turns out that those events are indeed controllable, namely a_1 and b_2, but their disablement also removes most of the plants legal behavior. As there is no other way to deal with uncontrollability of K_5, other than adding some actuators to make 'f' controllable perhaps, we obtain K_6, Figure 3.14. K_6 is (1) Controllable, (2) ($L_m(\mathbf{G})$, P_1, P_2)-coobservable, (3) $L_m(\mathbf{G})$-closed, and (4) $\mathbf{G^i}$-nonblocking for all $i \in \mathcal{I}$. Therefore local supervisor S_1 (S_2) should be designed to enable 'a_1' ('b_2') at first, and disables it upon its first execution. While the maximally permissive solution to this problem is found, it is however very limited in behavior as our decentralized supervisors are only able to exercise control by what they can observe themselves, not

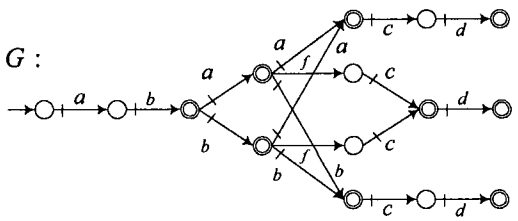

Figure 3.15: Plant model 1

having access to the information of the other supervisor. It is interesting to see what would be the result of this example if both local supervisors had information about the other supervisor as well, to see the importance of communication between local supervisors and to compare the behavior of resulting 'plant under supervision' with the case where there is no communication. Incorporating communication between supervisors changes the terminology that we have used thus far in this chapter, namely 'decentralized supervisor', to 'distributed supervisor' and results in unwanted confusion, so instead of implementing communication we simply change local events to global ones, implying the communication between supervisors without going through the troubles of implementing one.

Example 2. In the following example $\Sigma = \{a, b, c, d, f\}$, $\Sigma_{1,c} = \{a, c, d\}$, $\Sigma_{2,c} = \{b, c, d\}$, and $\Sigma_{1,o} = \Sigma_{2,o} = \{a, b, c, d\}$. '$f$' is an unobservable uncontrollable event for both supervisors. Possible plant models and their respective specification are similar to Example 1, so we have plant model $\mathbf{G}$, Figure 3.15, and specification E, Figure 3.16. To find K a sublanguage of E with the properties in Theorem 3.2.1 we start by $K_1 = E$, Figure 3.17. It appears that K_1 is (1) Controllable, (2) ($L_m(\mathbf{G})$, P_1, P_2)-coobservable, (3) $L_m(\mathbf{G})$-closed, and (4) $\mathbf{G^i}$-nonblocking for all $i \in \mathcal{I}$. Structure of S_1 and S_2 are given in Figure 3.18 and Figure 3.19, respectively. Comparing K_1 of Figure 3.17 to K_6 of Figure 3.14, we could easily see the impact of combining the information of local supervisors with each other. However when we changed local

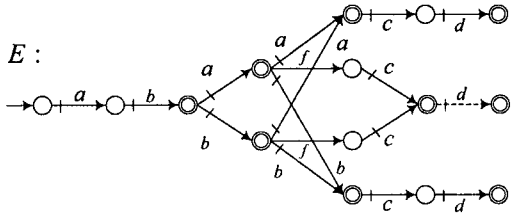

Figure 3.16: Plant specification 1

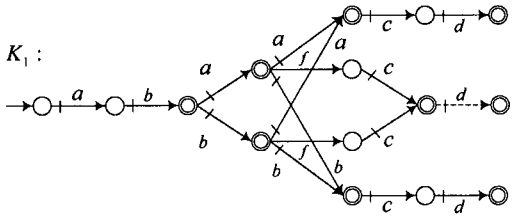

Figure 3.17: $K_1 \subseteq E$

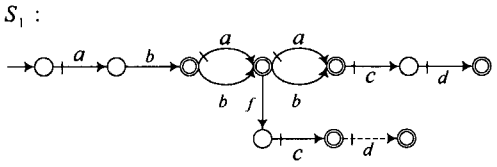

Figure 3.18: Supervisor S_1

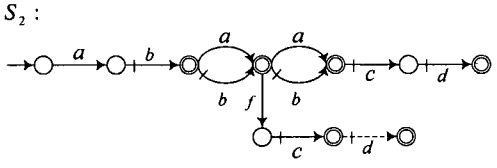

Figure 3.19: Supervisor S_2

events to global to examine the effect of communication between supervisors, we implicitly assumed that any such communication is instantaneous and without delay. While a lot of work has been done through this assumption by many researchers, little work has been done on the subject of delay in communication channels. This is the subject of the next chapter.

3.4 Conclusion

In this chapter we introduced the decentralized version of the robust supervisory control problem (RDSC problem) previously formulated by Lin. We provided the necessary and sufficient conditions for the existence of the solution to RDSC problem, and formally proved it. We examined the conditions of the solution of RDSC problem with two examples and also showed the limitations of control without communication by comparing the results of those two examples.

Chapter 4

Robustness With Respect To Communication Delay

Plant model uncertainties, the subject of previous chapter, is not the only source of non-determinism, but the most discussed one. Working with communicating agents in a distributed system leads to the introduction of another source of non-determinism: delay. In this chapter we try to examine the requirements for the construction of a set of robust decentralized supervisors which are communicating with each other a non-ideal channel in which the delay in not zero.

In the previous chapter we only considered *noncommunicating* decentralized supervisors and we saw that the results of such restraint is the limited behavior of the plant under supervision. While working on implementing communication in our work, we noticed a huge difference between the amount of work that has been done on the issue of communication between supervisors with the assumption of ideal channels, and what has been done with the assumption of delay in communication channels. This chapter can be viewed as an independent topic on robust decentralized supervisory control of discrete-event systems or serves as an extension of the previous chapter's

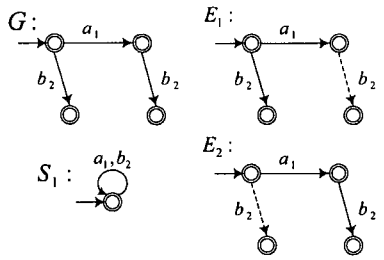

Figure 4.1: Example 4.1.1

work when one desires to implement communication between supervisors in a practical setting, where delay is not-negligible. At the end of this chapter we also present an example to further examine the results of this chapter.

4.1 Problem Formulation

It is common in supervisory control of distributed discrete-event systems to assume that communication between supervisors is instantaneous, thus allowing for more sophisticated problems to be addressed. Interestingly, some control problems have the property that supervisors designed under the assumption of no-delay still work with the introduction of delay in communication between supervisors, while some other control problems simply can no longer be solved by the same supervisors with the introduction of delay. Our aim is to characterize the former group, which turns out to be a nontrivial subclass of control problems. Our motivation for this work could be better understood with an example.

Example 4.1.1. *Let* $\Sigma = \{a_1, b_2\}$, $\Sigma_1 = \Sigma_{c,1} = \Sigma_{o,1} = \{a_1\}$, *and* $\Sigma_2 = \Sigma_{c,2} = \Sigma_{o,2} = \{b_2\}$. *Plant model is shown in Figure 4.1 along with two different specification* E_1 *and* E_2. *Examining* E_1 *and* E_2 *shows that they are not coobservable, and therefore it is not possible to achieve any of them without communication between the two supervisors;*

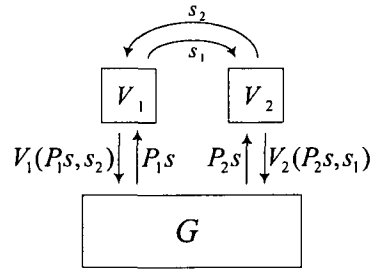

Figure 4.2: System schematic

then it is easy to show that a simple communication policy, for example communicating 'a_1' to supervisor 2, is enough to achieve both specifications when communication is instantaneous. Thus, in our supervisor design, supervisor 1 does not do anything, while supervisor 2 can be implemented in each case by the automaton of E_i, $i = 1, 2$. With the introduction of delay, however, we see a difference: we can no longer achieve E_1, while E_2 can be achieved with the same supervisors. In the presence of delay, in case of E_1, supervisor 2 enables 'b_2' until it receives 'a_1'. If 'b_2' actually happens when 'a_1' has happened but not yet received by supervisor 2, E_1 is clearly violated. In case of E_2, supervisor 2 disables 'b_2' until it receives 'a_1', in which case 'b_2' is enabled. Because of the nature of E_2 that allows supervisor 2 to "enable more" as new communication arrives, E_2 can be implemented in the presence of delay.

A class of *unbounded-communication-delay-robust* languages (UCDR languages for short), which will be formally defined later, could be regarded as the only class of specifications that can be achieved under the assumption of unbounded delay in the communication network. Figure 4.2 shows the schematic of the system, including the plant, a set of local supervisors, and the communication between them that will be discussed in details later.

Our main assumptions for this work are as follows:

- We assume that the plant consists of a number of distributed components,

each defined over a local alphabet Σ_i, $i \in \mathcal{I} = \{1, 2, \ldots, n\}$. We assume that a supervisor for component i has control over some events in Σ_i, that local alphabets are disjoint, and that all events in Σ_i are observable to supervisor $i \in \mathcal{I}$; thus $\Sigma_{c,i} \subseteq \Sigma_i$, $\Sigma_{o,i} = \Sigma_i$, $\Sigma_{uc,i} = \Sigma \setminus \Sigma_{c,i}$ and $\Sigma_{uo,i} = \Sigma \setminus \Sigma_i$.

- (Unbounded delay assumption) It is assumed that in our setting all events are communicated, and all communications are *eventually* received. There is a First-input-first-output (FIFO) communication channel between every pair of supervisors; it is such that the network preserves communication order while, it might take an arbitrarily long time before a message is delivered.

We shall also call UCDR supervisors those supervisors that implement a UCDR language as the specification. UCDR supervisors prevent illegal strings without requiring any particular communication to be delivered to them. In other words a local supervisor disables events that might generate illegal strings, with reliance solely on its own observations. Furthermore, local supervisors will enable some events after enough communication has been delivered to them to disambiguate look-alike cases. While it might take arbitrary long before any particular communication is delivered, the local supervisors will not be blocking as all those communications will be eventually delivered. These points are summarized in the following remark.

Remark 4.1.2. *In a set of UCDR supervisors, each supervisor decides when to disable its locally controllable events based on its own observations. Enabling a locally controllable event, on the other hand, might require some communication to be delivered.*

Let $\mathbf{G}$ be a plant over alphabet Σ, where Σ is partitioned into n disjoint subal-

phabets $\Sigma_1, \Sigma_2, \ldots, \Sigma_n$ such that

$$\Sigma = \bigcup_{i \in \mathcal{I}} \Sigma_i.$$

Also let $\Sigma_{c,i} \subseteq \Sigma_i$ and define the set of local control patterns for supervisor i as

$$\Gamma_i := \{\gamma \in \mathcal{P}(\Sigma) | \gamma \supseteq \Sigma \backslash \Sigma_{c,i}\}.$$

Also for convenience define

$$\Gamma := \{\gamma \in \mathcal{P}(\Sigma) | \gamma \supseteq \Sigma \backslash \bigcup_{i \in \mathcal{I}} \Sigma_{c,i}\}.$$

Let $s \in L(\mathbf{G})$. For $i \in \mathcal{I}$, supervisor i can base its control decisions on its direct observation of the plant (i.e. $P_i s$), and on the (possibly) incomplete information $s_j \leq P_j s$ that it has received thus far from all other supervisors j, $j \in \mathcal{I} \setminus \{i\}$. Then a natural way to define supervisory control for supervisor i is through a partial map

$$V_i : \prod_{i \in \mathcal{I}} \Sigma_i^* \to \Gamma_i$$

such that $V_i(s_i, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}})$ is defined if there exists $s \in L(\mathbf{G})$ such that:

1. $s_i = P_i s$, and

2. $s_j \leq P_j s$ for $j \in \mathcal{I} \setminus \{i\}$.

Suppose at $s \in L(\mathbf{G})$, supervisor i has received $s_j \leq P_j s$ through communication from other supervisors, $j \in \mathcal{I} \setminus \{i\}$. Then supervisor i enables the events in $V_i(P_i s, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}})$. At some unspecified time in future, supervisor i receives additional information through communication, leading it to form (still possibly incomplete) projections $s'_j \leq P_j s$, $j \in \mathcal{I} \setminus \{i\}$. Note that by FIFO assumption for the channels $s_j \leq s'_j$ $(\forall j \neq i)$, which is an indication that the information now is more complete, and is closer to the reality of what have actually happened in the plant

(i.e. $P_j s$). As a result of the additional information received, supervisor i updates its control decision to $V_i(P_i s, \{s'_j\}_{j\in\mathcal{I}\setminus\{i\}})$. The process of updating control decisions continues until all communications from other supervisors are received in their entirety, in which case the control decision of supervisor i is updated to $V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$.

Definition 4.1.3. *(UCDR supervisors) A set of local supervisors $\{V_i\}_{i\in\mathcal{I}}$ is called UCDR if $\forall s, s' \in L(\mathbf{G})$, $\forall i \in \mathcal{I}$, $\forall s_j \leq P_j s$, $s'_j \leq P_j s'$, $j = \mathcal{I}\setminus\{i\}$ we have:*

$$[P_i s = P_i s' \wedge s_j \leq s'_j] \Rightarrow V_i(P_i s, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}}) \subseteq V_i(P_i s', \{s'_j\}_{j\in\mathcal{I}\setminus\{i\}})$$

For convenience we define a partial map

$$\Pi_i : \prod_{i\in\mathcal{I}} \Sigma_i^* \to \mathcal{P}(L(\mathbf{G}))$$

defined on $(s_i, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}})$ when V_i is defined, which for $(s_i, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}})$ gives the set of all strings that agent i thinks might have happened in the plant. Formally,

Definition 4.1.4. $\Pi_i(s_i, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}}) = \{s' \in L(\mathbf{G}) |\ P_i(s') = s_i \wedge \forall j \neq i.\ P_j(s') \geq s_j\}$.

The next lemma will come handy in the proof of our main result.

Lemma 4.1.5. *Let $s \in L(\mathbf{G})$ be the string generated by the plant, and $(P_i s, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}})$ be the information agent i has received, through direct observation and communication, where $\forall j \in \mathcal{I}\setminus\{i\}$. $s_j \leq P_j s$. Then,*

$$s \in \Pi_i(P_i s, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}})$$

in particular,

$$s \in \Pi_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}}).$$

Proof. Immediate from the definition. ■

Fusion of local supervisors' decisions is done in the following manner. Define

$$\mathcal{V} : L(\mathbf{G}) \to \mathcal{P}(\Gamma)$$

such that for $s \in L(\mathbf{G})$, $\mathcal{V}(s)$ is equal to

$$\{\bigcap_{i\in\mathcal{I}} V_i(s_i^i, \{s_j^i\}_{j\in\mathcal{I}\setminus\{i\}}) | s_i^i = P_i s \wedge s_j^i \leq P_j s \ (\forall j \in \mathcal{I}\setminus\{i\})\}$$

Evidently $\mathcal{V}$ is a multi-valued function as control decisions might be updated with the arrival of new information at local sites. Since we are interested in UDCR supervisors, there is one value that we are mostly interested in, which corresponds to the case where all communications are received:

$$\bigcap_{i\in\mathcal{I}} V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}}).$$

According to the definition of UCDR supervisors, local supervisors are least restrictive in this case. We denote this "optimal" value by $\mathcal{V}^*(s)$, thereby defining a single-valued map

$$\mathcal{V}^* : L(\mathbf{G}) \rightarrow \Gamma.$$

Based on the above definitions, the problem of supervisor existence is posed as follows.

Definition 4.1.6. *(Unbounded Communication Delay Robust Supervisory Control Problem: UCDR-SC) Given a plant* $\mathbf{G}$ *whose closed behavior is* $L(\mathbf{G})$*, an alphabet* Σ *partitioned into disjoint sets* Σ_i*,* $i \in \mathcal{I}$*, sets of* $\Sigma_{c,i} \subseteq \Sigma_i$*, and a legal non-empty language* $E \subseteq L_m(\mathbf{G})$*, construct a set of UCDR supervisors* $\{V_i\}_{i\in\mathcal{I}}$ *such that* $\mathcal{V}^*$ *is a nonblocking supervisor for* $\mathbf{G}$ *and:*

$$L(\mathcal{V}^*/G) = \overline{E}$$

It is important to note that $L(\mathcal{V}^*/\mathbf{G})$ can also be considered as the case that no delay is present and the channels are ideal. In fact, assuming a particular delay in an example might lead to a system behavior that is strictly smaller than $L(\mathcal{V}^*/\mathbf{G})$, but as our unbounded delay assumption also contains zero delay this optimal behavior will always be achieved.

We define a language property, *unbounded communication delay robust*, as one of the conditions for the existence of a solution for UCDR-SC.

Definition 4.1.7. *(UCDR language) A language $K \subseteq L_m(\mathbf{G})$ is called UCDR with respect to $\mathbf{G}$ if and only if $\forall s \in \Sigma^*$, $\forall \sigma \in \Sigma_{c,i}$ s.t. $s\sigma \in \overline{K}$, the following holds.*

$$\forall s' \in \Pi_i(P_i s, \{P_j s\}_{j \in \mathcal{I} \setminus \{i\}}) : \; s' \in \overline{K} \wedge s'\sigma \in L(\mathbf{G}) \Rightarrow s'\sigma \in \overline{K}$$

4.2 Solution to UCDR-SC Problem

Theorem 4.2.1. *There exists a set of UCDR supervisors that solve UCDR-SC if and only if:*

1. *E is Controllable*
2. *E is Unbounded-Communication-Delay-Robust (UCDR)*
3. *E is $L_m(\mathbf{G})$-closed*

Proof.(If) : Assume E has the three stated properties. We shall construct a set of UCDR supervisors $\{V_i\}_{i \in \mathcal{I}}$, such that $L(\mathcal{V}^*/\mathbf{G}) = \overline{E}$ and $\mathcal{V}^*$ is nonblocking.

For $i \in I$, define V_i in the following way: V_i enables a controllable event in $\Sigma_{c,i}$ at $(s_i, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}})$ if it is legal for all strings in $\Pi_i(s_i, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}})$, formally

$$V_i(s_i, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}}) =$$
$$\{\sigma \in \Sigma_{c,i} |\; \forall s' \in \Pi_i(s_i, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}}).\; s' \in \overline{E} \wedge s'\sigma \in L(\mathbf{G}) \implies s'\sigma \in \overline{E}\} \cup \Sigma_{uc,i}.$$

With this construction, proof of $L(\mathcal{V}^*/\mathbf{G}) = \overline{E}$ is best done by induction on the length of strings:

Base of induction: For $|s| = 0$ ($s = \epsilon$), we have $\epsilon \in \overline{E}$ as E is nonempty, and $\epsilon \in L(\mathcal{V}^*/\mathbf{G})$, so

$$\epsilon \in L(\mathcal{V}/\mathbf{G}) \Leftrightarrow \epsilon \in \overline{E}.$$

Induction Hypothesis: Assume for all $|s| = n$ ($n \geq 0$), it is true that $s \in L(\mathcal{V}^*/G) \Leftrightarrow s \in \overline{E}$. We show that for all $\sigma \in \Sigma$ we have,

$$s\sigma \in L(\mathcal{V}^*/\mathbf{G}) \Leftrightarrow s\sigma \in \overline{E}.$$

Assume the nontrivial case where $s\sigma \in L(\mathbf{G})$, there are two cases.

1. $\sigma \notin \bigcup_{i\in\mathcal{I}} \Sigma_{c,i} = \Sigma_c$: then $s\sigma \in \overline{E}$ (E is controllable), and $\forall i \in \mathcal{I}.\ \sigma \in V_i(P_i s, \{P_j s\}_{\in\mathcal{I}\setminus\{i\}})$ (σ is uncontrollable), and by definition of $\mathcal{V}^*$, $\sigma \in \mathcal{V}^*(s)$, and $s\sigma \in L(\mathcal{V}^*/\mathbf{G})$. Therefore for uncontrollable event σ

$$s\sigma \in L(\mathcal{V}^*/\mathbf{G}) \Leftrightarrow s\sigma \in \overline{E}.$$

2. $\sigma \in \Sigma_{c,i}$: suppose $s\sigma \in \overline{E}$. Thus $s \in \overline{E}$, which yields

$$s \in L(\mathcal{V}^*/\mathbf{G}) \qquad \text{(by induction hypothesis)}$$

We have to show $\sigma \in \mathcal{V}^*(s)$, i.e. $\sigma \in \cap_{i\in\mathcal{I}} V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$. Knowing that σ is only controllable by supervisor i implies

$$\forall k \in \mathcal{I}\setminus\{i\}.\ \sigma \in V_k(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}}).$$

From our construction we have $\sigma \in V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$ if $\forall s' \in \Pi_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$: $s'\sigma \in L(\mathbf{G}) \wedge s' \in \overline{E} \implies s'\sigma \in \overline{E}$, which is guaranteed by the fact that E is a UCDR language. Thus we have

$$\sigma \in V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$$

from which it follows that $\sigma \in \mathcal{V}^*(s)$. Together with $s\sigma \in L(\mathbf{G})$, we conclude that

$$s\sigma \in L(\mathcal{V}^*/\mathbf{G}).$$

This proves one side of the equivalence, that is,

$$s\sigma \in L(\mathcal{V}^*/\mathbf{G}) \impliedby s\sigma \in \overline{E}.$$

For the other side, assume that $s\sigma \in L(\mathcal{V}^*/\mathbf{G})$. It follows that $\sigma \in \mathcal{V}^*(s)$, $s\sigma \in L(\mathbf{G})$ and from the induction hypothesis that $s \in \overline{E}$. By contradiction assume that $s\sigma \notin \overline{E}$. By Lemma 4.1.5 we know that $s \in \Pi_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$. Since $s \in \overline{E}$, $s\sigma \in L(\mathbf{G})$ but $s\sigma \notin \overline{E}$ it follows from our construction that

$$\sigma \notin V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$$

from which it follows that

$$\sigma \notin \mathcal{V}^*(s)$$

a contradiction. The inductive proof is complete.

Now that we have shown $L(\mathcal{V}^*/\mathbf{G}) = \overline{E}$, we need to show the nonblockingness of $\mathcal{V}^*$:

$$\begin{aligned}
\overline{L_m(\mathcal{V}^*/\mathbf{G})} &= \overline{L(\mathcal{V}^*/\mathbf{G}) \bigcap L_m(\mathbf{G})} \\
&= \overline{\overline{E} \bigcap L_m(\mathbf{G})} \\
&= \overline{E} && \text{(by } L_m(\mathbf{G})\text{-closure of } E) \\
&= L(\mathcal{V}^*/\mathbf{G})
\end{aligned}$$

Finally, we have to show $\{V_i\}_{i\in\mathcal{I}}$ is UCDR. If not, assume by contradiction that for $i \in \mathcal{I}$ there exist $s, s' \in L(\mathbf{G})$, $s_j \leq P_j s$ and $\forall j \in \mathcal{I} \setminus \{i\}.\ s'_j \leq P_j s'$, such that

$$P_i s = P_i s' \ \wedge\ \forall j \in \mathcal{I} \setminus \{i\}.\ s_j \leq s'_j \ \wedge\ V_i(P_i s, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}}) \not\subseteq V_i(P_i s', \{s'_j\}_{j\in\mathcal{I}\setminus\{i\}}).$$

In other words, there exists $\sigma \in \Sigma_{c,i}$ such that $\sigma \in V_i(P_i s, \{s_j\}_{j\in\mathcal{I}\setminus\{i\}})$ but $\sigma \notin V_i(P_i s', \{s'_j\}_{j\in\mathcal{I}\setminus\{i\}})$. From $\sigma \notin V_i(P_i s', \{s'_j\}_{j\in\mathcal{I}\setminus\{i\}})$ and the definition of V_i it follows that there exists $s'' \in \Pi_i(P_i s', \{s'_j\}_{j\in\mathcal{I}\setminus\{i\}})$ such that

$$s''\sigma \in L(\mathbf{G}) \setminus \overline{E} \ \wedge\ s'' \in \overline{E}.$$

Since $P_i s = P_i s'$, $\forall j \in \mathcal{I} \setminus \{i\}$. $s_j \leq s'_j$ and $s'' \in \Pi_i(P_i s', \{s'_j\}_{j \in \mathcal{I} \setminus \{i\}})$ it follows that

$$s'' \in \Pi_i(P_i s, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}})$$

combined with the facts that $s''\sigma \in L(\mathbf{G}) \setminus \overline{E}$ and $s'' \in \overline{E}$ yields

$$\sigma \notin V_i(P_i s, \{s_j\}_{j \in \mathcal{I} \setminus \{i\}}) \qquad \text{(by definition of } V_i\text{)}$$

a contradiction. So, $\{V_i\}_{i \in \mathcal{I}}$ must be UCDR.

(Only if) : Controllability. Assume there exists a set of UCDR supervisors satisfying $L(\mathcal{V}^*/\mathbf{G}) = \overline{E}$. Let $s \in \overline{E}$ and $\sigma \in \Sigma_{uc}$ such that $s\sigma \in \overline{E}\Sigma_{uc} \cap L(\mathbf{G})$,

$$\begin{aligned} s\sigma \in \overline{E}\Sigma_{uc} \cap L(\mathbf{G}) &\Rightarrow s\sigma \in L(\mathbf{G}) \ \wedge \ s \in L(\mathcal{V}^*/\mathbf{G}) \\ &\Rightarrow s\sigma \in L(\mathcal{V}^*/\mathbf{G}) \qquad \text{(since } \sigma \text{ is uncontrollable)} \\ &\Rightarrow s\sigma \in \overline{E} \qquad (L(\mathcal{V}/\mathbf{G}) = \overline{E}). \end{aligned}$$

So,

$$\overline{E}\Sigma_{uc} \cap L(\mathbf{G}) \subseteq \overline{E}$$

$L_m(\mathbf{G})$-closure.

$$\begin{aligned} E &= \overline{L(\mathcal{V}^*/\mathbf{G})} \\ &= L_m(\mathcal{V}^*/\mathbf{G}) \qquad (\mathcal{V}^* \text{ is nonblocking}) \\ &= L(\mathcal{V}^*/\mathbf{G}) \cap L_m(\mathbf{G}) \\ &= \overline{E} \cap L_m(\mathbf{G}) \end{aligned}$$

E is a UCDR language. By contradiction, suppose that E is not a UCDR language. It follows that there exist $i \in \mathcal{I}$, $s \in \Sigma^*$, $\sigma \in \Sigma_{c,i}$, with $s\sigma \in \overline{E}$, and $s' \in \Pi_i(P_i s, \{P_j s\}_{j \in \mathcal{I} \setminus \{i\}})$ such that

$$s' \in \overline{E} \ \wedge \ s'\sigma \in L(\mathbf{G}) \setminus \overline{E}.$$

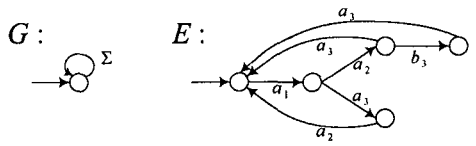

Figure 4.3: Illustrative example, $\Sigma_1 = \{a_1\}$, $\Sigma_2 = \{a_2\}$, and $\Sigma_3 = \{a_3, b_3\}$

Since $L_m(\mathcal{V}^*/\mathbf{G}) = \overline{E}$ it follows that $\sigma \in \mathcal{V}^*(s)$ and $\sigma \notin \mathcal{V}^*(s')$. Since σ can be controlled solely by supervisor i this in turn implies $\sigma \in V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}})$ and $\sigma \notin V_i(P_i s', \{P_j s'\}_{j\in\mathcal{I}\setminus\{i\}})$. Therefore,

$$V_i(P_i s, \{P_j s\}_{j\in\mathcal{I}\setminus\{i\}}) \nsubseteq V_i(P_i s', \{P_j s'\}_{j\in\mathcal{I}\setminus\{i\}})$$

while $P_i s = P_i s'$ and $\forall j \in \mathcal{I} \setminus \{i\}$. $P_j s \leq P_j s'$, contradicting the assumption that $\{V_i\}_{i\in\mathcal{I}}$ are UCDR. ■

We use the following example as an illustrative example.

Example 4.2.2. *Consider the DES E in figure 4.3 as an specification for the distributed plant $\mathbf{G}$ that has no restriction over the occurrence of events, i.e. any event can be executed at any state. Let $\Sigma_{c,1} = \Sigma_{o,1} = \{a_1\}$, $\Sigma_{c,2} = \Sigma_{o,2} = \{a_2\}$, and $\Sigma_{c,3} = \Sigma_{o,3} = \{a_3, b_3\}$. For example, in the initial state only 'a_1' is legal and all other events should be disabled. E is controllable and since marking is not an issue, it remains to check whether E is a UCDR language or not. Taking $\sigma =$ 'a_1', we have to look for $s, s' \in \Sigma^*$ such that $P_1 s = P_1 s'$, $i = 2, 3. P_i s \leq P_i s'$, $s, s', s\sigma \in E$, and $s'\sigma \notin E$. The shortest string that satisfies $sa_1 \in E$ is $s = \epsilon$ and for that there is no s' such that $P_1 s' = \epsilon$. Then we take $s =$ '$a_1 a_2 a_3$' and observe that there is no s' such that $P_1 s' =$ 'a_1', $P_2 s' \geq$ 'a_2', $P_3 s' \geq$ 'a_3', and continuation of s' with 'a_1' produces an illegal sequence. Checking with the same method for $\sigma =$ 'a_2', $\sigma =$ 'a_3', and $\sigma =$ 'b_3' shows E is UCDR according to the definition 4.1.7. Theorem 4.2.1 states that there exists a set of UCDR supervisors that can implement E. The supervisors are designed*

as follows: Supervisor 1 initially enables 'a_1', then disables it upon execution and waits for communication of 'a_2' and 'a_3'. Upon receiving both 'a_2' and 'a_3', in any order, indicating the system has gone back to the initial state, the supervisor enables 'a_1'. Supervisor 2 disables 'a_2' until it receives 'a_1', at that point it enables 'a_2'. It disables 'a_2' after its occurrence and waits for another communication of 'a_1'. Supervisor 3 disables both 'a_3' and 'b_3' until it receives 'a_1'. At that point it enables 'a_3'. If 'a_3' occurs, it waits for 'a_2' to be received, but if 'a_2' is received before execution of 'a_3', the supervisor also enables 'b_3'. The occurrence of 'a_3' disables 'b_3' and 'a_3', and the system goes back to the initial state. The occurrence of 'b_3' leaves 'a_3' enabled and disables 'b_3', at which point execution of 'a_3' is required for supervisor 3 to disable 'a_3' and the system to return to the initial state.

4.3 Properties of UCDR Languages

4.3.1 Relation to other observational classes

In this section we explore the relationship between UCDR class of specification languages with coobservable languages [13], jointly observable languages [27], and weakly jointly observable languages [51]. We first recall the definitions of joint observability and weak joint observability.

Definition 4.3.1. *(Joint observability [27]) Let K, L be two regular languages over Σ, with $K \subseteq L$. Given k subsets of Σ, $i \in \mathcal{I} = \{1, \ldots, k\}.\Sigma_i \subseteq \Sigma$, we call K jointly observable with respect to L and Σ_i, if*

$$\forall s \in K,\ s' \in L \setminus K \Rightarrow \exists i \in \mathcal{I}.\ P_i s \neq P_i s'$$

Definition 4.3.2. *(Weak Joint observability [51]) Let K, L be two regular languages over Σ, with $K \subseteq L$. Given k subsets of Σ, $i \in \mathcal{I} = \{1, \ldots, k\}.\Sigma_i \subseteq \Sigma$, we call K*

weakly jointly observable with respect to L *and* Σ_i *if*

$$\forall s, s' \in \Sigma^*,\ \sigma \in \Sigma, \textit{ such that } s\sigma \in K,\ s'\sigma \in L \setminus K \Rightarrow \exists i \in \mathcal{I}.\ P_i s \neq P_i s'$$

Our first result states that the class of UCDR languages is a proper superclass of coobservable languages.

Proposition 4.3.3. *Every coobservable language is a UCDR language, but the converse is not true.*

Proof. First we prove that violating the UCDR property also violates coobservability, and then we show by an example that a UCDR language may not be coobservable.

Assume that $E \subseteq \Sigma^*$ is not UCDR, so there exist $i \in \mathcal{I}$ and strings $s, s' \in \Sigma^*$ such that $s' \in \Pi_i(P_i s, \{P_j s\}_{j \in \mathcal{I}\setminus\{i\}})$ and $s\sigma \in \overline{E}$, $s'\sigma \in L(\mathbf{G})$ and $s' \in \overline{E}$, but $s'\sigma \notin \overline{E}$ (Bear in mind that only supervisor i controls σ.) $s' \in \Pi_i(P_i s, \{P_j s\}_{j \in \mathcal{I}\setminus\{i\}})$ implies $P_i(s') = P_i(s)$, also $s\sigma \in \overline{E}$, $s' \in \overline{E}$ and $s'\sigma \in L(\mathbf{G}) \setminus \overline{E}$, i.e. E is not coobservable. This concludes the proof that coobservable class is a subclass of UCDR class. E_2 in Figure 4.1 is an example of a UCDR language which is not coobservable. Let $s = `a_1'$, $s' = \epsilon$, and $\sigma = `b_2'$, and we have $P_2 s = P_2 s'$. $s', s \in \overline{E}$, $s\sigma = `a_1 b_2' \in \overline{E}$ but $s'\sigma = `b_2' \notin \overline{E}$ which shows that E_2 is not coobservable. E_2 can be implemented by UCDR supervisors as follows. Supervisor 1 disables 'a_1' after its first occurrence, and supervisor 2 keeps 'b_2' disabled until it receives 'a_1'. So coobservable class of languages is a proper subclass of UCDR class of languages. ■

Proposition 4.3.3 implies that every coobservable language that can be achieved by a set of decentralized supervisors can also be achieved when the local supervisors communicate with each other over a non-ideal channel. In other words, the extra information available for the local supervisors after incorporating communication will not affect the control commands.

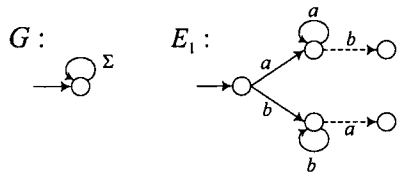

Figure 4.4: A non-UCDR, jointly observable specification language.

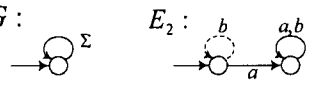

Figure 4.5: A non-jointly observable, UCDR specification language.

Our next result shows that the classes of UCDR and jointly observable languages are not related.

Proposition 4.3.4. *UCDR property and joint observability are incomparable.*

Proof. Let $\Sigma_1 = \Sigma_{c,1} = \{a\}$ and $\Sigma_2 = \Sigma_{c,2} = \{b\}$. E_1, Figure 4.4, shows a language that is jointly-observable but not UCDR. Note that any illegal sequence should contain both 'a' and 'b' while any legal one only contains exclusively either 'a' or 'b', so E is jointly observable. But it is not UCDR as for $s = \epsilon$, $s' =$ 'b', and $\sigma =$ 'a' we have $s' \in \Pi_1(\epsilon, \epsilon) \wedge s\sigma =$ 'a' $\in \overline{E} \wedge s' =$ 'b' $\in \overline{E} \wedge s'\sigma =$ 'ba' $\in L(\mathbf{G}) - \overline{E}$.

On the other hand, E_2, shown in Figure 4.5, is UCDR but not jointly observable. Taking $s =$ 'ab' and $s' =$ 'ba', we have $P_1(s) = P_1(s') \wedge P_2(s) = P_2(s')$ but s is illegal and s' is legal, implying that E_2 is not jointly observable. It can be readily seen that E_2 is UCDR because it can be implemented by UCDR supervisors: supervisor 1 takes no action, while all supervisor 2 needs to do is disable 'b' until it receives 'a' from supervisor 1. ■

Our final result shows that UCDR class is a proper subclass of the class of weakly jointly observable languages.

Proposition 4.3.5. *Every UCDR language is weakly jointly observable, but the con-*

verse is not true.

Proof. We first prove that violating weak joint observability also violates UCDR property, and then by an example we show a weakly jointly observable language that is not UCDR.

Assume that $E \subseteq \Sigma^*$ is not weakly jointly observable. There must exist $s, s' \in \overline{E}$ and $\sigma \in \Sigma_c$ such that $\forall i \in \mathcal{I} : P_i(s) = P_i(s')$, $s\sigma \in \overline{E}$ and $s'\sigma \in L(\mathbf{G}) \setminus \overline{E}$. Assume that $\sigma \in \Sigma_k,\ k \in \mathcal{I}$. From $\forall i \in \mathcal{I} : P_i(s) = P_i(s')$ we have $s' \in \Pi_k(P_k s, \{P_j s\}_{j \in \mathcal{I} \setminus \{k\}})$. Together with $s\sigma \in \overline{E}$ and $s'\sigma \in L(\mathbf{G}) \setminus \overline{E}$ this implies that E is not a UCDR language. Figure 4.4 is an example of a weakly jointly observable language (since it is jointly observable) which is not UCDR. ■

4.3.2 Closure properties of UCDR languages

Proposition 4.3.6. *The union of two UCDR languages is not a UCDR language.*

Proof. By counterexample let $\Sigma = \{a_1, a_2\}$, $\Sigma_{c,1} = \Sigma_{o,1} = \{a_1\}$, and $\Sigma_{c,2} = \Sigma_{o,2} = \{a_2\}$.

$$L(\mathbf{G}) = \Sigma^*, \quad E_1 = \overline{a_1 a_2} + \overline{a_2 a_1},\ E_2 = \overline{a_1 a_2 a_1}$$

It is clear that E_1 and E_2 are both UCDR, but their union is not. To see that $E = E_1 \cup E_2$ is not UCDR let $s =$ '$a_1 a_2$', $s' =$ '$a_2 a_1$', and $\sigma =$ 'a_1', and observe that $s' \in \Pi_1(a_1, a_2) \wedge s\sigma =$ '$a_1 a_2 a_1$' $\in \overline{E}$. Should E is UCDR $s'\sigma$ must be in $\overline{E}$, but $s'\sigma =$ '$a_2 a_1 a_1$' $\in L(\mathbf{G}) - \overline{E}$, which proves that the union of two UCDR languages is not UCDR in general. ■

Proposition 4.3.7. *The intersection of two closed UCDR languages is a UCDR language.*

Proof. We proves that if the intersection of two closed languages fails the UCDR property, then none of the two languages are UCDR.

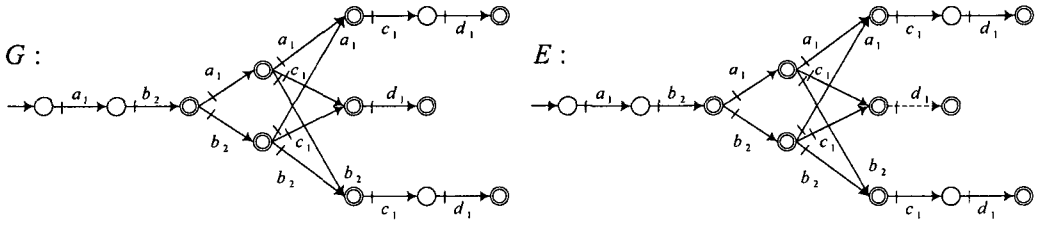

Figure 4.6: Plant and specification models

Assume that $E = E_1 \cap E_2 \subseteq \Sigma^*$ is not UCDR, so there exist $i \in \mathcal{I}$ and strings $s, s' \in \Sigma^*$ such that $s' \in \Pi_i(P_i s, \{P_j s\}_{j \in \mathcal{I} \setminus \{i\}})$ and $s\sigma \in \overline{E}$, $s'\sigma \in L(\mathbf{G})$ and $s' \in \overline{E}$, but $s'\sigma \notin \overline{E}$. As $E = E_1 \cap E_2$ and $s, s' \in \overline{E} = E$, then $s, s' \in E_1 \wedge s, s' \in E_2$ which means that none of E_1 and E_2 are UCDR. ■

Proposition 4.3.6 and Proposition 4.3.7 show that UCDR property behaves much like how observability property behaves, but the results of the above propositions are useful after one develops an algorithm to check the UCDR property for a given language. To check the UCDR property one has to check the condition presented in Definition4.1.7 on the strings of a language, which in general are not finite. As such, developing an algorithm to check the UCDR property and prove that it terminates in finite time is one of our future works.

4.4 An Example

It is interesting to compare the results of introducing delay in communication between supervisors with the case when communication is instantaneous. First, we have to present an example that meet the requirements of this chapter, namely to assign each event to only one supervisor and to have all the events observable. So, we assign 'a_1', 'c_1' and 'd_1' to supervisor 1, and 'b_2' to supervisor 2. Plant model, $\mathbf{G}$, and specification model, E, are given in Figure 4.6. When the communication is instantaneous with

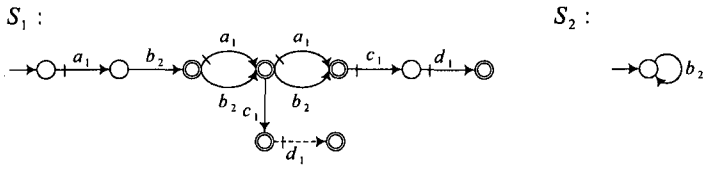

Figure 4.7: Supervisors S_1 and S_2

the supervisors given in Figure 4.7 the specification will be achieved. Specification, E, is (1) Controllable, and (2) $L_m(\mathbf{G})$-closed. To check whether it is a UCDR language or not, take $\sigma = `d_1'$, we look for $s, s' \in \Sigma^*$ such that $P_1 s = P_1 s'$, $P_2 s \leq P_2 s'$, $s, s', s\sigma \in E$, and $s'\sigma \notin E$. The only two strings that satisfy $s'\sigma \notin E$ are $s'_1 = a_1 b_2 a_1 c_1$ and $s'_2 = a_1 b_2 b_2 c_1$, and there are no string that satisfies, $P_1 s = P_1 s'$, $P_2 s \leq P_2 s'$, and $s\sigma \in E$. So, E is a UCDR language; So, it is possible to achieve E with a set of UCDR supervisors using the construction proposed in the proof of Theorem 4.2.1. This example shows that it is possible to achieve the same system behavior after introduction of delay in a previously solved problem, with the assumption of ideal channels, if the UCDR property is satisfied.

4.5 Conclusion

In this chapter we proposed a new language property called *unbounded-communication-delay-robust* (UCDR), and a new method of constructing local supervisors with the assumption of delay. We provided the necessary and sufficient conditions for the existence of distributed nonblocking supervisors achieving a given specification with the assumption of unbounded delay in communication network, and formally proved it. We also investigated the relation of UCDR languages with other observational languages, and argued that the property know as joint-observability does not capture the entire problem of distributed supervisory control with unbounded delay.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In this thesis we studied robust supervisory control of discrete event systems and problems associated with it. We summarize our work as follows.

We considered Robust Decentralized Supervisory Control (RDSC for short) as a natural extension of previous work in the area of robust supervisory control with respect to plant model, when the system under control is distributed in nature and thus no single supervisor is likely to be able to control the system. In this case, we would like to synthesize a set of decentralized supervisors (rather than one centralized supervisor) that can solve the given robust supervisory control problem. We derived necessary and sufficient conditions for the existence of a solution to RDSC problem and formally proved them. We observed that this approach is too restrictive. The nature of the coobservability condition, one of the conditions for the existence of a solution to RDSC problem, requires 'each' local supervisor to see enough of the plant to make its decisions. This observation led us to believe that having communicating local supervisors is a more realistic assumption when one tries to tackle the problem

of robust supervisory control in distributed systems.

To this end, we formulated a robust supervisory control with respect to communication channel delay that arises from not-negligible delay in the communication between local supervisors when a global objective has to be achieved. This problem, rarely touched by previous researchers, required us to redefine supervisor and supervisory decision making rules to account for the uncertainty associated with delay in communication channels. We defined the supervisory action in this setting when local supervisors should also take into account the possible future communications, and the fact that their view of the system might be out-of-date.

We assumed that delay is unbounded but it is finite, meaning that any message sent from a local supervisor will be received by any other local supervisors after a finite but unknown delay. We then formulated Unbounded Communication Delay Robust Supervisory Control (UCDR-SC) problem and introduced a new language property called *unbounded-communication-delay-robust* (UCDR for short) as one of the conditions to solve it. UCDR property ensures that a particular language can be achieved by appropriate supervisors in the presence of delay. We then presented necessary and sufficient conditions under which a UCDR-SC problem can be solved, and formally proved them.

Finally, we showed that the new class of UCDR languages has some interesting relations with other observational languages such as coobservable languages which we believe makes this class of languages a suitable subject for future researchers.

5.2 Future Work

We understand that this work is just some initial steps toward a comprehensive study on the robustness of decentralized supervisory control problems and mention some of

the challenges in the way.

1. First and foremost, it is essential to find an algorithm to check the unbounded-communication-delay-robust property of a given language to see if it can be implemented in the presence of delay, and also to open the way to come up with an algorithm to give a UCDR sublanguage of any given language. UCDR property imposes a condition on the strings of a language and thus can be checked by searching all the possible strings of a language, however, as we work with regular languages, this method will not yield an algorithm as the termination of the search can not be guaranteed. At this time, it is unclear whether such an algorithm exists or the problem of checking UCDR property is undecidable. One might try to find an algorithm on the structure of the language, which is finite, but should also account for the complications that having self-loops and loops brings into checking the UCDR property. On the other hand, one might try the reduction to one of the known undecidable problems [28], and proves that such an algorithm does not exist.

2. We assumed that every event is observable by at least one local supervisor. While this assumption in general is not restricting, removing that would lead to a general solution of the problem. Also we assumed that any event can be controlled by at most one local supervisor which removes the common events decision fusion rules from our discussion. It seems that the disjunctive fusion rule ensures safety but formal proof should be presented.

3. It might be interesting to investigate *Bounded Communication Delay Robust* Supervisory Control problem when the delay is finite and is known to be smaller than $K \in \mathbb{N}$. It is interesting to see if the resulting class of languages that can be implemented by this assumption is larger than the class of UCDR languages.

4. It is interesting to see if the results of our work can be presented in a single formulation where the system bears uncertainties in both the system structure and in the delay in communication.

Bibliography

[1] P. Ramadge, "Control and Supervision of Discrete Event Processes," Ph.D. dissertation, Dept. of Elec. Eng., Univ of Toronto, 1983.

[2] Y. L. Chen and G. Provan, "Model-Based Control Fault-Tolerant Reconfiguration for General Netwrok Topologies," in *IEEE MICRO*, vol. 21, no. 5, September 2001, pp. 64–76.

[3] M. Mossaei and S. Hashtrudi-Zad, "Fault Recovery in Contorl Systems: A Madular Discrete-Event Approach," in *Proceedings of International Conference on Electrical and Electronics Engineering (CINVESTAV/IEEE)*, Mexico, September 2004, pp. 445–450.

[4] 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.

[5] S. Bourdon, M. Lawford, and W. Wonham, "Robust Nonblocking Supervisory Control of Discrete-Event Systems," in *Proceedings of the 2002 American Control Conference*, May 2002, pp. 730–735.

[6] R. Boel and J. van Schuppen, "Decentralized Failure Diagnosis for Discrete-Event Systems with Costly Communication Between Diagnosers," in *Proceedings of Sixth International Workshop on Discrete Event Systems*, 2002, pp. 175 - 181.

[7] K. Rudie, S. Lafortune, and F. Lin, "Minimal Communication in a Distributed

Discrete-Event System," *IEEE Transactions on Automatic Control*, vol. 48, pp. 957 – 975, June 2003.

[8] F. Lin, K. Rudie, and S. Lafortune, "Minimal Communication for Essential Transitions in a Distributed Discrete-Event System," *IEEE Transactions on Automatic Control*, vol. 52, pp. 1495 – 1502, August 2007.

[9] 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.

[10] F. Lin and W. Wonham, "On Observability of Discrete-Event Systems," *Inf. Sci.*, vol. 44, pp. 173–198, 1988.

[11] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, "Supervisory Control of Discrete-Event Processes with Partial Observations," *IEEE Transactions on Automatic Control*, vol. 33, no. 3, pp. 249–260, March 1988.

[12] F. Lin and W. Wonham, "Decentralized Supervisory Control of Discrete-Event Systems," *Inf. Sci.*, vol. 44, pp. 199–224, 1988.

[13] K. Rudie and W. Wonham, "Think Globally, Act Locally: Decentralized Supervisory Control," *IEEE Transactions on Automatic Control*, vol. 37, pp. 1692–708, 1992.

[14] J. Titsiklis, "On the control of discrete-event dynamical systems," *Math. Control, Signal, System*, vol. 2, pp. 95–107, 1989.

[15] K. Rudie and J. Willems, "The Computational Complexity of Decentralized Discrete-Event Control Problems," *IEEE Transactions on Automatic Control*, vol. 40, no. 7, pp. 1313–1319, July 1995.

[16] J. Prosser, M. Kam, and H. Kwatny, "Decision Fusion and Supervisor Synthesis in

Decentralized Discrete-Event Systems," in *Proceedings of the 1997 American Control Conference*, June 1997, pp. 2251–2255.

[17] T.-S. Yoo and S. Lafortune, "New Results on Decentralized Supervisory Control of Discrete-Event Systems," in *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 1, Sydney, Australia, December 2000, pp. 1–6.

[18] ——, "A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems," *Discrete Event Dynamic Systems: Theory Application*, vol. 12, no. 3, pp. 335–377, 2002.

[19] S. Takai, R. Kumar, and T. Ushio, "Characterization of Co-Observable Languages and Formulas for Their Super/Sublanguages," *IEEE Transactions on Automatic Control*, vol. 50, no. 4, pp. 434–447, April 2005.

[20] Y. Huang, K. Rudie, and F. Lin, "Decentralized Control of Discrete-Event Systems When Supervisors Observe Particular Event Occurrences," *IEEE Transactions on Automatic Control*, vol. 53, pp. 384–388, February 2008.

[21] S. Ricker and K. Rudie, "Incorporating Communication and Knowledge into Decentralized Discrete-Event Systems," in *Proceedings of the 38th IEEE Conference on Decision and Control*, Phoenix, Arizona USA, December 1999, pp. 1326–1332.

[22] G. Barrett and S. Lafortune, "Decentralized Supervisory Control with Communicating Controllers," *IEEE Transactions on Automatic Control*, vol. 45, no. 9, pp. 1620–1638, September 2000.

[23] W. Wang, S. Lafortune, and F. Lin, "Minimization of Communication in Distributed Dscrete Event Systems," in *Proocidings of the European Control Conference*, July 2007, pp. 4960–4967.

[24] S. Balemi, "Communication Delays in Connections of Input/Output Discrete Event Processes," in *Proceedings of the 31st IEEE Conference on Decision and Control*, vol. 4, 1992, pp. 3374 – 3379.

[25] R. Debouk, S. Lafortune, and D. Teneketzis, "On the Effect of Communication Delays in Failure Diagnosis of Decentralized Discrete Event Systems," in *Proceedings of the 39th IEEE Conference on Decision and Control*, vol. 3, December 2000, pp. 2245 – 2251.

[26] S. Ricker and J. van Schuppen, "Asynchronous Communication in Timed Discrete-Event Systems," in *Proceedings of the 2001 American Control Conference*, vol. 1, 2001, pp. 305 – 306.

[27] S. Tripakis, "Undecidable Problems of Decentralized Observation and Control," in *Proceedings of the 40th IEEE Conference on Decision and Control*, vol. 5, December 2001, pp. 4104 – 4109.

[28] M. Garey and D. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness.* Freeman, 1979.

[29] S. Tripakis, "Decentralized Control of Discrete-Event Systems with Bounded or Unbounded Delay Communication," *IEEE Transactions on Automatic Control*, vol. 49, pp. 1489 – 1501, September 2004.

[30] R. Sengupta and S. Tripakis, "Decentralized Diagnosability of Regular Languages is Undecidable," in *Proceedings of the 41st IEEE Conference on Decision and Control*, vol. 1, December 2002, pp. 423 – 428.

[31] W. Qiu, R. Kumar, and S. Jiang, "On Decidability of Distributed Diagnosis Under Unbounded-Delay Communication," *IEEE Transactions on Automatic Control*, vol. 52, pp. 114 – 116, January 2007.

[32] S. Park and K. Cho, "Delay-Robust Supervisory Control of Discrete-Event Systems with Bounded Communication Delays," *IEEE Transactions on Automatic Control*, vol. 51, pp. 911 – 915, May 2006.

[33] ——, "Supervisory Control of Discrete Event Systems with Communication Delays and Partial Observations," *Systems and Control Letters*, vol. 56, pp. 106–112, February 2007.

[34] ——, "Decentralized Supervisory Control of Discrete Event Systems with Communication Delays Based on Conjunctive and Permissive Decision Structures," *Automatica*, vol. 43, pp. 738–743, April 2007.

[35] J. Cury and B. Krogh, "Design of Robust Supervisors for Discrete Event Systems With Infinite Behaviors," in *Proceedings of the 35th IEEE Conference on Decision and Control*, 1996, pp. 2219–2224.

[36] ——, "Robustness of Supervisors for Discrete-Event Systems," *IEEE Transactions on Automatic Control*, vol. 44, pp. 376–379, 1999.

[37] S. Takai, "Synthesis of Maximally Permissive and Robust Supervisors for Prefix-Closed Language Specifications," *IEEE Transactions on Automatic Control*, vol. 47, no. 1, pp. 132–136, January 2002.

[38] ——, "Maximazing Robustness of Supervisors for Partially Observed Discrete Event Systems," *Automatica*, vol. 40, no. 3, pp. 531–535, 2004.

[39] S. Park and J. Lim, "Robust and Nonblocking Supervisor for Discrete-Event Systems with Model Uncertainty Under Partial Observation," *IEEE Transactions on Automatic Control*, vol. 45, no. 12, pp. 2393–2396, December 2000.

[40] S. Bourdon, M. Lawford, and W. Wonham, "Robust Nonblocking Supervisory Control of Discrete-Event Systems," *IEEE Transactions on Automatic Control*, vol. 50, no. 12, pp. 2015–2021, December 2005.

[41] S. Takai, "Maximally Permissive Robust Supervisors for a Class of Specification Languages," in *Proceedings of the 5th IFAC Conference on System Structure and Control*, vol. 2, Nantes, France, July 1998, pp. 429–434.

[42] A. Saboori and S. Hashtrudi-Zad, "Robust Supervisory Control and Blocking Invariant Languages," in *Proceedings of the 43th Annual Allerton Conference on Communication, Control, and Computing*, University of Illinois at Urbana-Champaign, September 2005, pp. 658–667.

[43] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems.* Klumer Academic Publishers, Norwell, MA, 1999.

[44] J. Hopcraft and J. Ullman, *Introduction to Automata Theory, Languages, and Compotation.* Addison-Wesley, Reading, MA, 1979.

[45] W. Wonham, "Supevisory Control of Discrete-Event Systems," Sytems Control Group, Dept. of Electrical and Computer Engineering, University of Toronto, 2006, ECE 1636F/1637S 2006-7.

[46] P. Ramadge and W. Wonham, "Supervisory Control of a Class of Discrete Event Systems," *SIAM Journal of Control Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[47] F. Lin, "On Controllability and Observability of Discrete Event Systems," Ph.D. dissertation, Dept. of Elec. Eng., Univ of Toronto, 1987.

[48] W. Wonham and P. Ramadge, "Modular Supervisor Control of Discrete Event Systems," *Math. Control Signals Syst.*, vol. 1, no. 1, pp. 13–30, 1988.

[49] K. Rudie and W. Wonham, "Think Globally, Act Locally: Decentralized Supervisory Control," in *Proceedings of the 1991 American Control Conference*, June 26-28 1991, pp. 898–903.

[50] A. Saboori, "Fault Recovery using Discete Event Models," Master's thesis, Dept. of Elec. and Comp. Eng., Concordia University, June 2005.

[51] A. Mannani and P. Gohari, "Decentralized Supervisor Control of Discrete-Event systems Over Communication Networks," *IEEE Transactions on Automatic Control*, vol. 53, no. 2, pp. 547–559, March 2008.

[52] P. Ramadge and W. Wonham, "Supervission of Discrete Event Precesses," in *Proceedings of the 21st IEEE Conf. Decision Control*, vol. 3, December 1982, pp. 1228–1229.

[53] S. Takai and R. Kumar, "Synthesis of Inference-Based Decentralized Control for Discrete Event Systems," *IEEE Transactions on Automatic Control*, vol. 53, pp. 522–534, March 2008.

[54] S. Ricker and K. Rudie, "Knowledge Is a Terrible Thing to Waste: Using Inference in Discrete-Event Control Problems," *IEEE Transactions on Automatic Control*, vol. 52, pp. 428 – 441, March 2007.

[55] S. Tripakis, "Decentralized Control of Discrete Event Systems with Bounded or Unbounded Delay Communication," in *Proceedings of Sixth International Workshop on Discrete Event Systems*, 2002, pp. 18 – 25.