COMPUTATIONS ON DISTRIBUTED DISCRETE-EVENT SYSTEMS

by

Kurt Ryan Rohloff

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2004

Doctoral Committee:

Professor Stéphane Lafortune, Chair
Associate Professor Kevin J. Compton
Assistant Professor Satyanarayana Lokam
Professor Demosthenis Teneketzis

For my grandparents:
Alfred Loring Ryan
Gertrude Reilly Ryan
Howard Emerson Rohloff
Alice Ongley Rohloff

ACKNOWLEDGEMENTS

This thesis primarily reports on work performed while the author was under the supervision of Professor Stéphane Lafortune at the University of Michigan. Chapter VI reports on work commenced while the author was temporarily employed at CWI in Amsterdam under the direction of Professor Jan H. van Schuppen.

Excluding Chapter V and Chapter VI, financial support for this thesis was provided in part by NSF grant CCR-0082784. Support from NSF grant CCR-0325571 is also acknowledged. Chapter V was supported in part by a GAANN fellowship for the 2000-2001 academic year. Financial support in part for Chapter VI was made available by the European Commission through the project Control and Computation (IST-2001-33520) of the Information Society Technologies Program.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF APPENDICES

Appendix

CHAPTER I

INTRODUCTION

Distributed systems are becoming increasingly more common and important in the modern world. As such there has been considerable interest lately in both the discrete-event systems and computer science communities in problems related to distributed systems. Examples of such systems include communication networks (wired and ad hoc), sensor networks, intelligent transportation systems, automated manufacturing systems and distributed software systems. In particular, computational issues are becoming increasingly relevant to the control and verification of these complex distributed systems. This thesis investigates the computational difficulty of problems associated with verification and control of distributed systems modelled as discrete-event systems and proposes methods for avoiding this computational difficulty.

Straightforward procedures to solve many verification and control problems for monolithic systems are well known in the discrete-event systems literature [10], but the current standard methods for performing control and verification tasks on distributed systems generally involve converting the distributed system models into a monolithic one through the use of composition operations. Unfortunately, the conversion of distributed models to monolithic ones suffers due to the well-known "state

explosion" problem. That is, when several finite-state systems are composed, the size of the state space of the converted system is potentially exponential in the number of components. Therefore, it may be desired that the system models be kept modular whenever possible. However, control and verification problems for distributed systems are poorly understood if the monolithic conversions are to be avoided by the solution methods for these problems. Consequently, there are few known algorithmic methods for control and verification specialized for distributed systems.

An important subclass of distributed system problems are decentralized control problems where a (possibly monolithic) system is supervised by a set of controllers. As system operation evolves, the local controllers make observations of the system's behavior and maintain estimates of the system's state based on the events the controllers observe. The controllers use their knowledge of the estimated system states to determine their local control actions which are combined globally to determine what system behavior is allowed. Local controllers usually do not have full knowledge of the system's behavior and must commonly decide their control actions after only viewing an observable subset of the system behavior. Decentralized control is commonly the most efficient or only method to control a system too large or complex to be controlled by a single centralized controller. Like the general class of distributed system problems, there are many important and not well understood decentralized control problems for which algorithmic methods need to be developed.

The first main contribution of this thesis is to show that many important and seemingly simple verification and control problems for decentralized control and distributed systems are PSPACE-complete. This means that these problems are most likely computationally intractable and therefore there are no time-efficient general methods for solving these problems. However, because of the overriding importance

of many of these problems, efficient methods for dealing with these problems still need to be developed. Therefore, with this groundwork on the fundamental difficulty of these problems the rest of this thesis focuses on methods for avoiding the computational difficulty of these problems.

The second main contribution of this thesis is a set of online decentralized control protocols that have a common sufficient and easily testable safety condition. These decentralized control protocols attempt to allow the controlled system to achieve behavior that is safe and in a sense locally maximal with respect to a given specification. There is no known method to synthesize safe globally maximal behavior for decentralized control or even whether global maximal behavior is always possible. The sufficient safety condition for this decentralized control protocol can be thought of as a decentralized actuator selection problem. The control protocols shown here are based on a new local observation state estimation procedure that accounts for past local control actions and are developed for the general decentralized control setting of [83]. In this setting controllable events are assigned to follow a "fusion by union" or "fusion by intersection" global control policy.

The third main contribution of this thesis is to show heuristic methods for efficiently approximating minimal sensor selections that are sufficient to solve controller existence problems. These minimal sensor selections, even for centralized systems, are shown to be computationally difficult to approximate as there are most likely no polynomial time approximation methods. It is shown how to convert sensor selection problems to a class of edge colored directed graph st-cut problems. Several heuristic methods are shown that compute sufficient sensor selections given a system and specification. An open decentralized minimal communication controller problem is also discussed and it is shown how to convert this problem to an edge colored directed

graph st-cut problem.

The fourth main contribution of this thesis is a method for dealing with distributed systems where the system modules are isomorphic to one another. A form of symmetry called state permutation symmetry for distributed systems is introduced where the modules are identical to one another except for a relabelling of state transitions. This symmetry definition induces the construction of a special reduced state quotient automaton used for the verification of global μ-calculus propositions when the modules interact.

The fifth main contribution of this thesis is a discussion on a special subclass of permutation symmetric systems where the subsystems are exact copies of one another except for a relabelling of private events. These similar module systems allow for a quotient automaton with an even more reduced state space for testing state reachability properties. A decomposition procedure that efficiently converts composed global models of these systems into their component subsystems is shown. This special subclass of distributed systems has a concise set of necessary and sufficient controller existence properties that allow for the time-efficient synthesis of control policies for local and global specifications.

This thesis generally follows the framework of supervisory control theory for discrete-event systems introduced in the seminal work of [54, 55]. The interested reader may consult the text [10] for a general introduction to discrete-event systems and supervisory controls theory. Because this thesis draws on several areas with large and diverse literatures, a survey of previous relevant results is given in Chapter II. A brief introduction to the necessary notation and background information is presented in Chapter III. The computational complexity contributions of this thesis are shown in Chapter IV. The decentralized online control protocols contribution is

discussed in Chapter V and the sensor selection methods are presented in Chapter VI. The contributions on permutation symmetric systems are in Chapter VII and the discussion of the similar module systems is in Chapter VIII. The thesis closes with a general discussion of the results and areas for future research in Chapter IX.

CHAPTER II

SURVEY OF PREVIOUS RESULTS

2.1 Chapter Overview

This chapter presents a survey of results in the discrete-event systems and computer science literatures relevant to the work discussed in this thesis. First, a general survey of earlier work on the supervisory control of monolithic and distributed discrete-event systems is shown. Then, a survey of results related to connections between distributed discrete-event systems and the theory of computation from computer science is given. A survey of results related to supervisory control is given along with a discussion of results related to sensor selection problems. This section closes with a survey of results related to the exploitation of symmetry for the verification and control of discrete state systems.

2.2 Supervisory Control and Distributed Discrete-Event Systems

The supervisory control of discrete-event systems framework used in this thesis is based on a finite automaton modelling formalism that is generally considered the simplest one for discrete-event systems that is expressive enough to model the behavior of real-world systems in a reasonable manner. This framework was introduced in

the seminal work of [54, 55] and later innovative work includes [13, 43, 44, 68]. The text [10] gives a general introduction to supervisory control and discrete-event system theory. When discussing distributed systems, the parallel composition operation is used to model interaction between these systems, which is currently the standard method to model the interaction between automata in discrete-event system theory.

Distributed system problems are currently receiving much attention from the discrete-event systems research community; see, e.g., [7, 11, 29, 31, 40, 41, 43, 45, 46, 52, 78, 81, 80]. Some of the earlier work related to the supervision of distributed systems can be seen in [43, 45, 53, 81, 78]. Properties of distributed discrete-event systems where the modules have disjoint event sets are investigated in [52, 53]. Various local specification and concurrent supervision problems, respectively, are investigated in [29, 31]. The supervision of distributed systems using specific architectures is discussed in [40, 41]. A form of distributed control where each controller in a distributed system has a different objective is discussed in [11]. Situations when local non-blocking behavior implies global non-blocking behavior (a property later called non-conflictingness) are discussed in [46]. Incremental system verification for distributed discrete-event systems with special system assumptions are discussed in [7].

There has also recently been a large volume of work investigating connections between theoretical computer science and the supervisory control of discrete-event systems. Many of the system models used in supervisory control are based on ideas that originated from computer science. Some notable examples of crossover work between control theory and theoretical computer science include [5, 6, 8, 19, 23, 47, 56, 67]. This thesis discusses and explores further connections between these fields.

2.3 Computational Complexity and Discrete-Event Systems

With the exception of [19], there has been little work investigating the computational complexity of distributed system supervisory control problems which shows some NP-hardness results for distributed supervision problems. Some of the problems discussed in [19] (among others) are shown in Chapter IV of this thesis to be PSPACE-complete. This gives a more exact understanding of the computational complexity of these problems as PSPACE-complete problems are known to be NP-hard, but not all NP-hard problems are PSPACE-complete. The complexity of verification for distributed systems using more complicated models such as temporal logic and alternating tree automata are discussed in [22, 36, 37, 76]. Although researchers in computer science have shown more complicated verification problems are PSPACE-complete, Chapter IV shows that the supposedly simpler verification problem for finite-state automata distributed systems is PSPACE-complete. Chapter IV reports in part on work that was originally published in [58, 61, 64].

For a deeper introduction to the theory of computation and the properties of the PSPACE problem class, consult one of the standard textbooks such as [15, 18]. Some of the main proofs in Chapter IV are inspired by the automata intersection problem initially investigated by Kozen [35]. Some other work on the automata intersection problem is also shown in [3, 39]. Also from the computer science community, [9] discusses the difficulty of coupled automata problems but does not discuss computational complexity nor the specific problems discussed here.

2.4 Decentralized Control

Several decentralized control laws for various discrete event systems have been introduced in the literature. See for example [1, 13, 31, 38, 44, 48, 51, 68, 71, 78, 83].

Several investigations have addressed decentralized control in a similar manner as in this thesis but by allowing the local controllers to communicate [4, 56, 79]. Online supervision for monolithic systems is discussed in [20, 71] under assumptions different from discussed herein. There has been little or no discussion in the control literature related to the online supervision of distributed systems. It has recently been discovered in [38, 73] that the problem of synthesizing *safe* and *non-blocking* decentralized controllers for a discrete event system is undecidable, so when discussing safe controller existence and synthesis problems, this thesis focuses on the safe controller problem.

In Chapter V several online decentralized control protocols are shown that use a new state estimation function. Generally, the decentralized control framework of [43, 68] is assumed throughout this thesis, but in Chapter V the general decentralized control framework of [83] is used. The generalized framework of Chapter V assumes that controllable events follow either a "fusion by union" or "fusion by intersection" global control policy. This distinction between the two control settings will be discussed further in Chapter V.

Several of the new decentralized control protocols of Chapter V attempt to maximize the set of locally enabled events. Ben Hadj-Alouane, et al. [20] have done work with maximal online control policies for centralized control systems, but their work has not yet been applied to general decentralized control framework as is done in Chapter V. This chapter reports in part on work that was originally published in [60, 63].

2.5 Sensor Selections

Variations of the sensor selection problem using frameworks similar to that of in Chapter VI have been investigated in [12, 21, 30, 82]. The problem of designing an observation function that is as coarse as possible is discussed in [12]. A projection mapping is assumed in [12] that is different from the standard natural projection operation used as the observation function in this thesis. Furthermore, optimization and approximation methods are not discussed in [12]. The optimization of the observable event set is discussed in [21] for achieving both observability and normality for a problem setting very similar to that discussed here. An exponential-time algorithm is also shown in [21] for giving an optimal observable set.

It is shown in [82] that the decision problem version of the sensor selection problem is NP-complete, so there are most likely no algorithms to solve this problem in polynomial time. Therefore, solving this optimization problem and finding the absolute lowest cost observation set for large industrial systems cannot always be done in a reasonable amount of time. An optimal sensor selection problem is also discussed in [32], except that the observation function is different from the one assumed in this thesis. Chapter VI reports in part on work originally presented in [33, 65].

2.6 Symmetric Systems

Many real-world distributed systems often exhibit a degree of similarity such that they can be modelled as interacting distributed subsystems that are exact copies of one another. Because it has been found that in general verification and control of distributed systems is complicated if not impossible (see Chapter IV) one would hope that if there is a degree of symmetry between the modules, control and verification problems would become simpler. Problems of interacting similar systems

have received very little attention in the literature except for [17]. Group-theoretic methods are used in [17] to define classes of states and transitions in centralized systems that are equivalent under defined permutation operations. These classes of equivalent states and transitions of a system are used to define a quotient automaton that has a state space smaller than the original system. This quotient automaton is used to perform various controller synthesis operations for the original system. Unfortunately, finding the optimal partition of state equivalence classes as in [17] is computationally difficult and is at least as difficult as the graph isomorphism problem [26]. This has induced several authors to attempt to design distributed systems with special architectures so that certain types of symmetry are guaranteed to occur *a priori*, therefore avoiding intensive symmetry optimality verification computations [17, 62].

There are several definitions of the μ-calculus, but for the sake of regularity, the discussions in this thesis are based on the commonly accepted μ-calculus in [14]. The μ-calculus is a very general language for making specifications developed by the formal methods community that is more general than many other common specification logics such as LTL, CTL and CTL*. Researchers have begun to investigate uses of the temporal logics in the field of discrete-event systems [34, 42, 72]. The verification of LTL propositions for discrete-event systems is discussed in [72], and synthesis methods for this setting are discussed in [42]. The decidability of testing whether classes of temporal formulas within CTL* hold in a discrete-event system is discussed in [34]. A version of the μ-calculus that allows one to solve some supervisory controls problems has also been presented in [1].

The use of symmetry for testing μ-calculus propositions has been discussed in [16] with a different system model and assumptions on the μ-calculus propositions used.

It is discussed in [14] that there are most likely no polynomial time algorithms for the μ-calculus model checking problem, so any methods for reducing the difficulty of testing μ-calculus propositions are very useful.

Chapter VII introduces a symmetry definition for a special class of discrete-event systems where the automata are isomorphic to one another. It is shown in this chapter how to more efficiently perform verification tasks for these symmetric discrete-event systems with respect to μ-calculus specifications. The permutation symmetry definition induces a state equivalence class that can be used to construct a quotient automaton. This quotient automaton may not always be the most efficient structure for performing verification, but generally offers large reductions in the difficulty of verifying many system properties. The work in Chapter VII was originally reported in part in [59].

Chapter VIII investigates a special case of the systems discussed in Chapter VII where the event alphabets are partitioned into sets of global and private events. These systems were briefly mentioned in [17], but Chapter VIII presents work that was in part originally presented in [57, 62].

CHAPTER III

NOTATIONAL AND BACKGROUND INFORMATION

3.1 Chapter Overview

To aid the reader, this chapter reviews the notation used throughout this thesis. First, the automata models used for distributed discrete-event systems are presented. The supervisory control model is then shown, both in the standard setting of [43] and in the generalized setting of [83]. A review of the theory of computational complexity is presented and this chapter closes with a presentation of several results from computer science on the computational difficulty of several important decision problems for automata. For more background information on discrete-event systems and supervisory control, please consult [10]. See [15, 18, 27] for background information on the theory of computation.

3.2 Distributed System Model

This thesis builds on previous work on automata problems from both the discrete-event systems and computer science communities. Although the notation for automata problems used in computer science and discrete-event systems is similar, there are subtle differences. In this thesis the notation of computer science theory

is generally used when automata intersection problems are discussed and the notation of supervisory control is used when work related to discrete-event systems is discussed.

The basic automaton system model G used in this thesis is defined is a 5-tuple $(X^G, x_o^G, \Sigma^G, \delta^G, X_m^G)$ where X^G is the set of states, x_o^G is the initial state, Σ^G is the automaton event set, $\delta^G : X^G \times \Sigma^G \to X^G$ is the (possibly partial) state transition function, and X_m^G is the set of "final" or "marked" states.

Deterministic systems and specification automata are generally assumed in this thesis. The motivation for this distinction is discussed in Section 3.4. However, this thesis does make use of nondeterministic automata at times. In this case the transition function δ^G is defined such that $\delta^G : X^G \times \Sigma^G \to 2^{X^G}$ where for a state $x \in X^G$ and an event $\sigma \in \Sigma^G$, $\delta^G(x, \sigma) \subseteq X$ represents the set of states G could be in if a σ event were to occur at state x. Also, the notation is sometimes used for both deterministic and nondeterministic automata such that for two states $x, y \in X^G$ and an event $\sigma \in \Sigma$, $x \overset{\sigma}{\mapsto}_G y$ denotes that there is a transition in G labelled by σ from x to y. These transition definitions are also extended in the usual manner for strings of transitions with labelled with strings in Σ^* instead of single transitions labelled by events.

For an automaton G, in theoretical computer science, the language *accepted* ($L(G)$) by the automaton G is the set of all strings that lead to a final state. $L(G)$ is equivalent to the language *marked* ($\mathcal{L}_m(G)$) in discrete-event system theory. The language *generated* in discrete-event system theory ($\mathcal{L}(G)$) is the set of strings whose state transitions are defined by the transition function $\delta^G(\cdot)$. Note that a script $\mathcal{L}$ is used for discrete-event systems notation and a regular L for computer science notation. When $\delta^G(\cdot)$ is a partial function, $\mathcal{L}(G) \subset \Sigma^*$. $\mathcal{L}(G)$ is a prefix-closed language

in that it contains all the prefixes of all its strings. The languages $\mathcal{L}_m(G)$ and $L(G)$ are not prefix-closed in general. For a language K, $\overline{K}$ denotes the set of the prefixes of the strings in K. An automaton that accepts a prefix-closed language is called a *prefix-closed automaton*. An automaton is said to be *nonblocking* if the prefix-closure of its marked language is equal to its generated language, i.e., $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$.

To review the parallel composition operation, consider G as defined above together with another automaton $H = (X^H, x_o^H, \Sigma^H, \delta^H, X_m^H)$.

The parallel composition of G and H is denoted by $G\|H$:

$G\|H := ((X^G \times X^H), (x_o^G, x_o^H), \Sigma^G \cup \Sigma^H, \delta^{G\|H}, (X_m^G \times X_m^H))$

where

$$\delta^{G\|H}((x^G, x^H), \sigma) = \left\{ \begin{array}{ll} (\delta^G(x^G, \sigma), \delta^G(x^H, \sigma)) & \text{if} \quad \delta^G(x^G, \sigma)! \wedge \delta^H(x^H, \sigma)! \\ (\delta^G(x^G, \sigma), x^H) & \text{if} \quad \delta^G(x^G, \sigma)! \wedge (\sigma \notin \Sigma^H) \\ (x^G, \delta^H(x^H, \sigma)) & \text{if} \quad \delta^H(x^H, \sigma)! \wedge (\sigma \notin \Sigma^G) \\ \text{undefined otherwise} & \end{array} \right\}$$

Note that the unary operator ! for $f(\alpha)!$ returns true if $f(\cdot)$ is defined for input α, false otherwise. It is sometimes assumed without loss of generality that the automata in this thesis have a common event set Σ unless explicitly stated otherwise. Self-loops can always be added at all states for all events not initially in an automaton's event set. The phrases "composed automata" and "intersected automata" denote the same concept - a set of automata interacting through parallel composition.

Given a set of h modules modelled as automata $\{H_1, H_2, \ldots, H_h\}$, the script notation $\mathcal{H}_1^h$ denotes the set of the module automata $\{H_1, H_2, \ldots, H_h\}$ and the regular notation H_1^h is used to denote the parallel composition $H_1\|H_2\|\cdots\|H_h$. The composed automaton H_1^h accepts (generates) a string t if and only if t is accepted

(generated) by all automata in $\mathcal{H}_1^h = \{H_1, H_2 \ldots, H_h\}$. This implies that $\mathcal{L}_m(H_1^h) = \mathcal{L}_m(H_1) \cap \cdots \cap \mathcal{L}_m(H_h)$. Similarly, for a set of k languages $\{K_1, K_2, \ldots, K_k\}$, the script notation $\mathcal{K}_1^k$ denotes the set $\{K_1, K_2, \ldots, K_k\}$ and the regular notation K_1^k to denote the intersection of the languages $K_1 \cap K_2 \cap \cdots \cap K_k$. The notation $\mathcal{L}(\mathcal{H}_1^h)$ and $\mathcal{L}_m(\mathcal{H}_1^h)$ is also used for the sets of languages $\{\mathcal{L}(H_1), \mathcal{L}(H_2), \ldots, \mathcal{L}(H_h)\}$ and $\{\mathcal{L}_m(H_1), \mathcal{L}_m(H_2), \ldots, \mathcal{L}_m(H_h)\}$, respectively.

Given two deterministic automata H_1 and H_2, simple polynomial time methods are shown in [27] for testing if $\mathcal{L}(H_1) \subseteq \mathcal{L}(H_2)$ or $\mathcal{L}(H_1) = \mathcal{L}(H_2)$. Also, with the above definition of the parallel composition operation the automaton $H_1 \| H_2$ can be constructed in polynomial time.

3.3 Supervisory Control Theory

Following the modelling formalisms of [54, 81], systems are modelled as finite-state automata with external controllers. Control actions are enforced by selectively disabling controllable events. Controllers are also realized as finite-state automata that can observe some events and control a potentially different set of events. Controllers should not be able to disable uncontrollable events and control actions should not update on the occurrence of locally unobservable events.

Given a centralized controller S and a system G, the composed system of S controlling G is denoted as the controlled system S/G. Furthermore, because parallel controllers realized as finite-state automata are assumed, S/G is equivalent to $S \| G$. Controller S is said to be nonblocking for system G if S/G is nonblocking, i.e., if $\overline{\mathcal{L}_m(S\|G)} = \mathcal{L}(S\|G)$.

As stated above, a controller may only observe a subset of the system events $\Sigma_o \subseteq \Sigma$. A natural projection operation $P : \Sigma \to \Sigma_o$ is traditionally used to model

a controller's observations of system behavior. For the empty event ϵ, $P(\epsilon) = \epsilon$, and for a string of events s and an event σ, $P(s\sigma) = P(s)\sigma$ if $\sigma \in \Sigma_o$ and $P(s\sigma) = P(s)$ otherwise. The inverse function $P^{-1} : \Sigma_o^* \to 2^{\Sigma^*}$ is defined such that $P^{-1}(s)$ is the set of strings with s as its projection. As system behavior progresses and a string of events s is generated by the system, a controller with natural projection observation function $P(\cdot)$ would observe $P(s)$. The controller would then use observation projection $P(s)$ to estimate the current system state and determine its control action. A controller is said to be *admissible* if it only attempts to disable controllable events and updates its control action on the occurrence of observable events.

Usually controllers are pre-computed in order to satisfy some specification. For example, given a system G and a language specification K, a control automaton S may be computed such that the behavior of the controlled system matches the specification ($\mathcal{L}_m(S/G) = K$), or is included in the specification ($\mathcal{L}_m(S/G) \subseteq K$). If it is desired that $\mathcal{L}_m(S/G) = K$, then K is called a matching specification. If it is desired that $\mathcal{L}_m(S/G) \subseteq K$, then K is called a safety specification. For a precomputed controller, all control actions are predetermined before the controller is used. However, there is a concept called *online control* discussed in Chapter V where the control actions are computed as system behavior progresses and are not known before run-time.

3.3.1 Decentralized Control

For the case of multiple controllers (i.e., standard decentralized control) in the framework of [43] an event is disabled if it is disabled by at least one controller. In this case the actions of local controllers are combined globally by a "fusion by intersection" policy. See Figure 3.1 for a schematic of a system with a set decentralized

controllers.

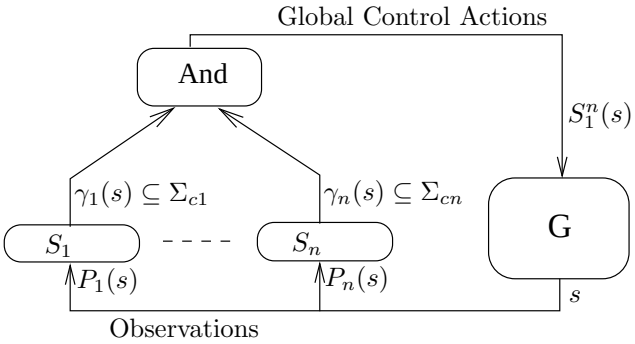

Figure 3.1: A schematic of a conjunctive decentralized control system

In Figure 3.1, the monolithic system G is controlled by n controllers, $\{S_1, \ldots, S_n\}$. The controllers $\{S_1, \ldots, S_n\}$ observe system events $\{\Sigma_{o1}, \ldots, \Sigma_{o1}\}$ and control events $\{\Sigma_{c1}, \ldots, \Sigma_{c1}\}$ respectively. For every controller i there is also a defined local observation projection $P_i : \Sigma \to \Sigma_{oi}$. Decentralized controller S_i makes observations of the system behavior s and generates control actions $\gamma_i(s)$. The local control actions $\{\gamma_1(s), \ldots, \gamma_n(s)\}$ are combined using an intersection operation to form the global control action that is enforced on the system G. Therefore, for this control system, if an event is locally disabled by a controller S_i, then it is globally disabled due to the global intersections of control actions.

For a set of decentralized controllers $\{S_1, \ldots, S_s\}$, a similar notation for $\mathcal{S}_1^s = \{S_1, \ldots, S_s\}$ and $S_1^s = S_1 \| \cdots \| S_s$ is used as seen above for $\mathcal{H}_1^h$ and H_1^h. Hence, a set of controllers $\mathcal{S}_1^s$ controlling G is equivalent to S_1^s/G. For the set of control automata $\{S_1, \ldots, S_s\}$, let Σ_{ci} and Σ_{oi} be the sets of events that are respectively controllable and observable by S_i. In this decentralized control setting let $\Sigma_c = \cup_{i=1}^n \Sigma_{ci}$ and $\Sigma_{uc} = \Sigma \setminus \Sigma_c$.

Three important properties related to decentralized controller existence are con-

trollability, co-observability and $\mathcal{L}_m(G)$-closure.

Definition 1 *[54] Consider the languages K and M such that $M = \overline{M}$ and the set of uncontrollable events Σ_{uc}. The language K is* controllable *with respect to M and Σ_{uc} if $\overline{K}\Sigma_{uc} \cap M \subseteq \overline{K}$.*

Definition 2 *Consider the languages K and M. The language K is M-*closed *if $K = \overline{K} \cap M$.*

Definition 3 *[68] Consider the languages K and M such that $M = \overline{M}$ and the sets of locally controllable, Σ_{ci}, and observable Σ_{oi} events such that $i \in \{1, \ldots, s\}$. The language K is* co-observable *with respect to M, P_i and Σ_{ci}, $i \in \{1, \ldots, s\}$ if for all $t \in \overline{K}$ and for all $\sigma \in \Sigma_c$,*

$$\left(t\sigma \notin \overline{K}\right) \text{ and } (t\sigma \in M) \Rightarrow$$
$$\exists i \in \{1, \ldots, s\} \text{ such that } P_i^{-1}\left[P_i(t)\right]\sigma \cap \overline{K} = \emptyset \text{ and } \sigma \in \Sigma_{ci}.$$

The concept of co-observability captures the notion that one supervisor always knows to disable an event when needed.[1] In the case of centralized control as discussed in [44], co-observability is called observability. The above system properties are central in the following controller existence theorem called the controllability and co-observability theorem.

Theorem 1 *[68] For a system G and a specification H such that $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G)$, sets of controllable events $\{\Sigma_{c1}, \ldots, \Sigma_{cs}\}$ and sets of observable events $\{\Sigma_{o1}, \ldots, \Sigma_{os}\}$ there exists a set of partial observation controllers $\{S_1, \ldots, S_s\}$ such that $\mathcal{L}_m(S_1^s/G) = \mathcal{L}_m(H)$ and $\mathcal{L}(S_1^s/G) = \overline{\mathcal{L}_m(H)}$ if and only if the following three conditions hold:*

[1] In this thesis the terminology of control theory is used even though it may be counter to naming conventions currently used in computer science theory. Namely, co-observability is not non-observability.

1. *$\mathcal{L}_m(H)$ is controllable with respect to $\mathcal{L}(G)$ and Σ_{uc}.*

2. *$\mathcal{L}_m(H)$ is co-observable with respect to $\mathcal{L}(G)$, $\Sigma_{o1},\ldots,\Sigma_{os}$ and $\Sigma_{c1},\ldots,\Sigma_{cs}$.*

3. *$\mathcal{L}_m(H)$ is $\mathcal{L}_m(G)$-closed.*

For languages generated by deterministic automata, controllability and $\mathcal{L}_m(G)$-closure can be decided in polynomial time using standard automata manipulation operations. There is a method presented in [67] for deciding the co-observability of languages generated by deterministic automata. For the sake of the reader, this method is shown in Appendix A for the two-controller case. The essence of this method is that for a system G, a specification automaton H, sets of controllable events Σ_{c1}, Σ_{c2} and sets of observable events Σ_{o1}, Σ_{o2}, an automaton $\mathcal{M}$ is constructed. For this method, $\mathcal{L}_m(\mathcal{M}) = \emptyset$ if and only if $\mathcal{L}_m(H)$ is co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2} and Σ_{c1}, Σ_{c2}.

The construction of $\mathcal{M}$ takes polynomial time if the number of controllers is bounded. Therefore, for deterministic monolithic system problems with a bounded number of controllers, controller existence can be decided in polynomial time.

A less restrictive version of Theorem 1 also holds for the case of generated language specifications (and hence prefix-closed specifications) where the $\mathcal{L}_m(G)$-closure condition is disregarded.

3.4 Computational Complexity

This section presents a brief review of needed concepts from the theory of computation. For a more thorough exposition of these topics, please consult one of the standard texts in the field such as [15, 18, 27].

Problems are said to be in class P if they can be solved in polynomial time

by a deterministic computation device such as a deterministic Turing machine or a deterministic RAM machine. The exact type of computation device does not matter as long as it is a "reasonable" computation device. Similarly, problems are said to be in class NP if they can be solved in polynomial time by a nondeterministic computation device. The class PSPACE includes all problems that can be solved by a deterministic computation device using a polynomial amount of space. Similarly, the class NPSPACE includes all problems that can be solved by a nondeterministic computation device using a polynomial amount of space.

It is known that NP $\subseteq$ PSPACE, but the inclusion is believed to be proper.

The NP vs. PSPACE problem is a major open problem in computer science similar to the P vs. NP problem. EXP is the class of problems that can be solved by a Turing machine in an exponential amount of time. It is known that PSPACE$\neq$EXP and it is believed that the sets PSPACE and EXP are incomparable. This implies that there may be some problems in PSPACE that cannot be solved in an exponential amount of time, but deciding if PSPACE and EXP are incomparable is another major open problem in computer science. However, a well known result from complexity theory is that PSPACE=NPSPACE [69]. This means that a Turing machine bounded using to a polynomial amount of space cannot solve more problems if it operates in a nondeterministic manner.

If a problem C is in PSPACE, it is then known that $C' \in$ PSPACE where C' is the "complement" problem of C. In the discussions contained herein, showing a language to be not co-observable (non-co-observable) is the complement problem of showing that the language is co-observable.

The concept of a "polynomial-time many-one reduction" is used to denote that one problem is "more difficult" than another. For two problems $C \subseteq \Gamma_c^*$ and $D \subseteq \Gamma_d^*$,

there is a polynomial-time many-one reduction from C to D (denoted $C \leq_m^p D$) if there exists a polynomial-time computable function $f : \Gamma_c^* \rightarrow \Gamma_d^*$ such that for each $x \in \Gamma_c^*$, $x \in C$ if and only if $f(x) \in D$ [15]. Intuitively, it can be thought that if a polynomial-time many-one reduction exists, a solution method for problem D can be used to solve problem C.

Problem D is called PSPACE-complete if it is in PSPACE and all problems in PSPACE can be reduced to D. PSPACE-complete problems are problems that are considered to be the "most difficult" of the problems in PSPACE and are at least as hard as all NP-complete problems. Showing a problem to be PSPACE-complete is considered strong evidence that the problem is computationally very difficult and can be solved in polynomial time if and only if P=NP and NP=PSPACE.

Given a set of deterministic finite-state automata $\{A_1, \ldots, A_n\}$ with a common alphabet Σ^A such that for $i \in \{1, \ldots, n\}$, $A_i = (X^{A_i}, x_o^{A_i}, \Sigma^A, \delta^{A_i}, X_m^{A_i})$, Kozen [35] demonstrates that the problem of deciding if $\mathcal{L}_m(A_1 \| \cdots \| A_n) = \emptyset$ is PSPACE-complete. This problem is called the deterministic finite-state automata intersection problem and is referred to as "DFA-Int".

It is well-known (cf. [18]) that the problem of showing the language equivalence of two nondeterministic finite-state automata is PSPACE-complete. With this information it is also easily shown that for two automata A and B, deciding $L(A) \subseteq L(B)$ for the nondeterministic case is also PSPACE-complete because verifying $L(A) \subseteq L(B)$ and $L(B) \subseteq L(A)$ also verifies that $L(A) = L(B)$, a known PSPACE-complete problem. Because of these discouraging results for simple nondeterministic automata comparison problems, deterministic automata are generally used in the rest of this thesis. In the deterministic case $L(A) \subseteq L(B)$ and $L(A) = L(B)$ can be tested in polynomial time [27].

CHAPTER IV

THE COMPUTATIONAL DIFFICULTY OF DISTRIBUTED SYSTEMS PROBLEMS

4.1 Chapter Overview

This chapter investigates computational issues associated with the supervision of distributed discrete-event systems. It is shown that in general many problems related to the supervision of these systems are PSPACE-complete. This shows that although there may be space-efficient methods of avoiding the state-explosion problem inherent to concurrent processes, there are most likely no time-efficient solutions that would aid in the study of "large-scale" systems. Computational difficulty results are shown using a reduction from a special class of automata intersection problem introduced here where accepted behavior is restricted to be prefix-closed. Deciding if there exists a controller for a modular system to achieve a global specification is shown to be PSPACE-complete. Several controller verification problems are shown to be PSPACE-complete, even for prefix-closed cases. Controller admissibility and online control operations are also discussed.

4.2 Complexity of Automata Intersection Problems

Computational complexity results for several important classes of automata intersection problems are now shown. When given two sets of interacting deterministic finite-state automata $\mathcal{A}_1^a$ and $\mathcal{B}_1^b$, deciding $L(A_1^a) \subseteq L(B_1^b)$ or $L(A_1^a) = L(B_1^b)$ is in PSPACE.

Proposition 1 *Given an instance of two sets of interacting deterministic finite-state automata $\mathcal{A}_1^a$ and $\mathcal{B}_1^b$ accepting languages not necessarily prefix-closed, the problem of deciding the following expressions are in PSPACE:*

1. $L(A_1^a) \subseteq L(B_1^b)$
2. $L(A_1^a) = L(B_1^b)$

Proof:

It is shown that the problem of deciding if $L(A_1^a) \not\subseteq L(B_1^b)$ is in NPSPACE. A well known result from complexity theory is that PSPACE=NPSPACE [69]. This means that a deterministic Turing machine bounded to a polynomial amount of space cannot solve more problems if it operates in a nondeterministic manner. Showing a problem to be in NPSPACE is sufficient to show that problem is also in PSPACE.

A nondeterministic algorithm is presented that uses a polynomial amount of space to decide if there is a string s accepted by all $A_1, \ldots, A_a$ that constitute $\mathcal{A}_1^a$ but not by all $B_1, \ldots, B_b$ that constitute $\mathcal{B}_1^b$. Initially, marker tokens are placed on all the start states of $A_1, \ldots, A_a, B_1, \ldots, B_b$ and events are generated nondeterministically from the common alphabet Σ. As the events are generated the markers are moved to hold the places of the current states of $A_1, \ldots, A_a, B_1, \ldots, B_b$ to simulate the occurrence of those events in the automata. If a generated event is ever undefined at the current

state of an automaton, operation halts. If a string of events is generated such that the current states in $A_1, \ldots, A_a$ are marked and a current state in one of the automata in $B_1, \ldots, B_b$ in not marked, it is consequently known that $L(A_1^a) \not\subseteq L(B_1^b)$.

Keeping track of the current automata states and updating them as nondeterministic events are generated requires less space than the encodings of $\mathcal{A}_1^a$ and $\mathcal{B}_1^b$ so this operation takes a polynomial amount of space with respect to the problem encoding. Therefore, deciding $L(A_1^a) \not\subseteq L(B_1^b)$ is in NPSPACE. Therefore it is known that deciding $L(A_1^a) \subseteq L(B_1^b)$ is in coNPSPACE. Because PSPACE=coNPSPACE [15] deciding if $L(A_1^a) \subseteq L(B_1^b)$ is also in PSPACE.

By a similar construction, it can easily be shown that deciding $L(A_1^a) = L(B_1^b)$ is also in PSPACE. ■

The DFA-Int problem discussed in [35] is similar to the problem discussed in Proposition 1. Kozen's proof for DFA-Int depends on a reduction from the LBA acceptance problem which is given a linear bounded automaton(LBA) Ψ and a string x, to decide if Ψ accepts x. The LBA acceptance problem is a well-known PSPACE-complete problem [27].

A linear bounded automaton Ψ is a special type of possibly nondeterministic Turing machine with a bounded tape defined as follows:

$\Psi = (Q, \Sigma, \Gamma, \delta, q_o, B, F, p)$

where

Q is the finite set of symbols to represent the set of states of Ψ,

Σ is the finite set of input symbols,

Γ is the finite set of tape symbols,

$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ is the next move function,

q_o is the start state symbol,

$ is a blank symbol,

F is the set of accepting state symbols,

p is a polynomial function.

A linear bounded automaton operates in a manner similar to a regular Turing machine except that the single input and work tape is constrained to use at most $p(n)$ cells where $n = |x|$ is the length of the input x.

Kozen reduces the LBA acceptance problem to the DFA-Int problem by generating a set of automata $\mathcal{B}_{\Psi,x}$ that simulate the set of computations performed by Ψ for a given input x. The string x is accepted by Ψ if and only if the automaton $B_{\Psi,x}$ equivalent to the parallel composition of the automata in $\mathcal{B}_{\Psi,x}$ accepts a non-empty language.

Many topics related to modular plants and specifications in discrete-event systems deal with the prefix-closure of languages so it would be advantageous for us to investigate whether automata intersection problems are PSPACE-complete for the special case of automata accepting prefix-closed languages. Therefore, this chapter expands the work in [35] by exploring parallel composition properties of automata generating prefix-closed languages.

Theorem 2 *Given a finite-state automaton A and a set of interacting finite-state automata $B_1, \ldots, B_b$ all accepting prefix-closed languages, the problem of deciding if $L(B_1^b) = L(A)$ is PSPACE-complete.*

Proof: This proof is a modified version of the proof in [35] that shows the DFA-Int problem is PSPACE-complete. Because neither are the proof of DFA-Int nor the present modifications trivial, Kozen's work is replicated and altered as necessary.

Using the definition of Ψ seen above, the linear bounded automaton acceptance problem for an instance of an LBA Ψ and a string x is reduced in polynomial time to the prefix-closed case of deciding $L(B_1^b) \neq L(A)$ for a given set of automata $\{B_1, \ldots, B_b\}$ and an automaton A. The comparison problems in Proposition 1 are more general than the problem in this theorem so it is known that deciding $L(B_1^b) = L(A)$ for the prefix-closed case is in PSPACE. It is therefore sufficient to demonstrate that the deterministic LBA acceptance problem can be reduced in polynomial-time using a many-one mapping to the problem of deciding $L(B_1^b) \neq L(A)$ for the prefix-closed case.

Assume without loss of generality that Ψ has a unique accepting state q_f and that Ψ erases its work tape and moves its read/write head all the way to the left of the tape before accepting. Also assume that Ψ takes an even number of steps before accepting. These assumptions can be made without loss of generality because the size of the Turing machine finite control will at most double with these modifications.

The instantaneous description (ID) of a Turing machine represents as a finite string the current state of the Turing machine, the current contents of the work tape and the location of the read/write head. Suppose $y = y_1 y_2$ where y is the contents of a Turing machine tape and y_1 represents the content of the tape to the left of the read/write head, and let the rest of y_2 be the content of the tape to the right of the read/write head. Let the first letter of y_2 represent the tape cell being read by the read/write head. If q represents the current state, an effective representation of the ID would be the string $y_1 q y_2$.

In this proof, the ID representation is padded with a string of normally unwritten blank symbols $\$^{p(n)-(|y_1|+|y_2|)}$ to make explicit in the representation of the ID the fact that the LBA has a work tape of size $p(n)$ where n is the length of the input string.

Therefore with $y = y_1 y_2$, an ID of the LBA would be $y_1 q y_2 \$^{p(n)-(|y_1|+|y_2|)}$.

If x is the input string to Ψ, the initial instantaneous description (ID_o) would be $q_o x \$^{p(n)-|x|}$. Because of the previous assumptions on how Ψ accepts a string, there is a unique accepting instantaneous description $ID_f = q_f \$^{p(n)}$. The notation $ID_j \vdash_\Psi ID_i$ is used to represent that according to the transition rules of Ψ, instantaneous description ID_i follows in one step from instantaneous description ID_j. It should therefore be readily apparent that $[x \in L(\Psi)]$ if an only if $[\exists(ID_o, ID_1, \ldots, ID_f)$ such that $\forall i \in [1, \ldots, f](ID_{i-1} \vdash_\Psi ID_i)]$. This means that a string x is accepted by Ψ if and only if there is a sequence of instantaneous descriptions $ID_o \vdash_\Psi ID_1 \vdash_\Psi \cdots \vdash_\Psi ID_f$ starting with the initial instantaneous description ID_o and finishing with the accepting instantaneous description ID_f.

Let $\Delta = \Gamma \cup Q \cup \{\$\}$ and let $\#$ be a previously unused symbol. To perform this reduction from an instance of the LBA acceptance problem to an instance of the prefix-closed automata intersection problem, a set of prefix-closed interacting automata is constructed $\mathcal{B}_{\Psi,x}$ such that the automaton $B_{\Psi,x}$ equivalent to the automaton formed by the parallel composition of the automata in $\mathcal{B}_{\Psi,x}$ accepts the language

$$
\begin{aligned}
&(((\Delta \cup \{\#\})^*) \setminus ((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*)) \\
&\qquad \bigcup \{\#ID_o \# ID_1 \# \cdots \# ID_f \#\#\} \subset (\Delta \cup \{\#\})^* \\
&\qquad\qquad\qquad\qquad \text{if } [\forall i \in [1, \ldots, f](ID_{i-1} \vdash_\Psi ID_i)], \\
&(((\Delta \cup \{\#\})^*) \setminus ((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*)) \\
&\qquad\qquad\qquad\qquad \text{otherwise.}
\end{aligned}
$$

$B_{\Psi,x}$ always accepts all strings not containing $\#\#$ and $B_{\Psi,x}$ accepts a string ending with $\#\#$ if and only if there is a sequence of instantaneous descriptions from ID_o to ID_f that represent a set of valid computations for Ψ.

An automaton $A_{\Psi,x}$ can be constructed in polynomial time with respect to the encoding of Ψ and x such that

$L(A_{\Psi,x}) = (((\Delta \cup \{\#\})^*) \setminus ((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*))$.

Note that $L(B_{\Psi,x})$ and $L(A_{\Psi,x})$ are both prefix-closed by construction. For this reduction $L(B_{\Psi,x}) \neq L(A_{\Psi,x})$ if and only if $x \in L(\Psi)$ where $\mathcal{B}_{\Psi,x}$ and A are constructed in polynomial time from x and Ψ. Therefore deciding $L(B_{\Psi,x}) = L(A_{\Psi,x})$ is PSPACE-complete because PSPACE = coPSPACE.

When the read/write head moves, the state updates, a symbol is written on the current tape cell and the tape head should move exactly one cell to the left or the right. Therefore, to verify that $(ID_i \vdash_\Psi ID_j)$, it needs to be verified that the tape contents in ID_i and ID_j are identical except for where the read/write head wrote to the tape during the transition and that the read/write head moved exactly one tape square to the left or right according to the next move function δ. With this in mind, given a three element string $\alpha_1\alpha_2\alpha_3$ from ID_i and a three element string $\beta_1\beta_2\beta_3$ from ID_j both at the same relative locations in the instantaneous descriptions, it can be verified in polynomial time that $\beta_1\beta_2\beta_3$ can follow from $\alpha_1\alpha_2\alpha_3$. If this is verified for all pairs of three element strings at the same locations in ID_i and ID_j, it can be verified in polynomial time that $(ID_i \vdash_\Psi ID_j)$.

Two sets of interacting automata, $\mathcal{B}^{even}$ and $\mathcal{B}^{odd}$ are now constructed. The automata in $\mathcal{B}^{even}$ verify for a sequence of instantaneous descriptions $ID_i, \ldots, ID_j$ that the instantaneous descriptions with odd indices follow from the instantaneous descriptions with even indices. Similarly, the automata in $\mathcal{B}^{odd}$ verify for a sequence of instantaneous descriptions $ID_i, \ldots, ID_j$ that the even instantaneous descriptions follow from the odd instantaneous descriptions.

With this in mind, B_i^{even} is constructed to accept the language

$$\left(\{\#\Delta^{i-1}\alpha_1\alpha_2\alpha_3\Delta^{p(n)-i-2}\#\Delta^{i-1}\beta_1\beta_2\beta_3\Delta^{p(n)-i-2}\}^*\{\#\#\}\right)$$
$$\bigcup((\Delta\cup\{\#\})^*\setminus((\Delta\cup\{\#\})^*\{\#\#\}(\Delta\cup\{\#\})^*))$$

where $\alpha_1\alpha_2\alpha_3, \beta_1\beta_2\beta_3 \in \Delta^3$. A string (i.e., $\#ID_0\#ID_1\#\cdots\#ID_f\#\#$) containing $\#\#$ is accepted by B_i^{even} only if the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the odd instantaneous descriptions follow from the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the even instantaneous descriptions.

Similarly, let us also construct B_i^{odd} to accept the language

$$(\{\#\Delta^{p(n)+1}\}\{\#\Delta^{i-1}\eta_1\eta_2\eta_3\Delta^{p(n)-i-2}\#\Delta^{i-1}\theta_1\theta_2\theta_3\Delta^{p(n)-i-2}\}^*\{\#\Delta^{p(n)+1}\#\#\})$$
$$\bigcup((\Delta\cup\{\#\})^*\setminus((\Delta\cup\{\#\})^*\{\#\#\}(\Delta\cup\{\#\})^*))$$

where $\eta_1\eta_2\eta_3, \theta_1\theta_2\theta_3 \in \Delta^3$. A string containing $\#\#$ (i.e., $\#ID_0\#ID_1\#\cdots\#ID_f\#\#$) is accepted by B_i^{odd} only if the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the even instantaneous descriptions follow from the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the odd instantaneous descriptions. Remember that it is assumed without loss of generality that f is odd.

Let us construct B_i^{even}'s and B_i^{odd}'s for i ranging from 0 to $(p(n)-1)$. This should take less than $6|\Delta|^3p(n)$ states each, so this construction can be performed in polynomial time with respect to the encodings of Ψ and x. Note that by their constructions, the languages accepted by the B_i^{even}'s and B_i^{odd}'s are prefix-closed.

Define $\mathcal{B}^{even} = \{B_0^{even}, \ldots, B_{p(n)-1}^{even}\}$ and $\mathcal{B}^{odd} = \{B_0^{odd}, \ldots, B_{p(n)-1}^{odd}\}$. these composed automata are respectively equivalent to the automata $(B_0^{even}\|\cdots\|B_{p(n)-1}^{even})$ and $(B_0^{odd}\|\cdots\|B_{p(n)-1}^{odd})$. B^{even} accepts a string (such as $\#ID_i\#ID_{i+1}\#\cdots\#ID_j\#\#$) containing $\#\#$ only if the odd instantaneous descriptions follow from the even instantaneous descriptions. Likewise, B^{odd} accepts a string containing $\#\#$ (notably $\#ID_i\#ID_{i+1}\#\cdots\#ID_j\#\#$) only if the even instantaneous descriptions follow from

the odd instantaneous descriptions.

A final automaton B^{final} is constructed that accepts the prefix closure of the following set of strings:

$$((\Delta \cup \{\#\})^* \setminus ((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*)) \bigcup \left(\{\#ID_o\}\{\#\Delta^{p(n)+1}\}^*\{\#ID_f\#\#\}\right)$$

B^{final} accepts a string of instantaneous descriptions ending with ## only if the first instantaneous description is ID_o and the final instantaneous description is ID_f. Note that B^{final} also accepts a prefix-closed language. Constructing B^{final} takes less than $6|\Delta|^3 p(n)$ states, so this construction can be performed in polynomial time.

Let $\mathcal{B}_{\Psi,x} = \{B^{final}\} \cup \mathcal{B}^{even} \cup \mathcal{B}^{odd}$. If there is a valid accepting computation for Ψ with input x then $ID_0, ID_1, \ldots, ID_f$ is a sequence of valid accepting computations for Ψ and

$\#ID_0\#ID_1\#\cdots\#ID_f\#\#$ is accepted by $B_{\Psi,x}$. Likewise, if a string containing ## is accepted by $B_{\Psi,x}$, it must be the string $\#ID_0\#ID_1\#\cdots\#ID_f\#\#$ representing a valid computation on Ψ for accepting input x. A string containing ## is accepted by all the automata in $\mathcal{B}_{\ominus,\S}$ if and only if there is a valid computation for Ψ that accepts x. It is therefore known $[x \in L(\Psi)] \iff [L(B_{\Psi,x}) \neq L(A_{\Psi,x})]$. This completes the many-one mapping.

$A_{\Psi,x}$ and the components of $\mathcal{B}_{\Psi,x}$ can be constructed in polynomial time with respect to the size of the encoding of x and Ψ. Therefore, the problem of deciding $L(\mathcal{B}_1^b) = L(A)$ for the prefix-closed case is PSPACE-complete. ■

The primary alterations in the proof of Theorem 2 from the proof of DFA-Int is that the automata in the Theorem 2 proof accept all prefix-closed strings not containing the substring ## and they accept a string containing ## if and only if their parts of the LBA computation are valid. Therefore a string containing ## is

accepted by the construction in Theorem 2 if and only if a string x is accepted by Ψ.

The result of Theorem 2 is particularly discouraging. PSPACE-complete problems are thought to be rather difficult. However, it is well known that the problems of deciding if $L(A) \subseteq L(B)$ and $L(A) = L(B)$ are in P for monolithic automata. This observation prompts the following proposition.

Proposition 2 *Given an instance of a deterministic finite-state automaton A not necessarily accepting a prefix-closed language and a finite set of interacting deterministic finite-state automata $\mathcal{B}_1^b = \{B_1, \ldots, B_b\}$ also not necessarily accepting prefix-closed languages, the problem of deciding if $L(A) \subseteq L(B_1^b)$ is in P.*

Proof: This proposition is demonstrated by presenting a polynomial time procedure to solve this problem.

Because the automata all have common alphabets

$[L(A) \subseteq L(B_1^b)] \iff [\forall\ i \in \{1, \ldots, b\}\ [L(A) \subseteq L(B_i)]]$

$L(A) \subseteq L(B_i)$ can be verified in polynomial time with respect to the length of the encoding of A and B_i, so $[\forall\ i \in \{1, \ldots, b\}, [L(A) \subseteq L(B_i)]]$ can be verified in polynomial time with respect to the length of the encoding of A and $\mathcal{B}_1^b$. Therefore the problem of deciding $L(A) \subseteq L(B_1^b)$ is in class P. ■

However, even with the positive results of Proposition 2, the converse problem is very difficult. For the prefix closed case, given a finite set of interacting deterministic finite-state automata $\mathcal{B}_1^b = \{B_1, \ldots, B_b\}$ deciding if $L(B_1^b) \subseteq L(A)$ is PSPACE-complete.

Theorem 3 *Given a finite-state automaton A and a set of interacting finite-state automata $\mathcal{B}_1^b = \{B_1, \ldots, B_b\}$ all generating prefix-closed languages, the problem of deciding if $L(B_1^b) \subseteq L(A)$ is PSPACE-complete.*

Proof: This problem is in PSPACE due to Proposition 1. The construction used in this proof is identical to the proof used in Theorem 2 above. This construction is not repeated for the sake of brevity. It can then be shown using $\mathcal{B}_{\Psi,x}$ and $A_{\Psi,x}$ from the construction in the proof of Theorem 2 that:

$[x \notin L(\Psi)] \iff [L(B_{\Psi,x}) \not\subseteq L(A_{\Psi,x})]$

Therefore, the problem of deciding $L(B_1^b) \not\subseteq L(A_{\Psi,x})$ for the prefix-closed case given $\{B_1, \ldots, B_b\}$ and A is PSPACE-complete. Therefore, the problem of deciding $L(B_1^b) \subseteq L(A_{\Psi,x})$ for the prefix-closed case is PSPACE-complete. ■

These complexity results can now be extended to the non-prefix-closed cases of the problems discussed above.

Corollary 1 *The problems of deciding if* $L(B_1^b) = L(A)$ *and* $L(B_1^b) \subseteq L(A)$ *given* $\mathcal{B}_1^b$ *and* A *for the non-prefix-closed case is PSPACE-complete.*

Proof: It is easy to see that these problems are a special case of the problem in Proposition 1 above, so these problems are in PSPACE. It should also be apparent that these problems are more general than the decision problems in Theorem 2 and Theorem 3 above so these problems are in PSPACE and are PSPACE-hard; consequently these problems are PSPACE-complete. ■

After Corollary 1 it should be readily apparent that the problems discussed in Proposition 1 are PSPACE-complete.

4.3 Control of Discrete-Event Systems

The complexity results of Section 4.2 are now used to show complexity results for several important supervisory control problems. First, the supervisory control

theory concepts of controllability, M-closure, and co-observability in Section 3.3 are extended to handle the cases where the systems and specifications are distributed.

Let $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$ be sets of languages. Let Σ_{ci} and Σ_{oi} be the locally controllable and observable event sets respectively for $i \in \{1, \ldots, s\}$. Let $P_i : \Sigma^* \rightarrow \Sigma_{oi}^*$ be the natural projection that erases events in $\Sigma \setminus \Sigma_{oi}$. Furthermore, let $\Sigma_c = \cup_{i=1}^{s} \Sigma_{ci}$ and $\Sigma_{uc} = \Sigma \setminus \Sigma_c$.

Definition 4 *Consider the sets of languages $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$ such that $M_1 = \overline{M_1}, M_2 = \overline{M_2}, \ldots, M_m = \overline{M_m}$ and the set of uncontrollable events Σ_{uc}. The set of languages $\mathcal{K}_1^k$ is* modular controllable *with respect to $\mathcal{M}_1^m$ and Σ_{uc} if $\overline{K_1^k}\Sigma_{uc} \cap M_1^m \subseteq \overline{K_1^k}$.*

Definition 5 *Consider the sets of languages $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$. The set of languages $\mathcal{K}_1^k$ is* modular $\mathcal{M}_1^m$-closed *if $K_1^k = \overline{K_1^k} \cap M_1^m$.*

Definition 6 *Consider the sets of languages $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$ such that $M_1 = \overline{M_1}, M_2 = \overline{M_2}, \ldots, M_m = \overline{M_m}$ and the sets of locally controllable, Σ_{ci}, and observable Σ_{oi} events such that $i \in \{1, \ldots, s\}$. The set of languages $\mathcal{K}_1^k$ is* modular co-observable *with respect to $\mathcal{M}_1^m$, P_i and Σ_{ci}, $i \in \{1, \ldots, s\}$ if for all $t \in \overline{K_1^k}$ and for all $\sigma \in \Sigma_c$,*

$$\left(t\sigma \notin \overline{K_1^k}\right) \text{ and } (t\sigma \in M_1^m) \Rightarrow$$
$$\exists i \in \{1, \ldots, s\} \text{ such that } P_i^{-1}\left[P_i(t)\right]\sigma \cap \overline{K_1^k} = \emptyset \text{ and } \sigma \in \Sigma_{ci}.$$

The following theorem for the existence of controllers for modular systems can now be demonstrated using the extended modular definitions of controllability, co-observability and language closure.

Theorem 4 *For a given set of finite-state automata system modules $\mathcal{G}_1^g$ and a set of finite-state automata specification modules $\mathcal{H}_1^h$ such that H_1^h is nonblocking, there exists a set of partial observation controllers $\{S_1, S_2, \ldots, S_s\}$ such that*

$\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ *and* $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$

if and only if the following three conditions hold:

1. $\mathcal{L}_m(\mathcal{H}_1^h)$ *is modular controllable with respect to* $\mathcal{L}(\mathcal{G}_1^g)$ *and* Σ_{uc}.

2. $\mathcal{L}_m(\mathcal{H}_1^h)$ *is modular co-observable with respect to* $\mathcal{L}(\mathcal{G}_1^g)$, $P_1, \ldots, P_s$ *and* $\Sigma_{c1}, \ldots, \Sigma_{cs}$.

3. $\mathcal{L}_m(\mathcal{H}_1^h)$ *is modular* $\mathcal{L}_m(\mathcal{G}_1^g)$*-closed.*

The proof of this theorem is constructive and is a generalization of the proof of the Controllability and Co-Observability Theorem discussed in [10]; it depends on a sample-path argument not shown here. This result says that a set of nonblocking controllers $S_1, S_2, \ldots, S_s$ that achieves a set of modular specifications $\mathcal{H}_1^h$ for a modular system $\mathcal{G}_1^g$ (i.e. $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ and $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$) exists if and only if the system is modular controllable, modular co-observable and modular $\mathcal{L}_m(\mathcal{G}_1^g)$-closed. These properties completely characterize necessary and sufficient existence conditions for controllers of modular systems. In turn, these properties can play a role in safe controller synthesis when existence conditions are not satisfied for a controlled system to match a specification. Safe controller synthesis for monolithic systems is discussed in [10].

Given $\mathcal{H}_1^h$, deciding if H_1^h is nonblocking is a PSPACE-complete problem. This can be shown using a simple reduction from the automata intersection problem presented in [35]. However, there may be enough foreknowledge to decide this property holds in a computationally feasible manner. It is assumed in this chapter that the modular specifications are given such that H_1^h is nonblocking. If the specification is blocking, no nonblocking controllers that achieves the specification can exist.

Similarly, the astute reader will note that $\mathcal{L}_m(H_1^h) \subseteq \mathcal{L}_m(G_1^g)$ is a necessary condition for both $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ and $\mathcal{L}_m(\mathcal{H}_1^h)$ to be modular $\mathcal{L}_m(\mathcal{G}_1^g)$-closed. If $\mathcal{L}_m(H_1^h) \not\subseteq \mathcal{L}_m(G_1^g)$, $\mathcal{H}_1^h$ can be replaced with $\mathcal{H}_1^h \cup \mathcal{G}_1^g$ so that the specification behavior is strictly smaller than the specification behavior. $H_1 \| \cdots \| H_h \| G_1 \| \cdots \| G_g$ is the automaton equivalent of the new specification behavior. This substitution will not alter the computational complexity of the problems discussed later in this chapter.

It is also easy to show a more general theorem concerned only with prefix-closed behavior when blocking is not a concern.

Theorem 5 *For a given set of prefix-closed finite-state automata system modules $\mathcal{G}_1^g$ and a set of prefix-closed finite-state automata specification modules $\mathcal{H}_1^h$ such that $\mathcal{L}(H_1^h) \neq \emptyset$, there exists a set of partial observation controllers $\{S_1, S_2, \ldots, S_s\}$ such that $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$ if and only if the following three conditions hold:*

1. *$\mathcal{L}(\mathcal{H}_1^h)$ is modular controllable with respect to $\mathcal{L}(\mathcal{G}_1^g)$ and Σ_{uc}.*
2. *$\mathcal{L}(\mathcal{H}_1^h)$ is modular co-observable with respect to $\mathcal{L}(\mathcal{G}_1^g)$, $P_1,\ldots,P_s$ and $\Sigma_{c1},\ldots,\Sigma_{cs}$.*
3. *$\mathcal{L}(H_1^h) \subseteq \mathcal{L}(G_1^g)$*

As with Theorem 4, if $\mathcal{L}_m(H_1^h) \not\subseteq \mathcal{L}_m(G_1^g)$, then $\mathcal{H}_1^h$ can be replaced with $\mathcal{H}_1^h \cup \mathcal{G}_1^g$. This substitution will not alter the computational complexity of the problems discussed later in this chapter related to this theorem.

The following proposition can now be shown:

Proposition 3 *Deciding modular controllability, modular co-observability and modular $\mathcal{M}_1^m$-closure for sets of languages specified by sets of finite-state automata is in PSPACE.*

Proof: Proving this proposition relies on a "token" argument similar to that employed by Kozen in [35] for proving automata intersection emptiness is in PSPACE. Given a set of events Σ_{uc} and two sets of automata $\mathcal{H}_1^h$ and $\mathcal{G}_1^g$, it is shown that the problem of deciding modular controllability of $\mathcal{L}_m(\mathcal{H}_1^h)$ with respect to $\mathcal{L}(\mathcal{G}_1^g)$ and Σ_{uc} is in PSPACE. Similar proofs exist to show deciding modular co-observability and modular $\mathcal{M}_1^m$-closure are in PSPACE but are not shown here due to space considerations. Regarding controllability, it is sufficient to show the converse problem of deciding non-controllability is in NPSPACE.

A nondeterministic string of events t is generated one event at a time and used to model the state transitions in the finite-state automata in $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$ starting from their respective start-states. The current states of the automata in $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$ need to be saved and updated as new events are generated. As each new event is generated and added to t, it is tested if $\exists \sigma \in \Sigma_{uc}$ such that

$$[\forall j \in \{1, \ldots, g\} | (t\sigma \in \mathcal{L}(G_j))] \wedge [\forall i \in \{1, \ldots, h\}](t \in \mathcal{L}(H_i))|]$$
$$\wedge [\exists l \in \{1, \ldots, h\} | (t\sigma \notin \mathcal{L}_m(H_i))].$$

If this property ever holds then modular controllability does not hold. All of these operations take a polynomial amount of memory with respect to the encodings of $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$. Because this problem is in NPSPACE, it is also in PSPACE [69]. ■

It should be noted that if the number of controllers, plants and specifications are bounded to be less than some constant k, then modular controllability, modular co-observability and modular $\mathcal{M}$-closure can then be decided in polynomial time if the modular systems and specifications are given as deterministic finite-state automata.

4.4 Existence Problems for the Control of Modular Systems

In this section the computational complexity of deciding controller existence for modular systems under various assumptions is explored. It is assumed that the specifications are nonblocking and that the uncontrolled systems allow at least as much behavior as the specifications (i.e., for the system automata $\mathcal{G}_1^g$ and the specification automata $\mathcal{H}_1^h$, $\mathcal{L}_m(H_1^h) \subseteq \mathcal{L}_m(G_1^g)$ and $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(G_1^g)$ respectively). This section shows one of the main contributions of this chapter: a large class of controller existence problems for modular discrete-event systems are PSPACE-complete. From Theorem 4 and Proposition 3 demonstrated above, it is easy to see that deciding if a decentralized controller exists for a modular specification and modular system is in PSPACE.

Corollary 2 *Given a set of finite-state automata system modules $\mathcal{G}_1^g$, a set of finite-state automata specification modules $\mathcal{H}_1^h$, sets of observable events $\Sigma_{o1}, \ldots, \Sigma_{os}$ and sets of controllable events $\Sigma_{c1}, \ldots, \Sigma_{cs}$, the problem of deciding if there is a set of decentralized controllers $\mathcal{S}_1^s$ such that $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ and $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$ is in PSPACE.*

Similar problems for prefix-closure specifications as seen in Theorem 5 and centralized controllers as seen in [10] can also be decided in PSPACE. A relatively simple problem is investigated: given a modular system and monolithic specification marking prefix-closed languages, is there a single full-observation controller such that the system satisfies the specification? It is shown that even for this restricted subclass of problems are PSPACE-complete.

Theorem 6 *The problem of deciding if there is a full-observation controller S with controllable event set Σ_c for a set of prefix-closed automata system modules $\mathcal{G}_1^g$ and*

a prefix-closed specification automaton H such that $\mathcal{L}(S/G_1^g) = \mathcal{L}(H)$ is PSPACE-complete even if it is known that $\mathcal{L}(H) \subseteq \mathcal{L}(G_1^g)$.

Proof: It is shown in Corollary 2 that this problem is in PSPACE. Here the problem of deciding whether $L(B_1^b) = L(A)$ is reduced to this problem where A and $\mathcal{B}_1^b$ are given prefix-closed finite-state automata. Let $\Sigma_c = \emptyset$. If there exists a controller S such that $\mathcal{L}(S/B_1^b) = \mathcal{L}(A)$ then $L(B_1^b) = L(A)$. If there does not exist a controller such that $\mathcal{L}(S/B_1^b) = \mathcal{L}(A)$ then $L(B_1^b) \neq L(A)$ because no event can be disabled. Since deciding if $L(B_1^b) = L(A)$ is PSPACE-complete even if it is known that $L(A) \subseteq L(B_1^b)$ from Theorem 2, the controller existence problem is also PSPACE-complete.

■

These results are particularly disappointing because they show that a relatively large and simple class of controller existence problems involving modular system automata is PSPACE-complete. Due to Theorem 6 it should also be apparent that deciding modular controllability for languages specified by finite-state automata is PSPACE-complete because modular observability and modular $\mathcal{L}_m(\mathcal{G}_1^g)$-closure are implied by full observation and prefix-closure, respectively.

For the case of full control (namely, $\Sigma_c = \Sigma$) and partial observation, a similar proof method can be used to show that controller existence problem for a modular system and a monolithic specifications is likewise PSPACE-complete. This implies that deciding both modular observability and modular co-observability for languages specified by deterministic finite-state automata is also PSPACE-complete.

If it is not known that $\mathcal{L}(H_1^h) \neq \emptyset$ or that $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(G_1^g)$, controller existence problems remain PSPACE-complete. Likewise, a large class of nonblocking controller existence problems for modular systems specified by finite-state automata are also PSPACE-complete due to Theorem 4 because the nonblocking controller problems

are known to be at least as difficult as prefix-closed specification problems.

An interesting related controller existence problem is also discussed in [66] where it is shown that the problem of deciding co-observability as introduced in [68] is PSPACE-complete if the number of controllers is unbounded even if deterministic monolithic systems and specifications are used. Due to the controllability and co-observability theorem [68], this result implies that deciding the decentralized controller existence problem is likewise PSPACE-complete.

4.5 Admissible Controllers

As stated before, a controller is *admissible* if it updates control actions on locally observable events and disables only locally controllable events. Note that a controller's admissibility when operating on a system is not related to that system's specification. For a monolithic discrete-event system, deciding if a controller automaton S is admissible for a given system automaton G can be decided in polynomial time. This is because testing admissibility of a parallel controller S is equivalent to testing that all transitions occur on observable events and that $\mathcal{L}(S)$ is controllable with respect to $\mathcal{L}(G)$ and Σ_{uc} [10]. This method of testing admissibility also holds for modular systems. To test if a parallel controller S is admissible for a modular system $\mathcal{G}_1^g$ it is first tested if state transitions occur only on the occurrence of observable events and then it is necessary and sufficient to verify that $\mathcal{L}(S)$ is modular controllable with respect to $\mathcal{L}(\mathcal{G}_1^g)$ and Σ_{uc}. Testing that all state transitions in the controller occur on observable events takes polynomial time, but, as was stated earlier in the chapter, testing modular controllability of a monolithic specification with respect to a modular finite-state automata system is PSPACE-complete. This prompts the following theorem whose proof was outlined above.

Theorem 7 *Verifying the admissibility of a single controller with respect to a modular finite-state automata system is also PSPACE-complete.*

When testing the admissibility of a decentralized control system $\{S_1, \ldots, S_s\}$, with respect to a modular system $\{G_1, \ldots, G_g\}$ it needs to be verified first that all state transitions in $\{S_1, \ldots, S_s\}$ occur on locally observable events. It then needs to be verified that each local controller disables only locally controllable events when the other controllers are in operation. More formally, $\forall i \in \{1, \ldots, s\}$, $\mathcal{L}(S_i)$ is modular controllable with respect to $\mathcal{L}(\mathcal{S}_1^s \cup \mathcal{G}_1^g \setminus S_i)$ and Σ_{uci}.

Besides having state transitions only on locally observable events, all local controllers S_i need to be controllable with respect the system it is modifying, i.e., $\mathcal{S}_1^s \cup \mathcal{G}_1^g \setminus S_i$. The following corollary should now be evident:

Corollary 3 *The problem of testing the admissibility of decentralized control systems specified by finite-state automata is PSPACE-complete.*

4.6 Complexity of Modular Controller Verification Problems

The computational complexity of the language matching verification problem for distributed systems is now discussed. For this problem, if given a distributed system controlled by an admissible controller it should be found if the behavior of this controlled system matches the behavior of a specification. This problem is investigated using reductions from automata intersection problems. Simple extensions of the results from Proposition 1 are first shown.

Proposition 4 *Given sets of finite-state automata $\mathcal{S}_1^s$, $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$, verifying if $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$, $\mathcal{L}_m(S_1^s/G_1^g) \subseteq \mathcal{L}_m(H_1^h)$ and $\mathcal{L}_m(H_1^h) \subseteq \mathcal{L}_m(S_1^s/G_1^g)$ are all problems in PSPACE.*

Proof: This proof is similar to the proof of Proposition 1 above. Using the previously discussed definitions regarding controller operation for the supervisory control set-up, the proof of this proposition should be apparent and is not included for the sake of brevity. ■

Using the results of Section 4.2 and Proposition 4 the computational difficulty of verifying a large class properties of distributed supervisory control systems can be found.

Theorem 8 *Given controller automata S and $\mathcal{S}_1^s$, uncontrolled system automata G and $\mathcal{G}_1^g$ and specification automata H and $\mathcal{H}_1^h$. Deciding the validity of each of the following expressions is PSPACE-complete:*

1. $\mathcal{L}(S_1^s/G) = \mathcal{L}(H)$
2. $\mathcal{L}(S/G_1^g) = \mathcal{L}(H)$
3. $\mathcal{L}(S/G) = \mathcal{L}(H_1^h)$
4. $\mathcal{L}(S_1^s/G) \subseteq \mathcal{L}(H)$
5. $\mathcal{L}(S/G_1^g) \subseteq \mathcal{L}(H)$
6. $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(S/G)$

Proof: The listed problems in this theorem are all special cases of the problems in Proposition 4. Therefore these problems are in PSPACE.

The problem in Theorem 2 can be reduced to Problems 1, 2 and 3 in this theorem. This reduction is not show for the sake of brevity, but should be readily apparent. Therefore, Problems 1, 2 and 3 are PSPACE-complete.

The problem in Theorem 3 can be reduced to Problems 4, 5 and 6 in this theorem. This reduction is also not show for the sake of brevity, but should be readily apparent. Therefore, Problems 4, 5 and 6 are like-wise PSPACE-complete. ■

It can be easily seen that the problems listed in Theorem 8 above are special cases of several other problems in PSPACE, notably problems where the marking properties of controlled modular discrete-event systems are verified. These problems are too numerous to conveniently list, but their computational complexity can easily be found as a consequence of Proposition 4 and Theorem 8. Although the listed completeness results deal only with problems where either the controller, plant or specification are modular, these results can be easily extended to cases where two or three of the controller, plant and specification are modular. It should be noted that if the number of controllers, plants and specifications is bounded to be less than some constant k, then all of the verification problems listed here can be decided in polynomial time.

Despite the seemingly overwhelming number of PSPACE-complete verification problems, there are several important verification problems that can be decided in polynomial time even when there is no restriction on the number of modules. It has already been seen in Proposition 2 that given the finite-state automata $\mathcal{B}_1^b$ and A, verifying $L(A) \subseteq L(B_1^b)$ is in P. This result can be used to prove the following propositions.

Proposition 5 *Given a controller automaton S, system automaton G and a set of specification automata $\mathcal{H}_1^h$, the problem of verifying $\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_1^h)$ is in P.*

Proof: Because it is assumed without loss of generality that S, G and $\mathcal{H}_1^h$ all have common alphabets, it is known that

$\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_1^h) \iff [\forall\, i \in \{1, \ldots, h\}\ [\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_i)]]$

$\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_i)$ can be verified in polynomial time with respect to the encodings of S, G and H_i, so verifying $\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_1^h)$ is also in P. ■

By similar reasoning, the following proposition can also be shown:

Proposition 6 *Given a set of controllers $S_1, \ldots, S_s$, a set of finite-state automata system modules $G_1, \ldots, G_g$ and a finite-state automata specification H, the problem of verifying $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_1^s/G_1^g)$ is in P.*

Proof: Because it is assumed without loss of generality that $\mathcal{S}_1^s$, $\mathcal{G}_1^g$ and H all have common alphabets, it is known that

$\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_1^s/G_1^g) \iff$

$[\forall\, i \in \{1, \ldots, s\}\ [\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_i)]] \wedge [\forall\, j \in \{1, \ldots, g\}\ [\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G_j)]]$

$\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_i)$ and $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G_j)$ can be verified in polynomial time with respect to the encodings of S_i, G_j and H, so verifying $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_1^s/G_1^g)$ is in P.

■

4.7 Online Control Actions

Previously there have been several attempts in the discrete-event systems community to use online control methods to synthesize controllers whose proper actions are difficult to precompute. See for instance [83] and Chapter V where online methods for safe decentralized control are discussed. One might think that similar online approaches might be used to synthesize safe controllers for modular systems that restrict behavior in a non-trivial manner, i.e., enable at least one event, but in general this is not possible to do in an efficient manner. A further discouraging result of the work presented earlier is that for a modular system $\mathcal{G}_1^g$ and a modular specification

$\mathcal{H}_1^h$, calculating if a single controllable event is safe to enable from the initial state is PSPACE-complete. This problem is called the single event problem.

Theorem 9 *Given a set of modular finite-state automata $\mathcal{G}_1^g$, a modular finite-state automata specification $\mathcal{H}_1^h$, a set of controllable events Σ_c and a set of observable events Σ_o, the problem of deciding if a controllable event σ is safe to be enabled at the initial state is PSPACE-complete.*

Proof: It is first shown that this decision problem is in PSPACE. Let S be a controller that enables only σ at the initial state and disables all on the occurrence of any more events. It has already been shown that verifying $\mathcal{L}(S/G_1^g) \subseteq \mathcal{L}(H_1^h)$ is in PSPACE, so the verification problem in Proposition 4 can be used to solve the problem in this proposition.

It is now shown that this problem is PSPACE-complete. Suppose two arbitrary sets of automata $\mathcal{B}_1^b$ and $\mathcal{A}_1^a$ are given. The PSPACE-complete problem of deciding if $L(B_1^b) \subseteq L(A_1^a)$ is reduced to the single event problem using a polynomial-time many-one reduction. This will show that deciding if a single event is valid to be enabled is PSPACE-complete.

Let Σ be the alphabets for $\mathcal{B}_1^b$ and $\mathcal{A}_1^a$ and let α be an event not in Σ. Suppose for every $B_i, i \in \{1, \ldots, b\}$ an automaton $\breve{B}_i$ is created from B_i such that $L(\breve{B}_i) = \{\alpha\}L(B_i)$ in the following manner. Suppose x_{0i} is the initial state of B_i. $\breve{B}_i$ is a copy B_i except a new start state $\breve{x}_{0i}$ is created and the only transition from $\breve{x}_{0i}$ is on the occurrence of α and leads to x_{0i}. This procedure is repeated to create $\breve{A}_j$ from A_j for $j \in \{1, \ldots, a\}$. Let $\breve{\mathcal{B}}_1^b$ be the system and let $\breve{\mathcal{A}}_1^a$ be the specification. Let $\Sigma_c = \{\alpha\}$ and let Σ_o be empty. It should be apparent that this construction can be completed in polynomial time with respect to the encodings of $\mathcal{B}_1^b$ and $\mathcal{A}_1^a$.

If $L(B_1^b) \subseteq L(A_1^a)$ then it is known that α can be enabled at the initial state and have a safe system because enabling α will not allow illegal behavior to occur. Similarly, if $L(B_1^b) \not\subseteq L(A_1^a)$ then it is known that α can not be enabled at the initial state and have a safe system because enabling α will lead to further illegal behavior because $L(\breve{B}_1^b) \not\subseteq L(\breve{A}_1^a)$. So, $L(B_1^b) \subseteq L(A_1^a)$ if and only if α can be enabled at the initial state and have a safe system with respect to $\breve{\mathcal{B}}_1^b$, $\breve{\mathcal{A}}_1^a$, Σ_c and Σ_o This completes the polynomial-time many-one reduction. ■

Theorem 9 can be extended to show PSPACE-hardness or PSPACE-completeness results for many common online control problems where multiple online control actions for safety or maximality are computed.

4.8 Discussion

It is shown in [19] that several controller existence problems with range specifications on distributed systems are NP-hard. The results of this chapter extend the results in [19] using different reduction methods. This chapter has shown that a large class of automata intersection problems are PSPACE-complete, even for supposedly "simpler" prefix-closed cases. These automata intersection results were then used to show many controller existence problems in the supervisory control of discrete-event systems framework are likewise PSPACE-complete and with controller verification problems. Deciding controller admissibility is also PSPACE-complete for modular systems. Similarly, it is shown that in an online control setting that calculating safe control actions for distributed systems are PSPACE-hard.

Despite the fact that there are most likely no time-efficient general methods to solve many distributed systems problems (unless P=PSPACE), methods still need to be developed to solve many of these problems. One possible solution might be

to focus research efforts on important subclasses of these problems that might have particular real-world relevance. For instance, it might be helpful to look at specific network architectures or at problems involving systems amenable to divide-and-conquer approaches. The remainder of this thesis focuses on investigating these special subproblems and other efforts to avoid the computational difficulty shown in this chapter. Due to the results of [66] it is known that in general decentralized control systems cannot be synthesized in polynomial time for systems to match specification. The next chapter discusses methods for avoiding this computational difficulty through the use of online decentralized control protocols for monolithic systems with easily testable sufficient safety conditions. Later chapters focus on other ways of avoiding computational difficulty such as using heuristic methods to solve difficult sensor selection problems as in Chapter VI and using symmetry reductions when distributed systems contain a type of symmetry as in Chapter VII and Chapter VIII.

CHAPTER V

ONLINE DECENTRALIZED CONTROL OF DISCRETE-EVENT SYSTEMS

5.1 Chapter Overview

This chapter discusses several online decentralized control protocols that can be used to control systems and attempts to achieve maximal subsets of safe system behavior. These protocols have a common sufficient safety condition that is a form of a sensor and actuator selection problem. A generalized control architecture is used where controllable events are said to follow either "fusion by union" or "fusion by intersection" policies. This generalized architecture allows for a larger class of systems to be modelled. The new control protocols make use of a new state estimator used with decentralized controllers that takes past control actions into account. This new state estimator is then the basis of the several new control protocols. Two of these decentralized control protocols generate maximal local control actions that allow an amount of "steering" of the controlled system. Several properties of these locally maximal control protocols are discussed.

5.2 System Assumptions and General Decentralized Control

As previously stated, this chapter assumes a more general version of decentralized control introduced in [83] where the controllable events Σ_c are partitioned into event sets Σ_{cd} and Σ_{ce} that respectively follow either "fusion by union" or "fusion by intersection" protocols. If a "fusion by union" event is enabled by any local controllers, then it is enabled globally (hence the e subscript on Σ_{ce}). The Σ_{ce} events are also called the "permissive" events. Conversely, if a "fusion by intersection" event is enabled by at least one controller, then it is enabled globally. The set Σ_{cd} denotes the fusion by intersection events and its events are also called the "anti-permissive" events. See Figure 5.2 for a schematic of a system with general decentralized control.

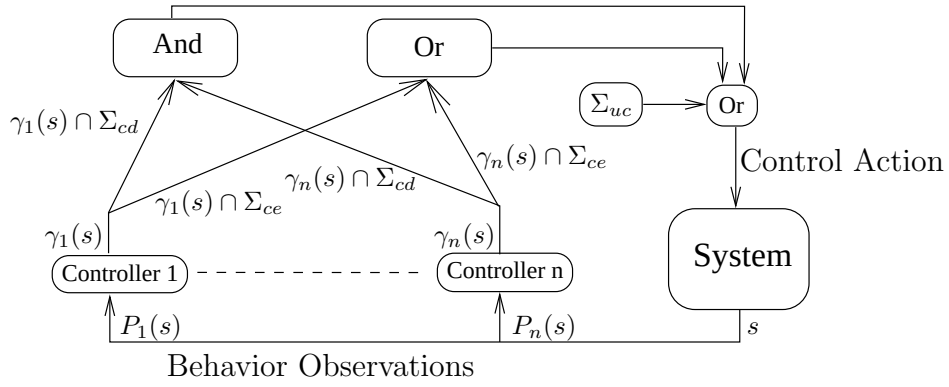

Figure 5.1: A schematic of a general decentralized control system.

The concept referred to as co-observability in Definition 3 is called "C&P co-observability" in [83] and is relevant to events that follow the fusion by intersection policy for locally enabled events. A dual property called D&A co-observability, defined below, is relevant to events that follow the fusion by union policy for locally enabled events.

Definition 7 *[83] A language K is* D&A co-observable *with respect to M, P_i, and*

Σ_{ci}, $i \in \{1, \ldots, n\}$ *if for all* $t \in \overline{K}$ *and for all* $\sigma \in \Sigma_c = \cup_{i=1}^{n} \Sigma_{ci}$,

$t\sigma \in \overline{K} \Rightarrow$

$$(\exists i \in I) \left[\left[\left(P_i^{-1}(P_i(t)) \cap \overline{K} \right) \sigma \cap M \subseteq \overline{K} \right] \wedge [\sigma \in \Sigma_{ci}] \right].$$

Similar to the $\mathcal{M}$ automaton method of [67], there is a method in [83] for deciding the D&A co-observability of languages generated by deterministic automata. This method is shown in Appendix B for the two-controller case. The essence of this method is that for a system G, a specification automata H, sets of controllable events Σ_{c1}, Σ_{c2} and sets of observable events Σ_{o1}, Σ_{o2}, an automaton $\mathcal{M}_d$ is constructed and $\mathcal{L}_m(\mathcal{M}_d) = \emptyset$ if and only if the $\mathcal{L}_m(H)$ is D&A co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2} and Σ_{c1}, Σ_{c2}.

In [83] the term *general co-observability* is introduced that combines C&P co-observability and D&A co-observability.

Definition 8 *[83] A language K is has the property of general co-observability w.r.t. M, P_i, Σ_{cdi} and Σ_{cei}, $i \in \{1, \ldots, n\}$ if*

1.K is C&P co-observable w.r.t. M, P_i and Σ_{cei}, $i \in \{1, \ldots, n\}$.

2.K is D&A co-observable w.r.t. M, P_i and Σ_{cdi}, $i \in \{1, \ldots, n\}$.

In [83] there is a version of the controllability and co-observability theorem (Theorem 1) where general co-observability replaces co-observability in the set of necessary and sufficient conditions for general decentralized controller existence. Like C&P co-observability and D&A co-observability, general co-observability can be decided in polynomial time for languages specified by finite automata if the number of controllers is bounded. If given a system G, a specification automaton H, sets of controllable events $\Sigma_{ce1}, \Sigma_{ce2}, \Sigma_{cd1}, \Sigma_{cd2}$ and sets of observable events Σ_{o1}, Σ_{o2}, construct $\mathcal{M}$ from G, H, $\Sigma_{ce1}, \Sigma_{ce2}$ and Σ_{o1}, Σ_{o2} as in Appendix A and construct the automaton

$\mathcal{M}_d$ from G, H, $\Sigma_{cd1}, \Sigma_{cd2}$ and Σ_{o1}, Σ_{o2} as in Appendix B. Then, $\mathcal{L}_m(\mathcal{M}) = \emptyset$ and $\mathcal{L}_m(\mathcal{M}_d) = \emptyset$ if and only if $\mathcal{L}_m(H)$ is general co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2} and $\Sigma_{ce1}, \Sigma_{ce2}, \Sigma_{cd1}, \Sigma_{cd2}$. Therefore, general co-observability can be tested in the amount of time it takes to construct $\mathcal{M}$ and $\mathcal{M}_d$.

This chapter deals exclusively with prefix-closed regular languages, so without loss of generality, this chapter uses the terminology that as system behavior progresses, the "state" of a system is the string of events generated by the system. Furthermore, the actions of a control protocol are considered to be a function $\gamma : \Sigma^* \rightarrow 2^{\Sigma_c}$ of the observed string of events generated by the system. However, due to the regularity assumption, the local control actions can be calculated from the state of some finite observer automaton when the control schemes are implemented. Also, because of the prefix-closure assumptions, this chapter exclusively investigates the safety properties of the control protocols when operating on a monolithic system G. The "possible" system states are those states the system could eventually enter if no control is present, i.e., all stings in $\mathcal{L}(G)$. A state estimate is a set of possible states the system may be in at a given instance. A "state estimator" is a function that returns an estimate of the current system state during system operation. For instance, using standard notation, if $P_i^{-1}(P_i(s))$ is a set of strings that an observer i considers could have occurred in the system after observing string $P_i(s)$, the set $P_i^{-1}(P_i(s))$ is called an estimate of the current system state. The control actions can be thought to be indirectly functions of the string of events generated by the system.

Suppose during the course of controller operation the string of events s has occurred in the system and that controller i enables a set of events $\gamma_i(s)$. An event $\sigma \in \gamma_i(s)$ is called a "don't care" event if the possible future behavior of the system after s is not altered if σ were to be disabled. That is, enabling σ after s does not

contribute to system behavior.

Due to the controllability and co-observability theorem, if a specification is controllable and co-observable with respect to a system, then there exists a set of decentralized controllers such that the controlled system can match the specification. If a safety specification language accepted by a finite state automaton is not controllable, an automaton accepting a maximal controllable sublanguage of the original language can be found in polynomial time. However, if a language accepted by a finite state automaton is not co-observable, there is no known method to find an automaton accepting a maximal co-observable sublanguage of that language or even if such an automaton exists at all.

5.3 Previous General Decentralized Control Work

There has been previous work in [83] on an online decentralized control protocol called *gdec* in the general decentralized control framework. This control protocol is formulated such that if a string s is generated by the system, each local controller i enforces the control action $\gamma_i^{gdec}(s)$ for a control specification K:

$$\begin{aligned} \gamma_i^{gdec}(s) \;=\; & \left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ & \left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s))\sigma \cap \overline{K} \neq \emptyset\right\} \cup \\ & \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}. \end{aligned} \tag{5.1}$$

Notice that in $\gamma_i^{gdec}(s)$, the set $P_i^{-1}(P_i(s))\sigma$ is an estimate of what states the system may be in immediately after event σ has occurred (and before any unobservable events have occurred).

In the definition of $\gamma_i^{gdec}(s)$, the set $\left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s))\sigma \cap \overline{K} \neq \emptyset\right\}$ represents all of the permissive events that i can control and leaves enabled. If $P_i^{-1}(P_i(s))\sigma \cap \overline{K}$

is an empty set, then when the event σ occurs in any of the current estimated states $(P_i^{-1}(P_i(s)))$, the resulting state of the system will not be in the legal set of states $(\overline{K})$. Naturally, if $\left(P_i^{-1}(P_i(s))\sigma \cap \overline{K} \neq \emptyset\right)$ holds, then σ can lead from a current legal estimated system state to another legal system state and so, σ is admissible as a permissive event and should be enabled. If σ is never possible in any of the system states, $\left(P_i^{-1}(P_i(s))\sigma \cap \mathcal{L}(G) = \emptyset\right)$ holds. In this case, σ is a "don't care" event, and σ is not enabled because

$$\left(P_i^{-1}(P_i(s))\sigma \cap \mathcal{L}(G) = \emptyset\right) \Rightarrow \left(P_i^{-1}(P_i(s))\sigma \cap \overline{K} = \emptyset\right).$$

Also in $\gamma_i^{gdec}(s)$, the set $\left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\}$ identifies the set of all anti-permissive events that i can control and enables. $\left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)$ identifies all estimated current system states that are legal. The set $\left(\left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G)\right)$ identifies all possible and likely states the system could be in if the event σ were to occur at a legal state. Note that $\left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma$ could be too large of a state estimate and might include behavior that is impossible even in the uncontrolled system, so there are some system states which may be disregarded. If a string is not possible, it will obviously not be legal, but unreachable system states need not be considered calculating a control strategy. If $\left(\left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right)$ holds it is known that any occurrence of σ at the current system state will not lead from legal to illegal behavior.

Note that it is possible that $\left(\left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) = \emptyset\right)$. This happens when σ cannot occur in any legal system states and yet still leads to a possible system state. In this case, anti-permissive events that lead to other illegal states could be enabled using the $\gamma_i^{gdec}(s)$ control protocol. This can be disregarded if σ is enabled or not in these "don't care" situations if generating safe sublanguages of

$\mathcal{L}(G)$ is the only concern because it is assumed the system is always in a legal state.

In this chapter online decentralized control protocols that improve upon $\gamma_i^{gdec}(s)$ are shown by upgrading the controller's state estimation ability. For the general decentralized control protocol described above, the expression $P_i^{-1}(P_i(s))$ operates as a state estimate for the i^{th} controller. As was indicated above, $P_i^{-1}(P_i(s))$ may include strings that are not possible in the uncontrolled system so $P_i^{-1}(P_i(s))$ could be a generous overestimate of the current system state. Because $P_i^{-1}(P_i(s))$ is too large, controller i, using the $P_i^{-1}(P_i(s))$ state estimate, could disable anti-permissive events unnecessarily. Intuitively, if a more accurate state estimate could be obtained, a supervisor could enable more anti-permissive and hence achieve a larger legal sublanguage.

5.4 Improved State Estimator

A new state estimation function is now developed for use with decentralized control protocols. It is assumed that the state estimator makes observations of system behavior with respect to observation function ($P_i(\cdot)$) and observes local control actions ($\gamma_i(\cdot)$). No assumptions are made on the local control action $\gamma_i(\cdot)$ or the actions of other controllers except that all controllers are admissible. If an event is disabled by the local controller that no other controller can enable, the local state estimator could disregard behavior progressing from that event when calculating the controlled unobservable reach from the current estimated states. Therefore, all unobservable events that controller i enables at state s ($\gamma_i(s)$) should be considered possible.

When calculating the state estimate in this manner, the state estimator needs to be sure that all events that could be enabled by other controller are considered possible so that the true system state is guaranteed to be in a controller's state

estimate. This prompts the definition of a "valid" state estimate, seen below.

Definition 9 Valid State Estimate*: A state estimate of a system such that the estimate includes all states the system could be in.*

It needs to be ensured that all state estimates used for control are valid. Therefore, a state estimator needs to be as conservative as possible when deciding if an event will be enabled by any other controller. For the rest of this chapter, the simplifying definition of $\Sigma_{cd}^{-i} \equiv \cup_{j\in I, j\neq i}\Sigma_{cdj}$ is used.

In the framework of general decentralized control, any unobservable event in Σ_{cd} that can be enabled by a local controller other than i $(\cup_{j\in I, j\neq i}\Sigma_{cdj})$should be considered possible. Even though a local control action $\gamma_i(s)$ may locally disable events in Σ_{cd}^{-i}, other controllers may enable those events without i observing. Furthermore, unobservable events in the set $\Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}$ should also always be considered possible, but events in this set should also always be enabled by the local control action $\gamma_i(\cdot)$ if i is a valid controller and does not try to disable events it cannot control. Because of the assumption of the admissibility of $\gamma_i(\cdot)$, the set $\Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}$ is not included in the list of possible events the state estimate should consider possible because $\gamma_i(\cdot)$ should already be included.

If any event in Σ_{cei} is disabled by i, then no other controller can enable that event and therefore, unobservable events in $\Sigma_{cei} \setminus \gamma_i(\cdot)$ do not need to be considered possible because no other controller could enabled them globally. By similar reasoning, unobservable events in $\Sigma_{cdi} \setminus \left[\gamma_i(\cdot) \cup \Sigma_{cd}^{-i}\right]$ do not need to be considered possible. Now, to combine all the statements just made about the unobservable controlled reach of the system, the set of all events that are considered possible to be enabled globally can be expressed as $\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}$ and the set of all strings

that could occur after the last observable event without the knowledge of i is included in $\left[\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*$.

The function $PS_i : \Sigma^* \rightarrow 2^{\Sigma^*}$ is used to express this new state estimator constructed from controller $\gamma_i(\cdot)$. The formal definition for $PS_i(s)$ can be seen below and is necessarily recursive because the state estimate is continually updated as i observes events and alters $\gamma_i(\cdot)$ is updated.

Definition 10

$$PS_i(s) = \begin{cases} \left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G) & \text{for } s = \varepsilon \\ \left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s'\sigma) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G) & \text{for } s = s'\sigma, s \neq \varepsilon \end{cases} \tag{5.2}$$

Before any events have been observed, the set of all unobservable strings that could have occurred with knowledge of the initial local control action $\gamma_i(\varepsilon)$ can be expressed as $PS_i(\varepsilon) = \left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$. It is assumed in this thesis that control actions can be updated instantaneously with respect to the occurrence of unobservable events. Therefore, local control actions can be calculated before unobservable events occur. In the definition of $PS_i(\cdot)$, the intersection of $\left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*$ with $\mathcal{L}(G)$ ensures that i considers only possible unobservable strings likely to have occurred.

Now, suppose the system has been operating for a while and string s' has occurred in the system. Controller i would then have $PS_i(s')$ as a state estimate. Now suppose an event $\sigma \in \Sigma_{oi}$ were to occur. Because σ is observable $P_i(\sigma) = \sigma$. After σ occurs, but before any unobservable events were to occur i would believe the system would be in a state in the set $PS_i(s')P_i(\sigma)$. After the control action takes at affect at state $s'\sigma$, controller i knows only strings in $\left[\left(\gamma_i(s'\sigma) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*$ can occur. So, controller i infers a string in $\left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s'\sigma) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right)$

could have occurred. Intersecting this set with the possible language, $\mathcal{L}(G)$ produces the state estimate $\left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s'\sigma) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G)$. Therefore, if σ is observable to i,

$$PS_i(s'\sigma) = \left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s'\sigma) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G).$$

Suppose now that σ is not observable to i. Therefore, $P_i(\sigma) = \varepsilon$ and $\gamma_i(s'\sigma) = \gamma_i(s')$. With these facts, it should be evident that

$$\begin{aligned} & \left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s'\sigma) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G) \\ = \; & \left(PS_i(s')\left[\left(\gamma_i(s') \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G) \\ = \; & PS_i(s'). \end{aligned}$$

So if σ is not observable to i,

$$PS_i(s'\sigma) = \left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s'\sigma) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G).$$

and $PS_i(s'\sigma) = PS_i(s')$, which should necessarily true because a controller should not change its state estimate on unobservable events.

It should be fairly intuitive from the above discussion that $PS_i(s)$ returns a valid state estimate. The essence of the proof is that when calculating $PS_i(s)$, the only events not considered possible to occur after the control action is updated are events that are guaranteed not to occur.

Now that the $PS_i(s)$ state estimator has been introduced, it is shown that the set of states considered likely by $PS_i(s)$ is always at least as small as $P_i^{-1}(P_i(s))$. The proof for this statement is based on the fact that $PS_i(s)$ considers at most the same number of unobservable events likely to occur as $P_i^{-1}(P_i(s))$ as system operation evolves.

Theorem 10 $PS_i(s) \subseteq P_i^{-1}(P_i(s))$.

Proof: $PS_i(s)$ represents all the strings i believes could have occurred when past control actions are taken into account. $P_i^{-1}(P_i(s))$ represents all the strings i believes could have occurred where control action is not taken into account. This lemma is proved by induction on the length of the string s.

Base:

$|s| = 0$, or $s = \varepsilon$.

$PS_i(\varepsilon) = \left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$.

$P_i^{-1}(P_i(\varepsilon)) = \Sigma_{uoi}^*$.

$\left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G) \subseteq \Sigma_{uoi}^*$.

$PS_i(\varepsilon) \subseteq P_i^{-1}(P_i(\varepsilon))$, which demonstrates the base of the induction proof.

Induction hypothesis:

For $|s'| = n - 1$.

$PS_i(s') \subseteq P_i^{-1}(P_i(s'))$.

Induction step:

For $|s| = n$.

Let $s = s'\sigma$ so $|s'| = n - 1$.

$P_i^{-1}(P_i(s)) = P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^*$.

$PS_i(s) = \left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G)$.

$PS_i(s') \subseteq PS_i(s')$, by the induction hypothesis.

$PS_i(s')P_i(\sigma) \subseteq P_i^{-1}(P_i(s'))P_i(\sigma)$.

$PS_i(s')P_i(\sigma)\Sigma_{uoi}^* \subseteq P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^*$.

$PS_i(s')P_i(\sigma)\left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \subseteq PS_i(s')P_i(\sigma)\Sigma_{uoi}^*$.

$PS_i(s')P_i(\sigma)\left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \subseteq P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^*$.

$\left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(\varepsilon) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G)$.

$$\subseteq P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^*.$$

$PS_i(s) \subseteq P_i^{-1}(P_i(s))$. ■

5.5 Memory Based Non-maximal Control Protocol

Now that an improved state estimator for general decentralized control systems has been defined, it shown how to use this estimator to devise an online decentralized control protocol. The *gdec* protocol is first reviewed. Remember that this protocol uses $P_i^{-1}(P_i(s))$ as its state estimator:

$$\begin{aligned}\gamma_i^{gdec}(s) &= \left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ &\quad \left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s))\sigma \cap \overline{K} \neq \emptyset\right\} \cup \\ &\quad \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}.\end{aligned}$$

Let $s = s'\alpha$. The event α may or may not be observable.

$$\begin{aligned}\gamma_i^{gdec}(s'\alpha) &= \left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s'\alpha)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ &\quad \left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s'\alpha))\sigma \cap \overline{K} \neq \emptyset\right\} \cup \\ &\quad \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}.\end{aligned}$$

Therefore,

$$\begin{aligned}\gamma_i^{gdec}(s'\alpha) &= \left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s'))P_i(\alpha)\Sigma_{uoi}^* \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ &\quad \left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s'))P_i(\alpha)\Sigma_{uoi}^*\sigma \cap \overline{K} \neq \emptyset\right\} \cup \\ &\quad \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}.\end{aligned}$$

When calculating $\gamma_i^{gdec}(s)$, the possible uncontrolled and unobserved behavior of the system after the last event is calculated in order to compute which events should

be disabled. In [83] the set of strings Σ^*_{uoi} is used to represent the set of all strings unobservable to i that could occur after the last observable event if the system were not controlled. As before, it is assumed that the control action is updated immediately upon the observation of new events. Note that for $\gamma_i^{gdec}(s)$, state estimate $P_i^{-1}(P_i(s'))P_i(\alpha)$ represents all states the system could be in immediately after the last event has occurred and before any more unobservable events occur. This state estimate does not change with the occurrence of unobservable events.

The *gdec* protocol is now modified to take advantage of the $PS(\cdot)$ state estimator. For this control protocol, the $P_i^{-1}(P_i(s'))$ state estimator from $\gamma^{gdec}(s)$ is replaced with the $PS_i(s')$ estimator. Then, for this new protocol, if after s' has occurred in the system and the next event observed is α, strings in Σ^*_{uoi} could occur if no controllers were operating. Therefore, $(PS_i(s')P_i(\alpha)\Sigma^*_{uoi})$ would include all system behavior i would need to consider as possible when calculating its local control action after α is observed. This prompts the definition of $PS_i^+(s)$, as seen below. The set $(PS_i(s')P_i(\alpha)\Sigma^*_{uoi})$ is intersected with the uncontrolled system behavior $\mathcal{L}(G)$ to assure that the controller only accounts for possible behavior. The projection $P_i(\alpha)$ is used instead of α to represent the last event to occur in the system because if α were unobservable to i, it must be true that $PS_i^+(s'\alpha) = PS_i^+(s')$. This ensures that control actions cannot be updated on the occurrence of unobservable events.

At initialization, ε is the current system state and $\Sigma^*_{uoi} \cap \mathcal{L}(G)$ includes all unobservable and uncontrolled behavior of the system that needs to be accounted for when calculating the initial local control action. Therefore, due to the recursive nature of the $PS_i^+(s)$ definition, $PS_i^+(\varepsilon)$ is defined separately for deciding the initial control action.

Definition 11

$$PS_i^+(s) = \begin{cases} \Sigma_{uoi}^* \cap \mathcal{L}(G) & for\ s = \varepsilon \\ (PS_i(s')P_i(\sigma)\Sigma_{uoi}^*) \cap \mathcal{L}(G) & for\ s = s'\sigma, s \neq \varepsilon \end{cases} \tag{5.3}$$

The set $PS_i(s)$ represents all strings controller i considers could have occurred with knowledge of all past control actions taken by i. $PS_i^+(s)$ represents the set of all strings controller i considers could have occurred if no control action is taken by i after the last observable event. The set $PS_i^+(s)$ can be used to determine which events need to be disabled. The *gdec* control protocol is now rewritten to take advantage of this improved state estimator.

Definition 12 *gmdec: General Memory Based Decentralized Control Protocol*

$$\begin{aligned} \gamma_i^{gmdec}(s) \;=\; & \left\{\sigma \in \Sigma_{cdi} : \left(PS_i^+(s) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \\ & \cup \left\{\sigma \in \Sigma_{cei} : PS_i^+(s)\sigma \cap \overline{K} \neq \emptyset\right\} \\ & \cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}. \end{aligned} \tag{5.4}$$

The notation is used that $S_{gmdec}(s)$ is the global control action generated by combining all of the memory based local control protocols $\{\gamma_1^{gmdec}(s), \ldots, \gamma_n^{gmdec}(s)\}$ in the same manner that *gdec* combines all of its local control protocol s according to the general decentralized control protocol. An anti-permissive event ($\sigma_d \in \Sigma_{cd}$) is globally enabled by $S_{gmdec}(s)$ if and only if there is some ($i \in I$) such that $\sigma_d \in \gamma_i^{gmdec}$. A permissive event ($\sigma_e \in \Sigma_{ce}$) is enabled by $S_{gmdec}(s)$ if and only if there is no ($i \in I$) such that $\sigma_e \in \gamma_i^{gmdec}$. The language generated by the global control protocol S_{gmdec} operating on the system G is represented as $\mathcal{L}(S_{gmdec}/G)$.

Consider the following example of this *gmdec* control protocol being used with a system and compared with the behavior of *gdec*.

Example 1 *Suppose there are two controllers for the uncontrolled system G seen in Figure 5.2. Suppose further that $\overline{K} = \mathcal{L}(H)$. The controllers have the following properties:*

$\Sigma_{o1} = \{\phi\}$, $\Sigma_{o2} = \emptyset$, $\Sigma_{c1} = \{\alpha, \beta\}$, $\Sigma_{c2} = \{\phi, \theta\}$, $\Sigma_{uc} = \emptyset$, $\Sigma_{ce} = \{\beta, \phi, \theta\}$, $\Sigma_{cd} = \{\alpha\}$.

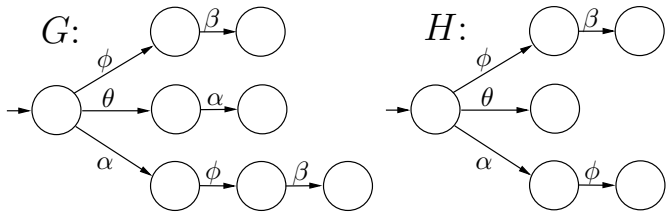

Figure 5.2: The uncontrolled system G and desired system H for Example 1.

The two languages $\mathcal{L}(S_{gmdec}/G)$ and $\mathcal{L}(S_{gdec}/G)$ are now compared as system behavior is simulated:

To calculate $\mathcal{L}(S_{gmdec}/G)$:	*To calculate $\mathcal{L}(S_{gdec}/G)$:*
$PS_1^+(\varepsilon) = \{\varepsilon, \theta, \theta\alpha, \alpha, \alpha\phi, \alpha\phi\beta\}$	$P_1^{-1}(P_1(\varepsilon)) \cap \mathcal{L}(G)$
$\gamma_1^{gmdec}(\varepsilon) = \{\beta, \phi, \theta\}$	$= \{\varepsilon, \theta, \theta\alpha, \alpha, \alpha\phi, \alpha\phi\beta\}$
$PS_1(\varepsilon) = \{\varepsilon, \theta\}$	$\gamma_1^{gdec}(\varepsilon) = \{\beta, \phi, \theta\}$
$PS_1^+(\phi) = \{\phi, \phi\beta\}$	$P_1^{-1}(P_1(\phi)) \cap \mathcal{L}(G)$
$\gamma_1^{gmdec}(\phi) = \{\alpha, \beta, \phi, \theta\}$	$= \{\phi, \phi\beta, \alpha\phi, \alpha\phi\beta\}$
$PS_1(\phi) = \{\phi, \phi\beta\}$	$\gamma_1^{gdec}(\phi) = \{\alpha, \phi, \theta\}$
$PS_2^+(\varepsilon) = \{\varepsilon, \phi, \phi\beta, \theta, \theta\alpha, \alpha, \alpha\phi, \alpha\phi\beta\}$	$P_2^{-1}(P_2(\varepsilon)) \cap \mathcal{L}(G)$
$\gamma_2^{gmdec}(\varepsilon) = \{\beta, \phi, \theta\}$	$= \{\varepsilon, \phi, \phi\beta, \theta, \theta\alpha, \alpha, \alpha\phi, \alpha\phi\beta\}$
$PS_2(\varepsilon) = \{\varepsilon, \phi, \phi\beta, \theta, \theta\alpha, \alpha, \alpha\phi, \alpha\phi\beta\}$	$\gamma_2^{gdec}(\varepsilon) = \{\beta, \phi, \theta\}$
Therefore, $\mathcal{L}(S_{gmdec}/G) = \{\varepsilon, \phi, \phi\beta, \theta\}$	*Therefore,* $\mathcal{L}(S_{gdec}/G) = \{\varepsilon, \phi, \theta\}$.

These systems can be seen in Figure 5.3.

For this problem instance, $\mathcal{L}(S_{gdec}/G) \subset \mathcal{L}(S_{gmdec}/G)$. Using gmdec, controller

1 remembers α was not enabled at the initial state and therefore knows the system is not at state $\alpha\phi$ after ϕ is observed. Therefore, gmdec legally enables β at state ϕ, but gdec cannot do this because controller 1 using gdec does not know if ϕ or $\alpha\phi$ has occurred after ϕ has been observed.

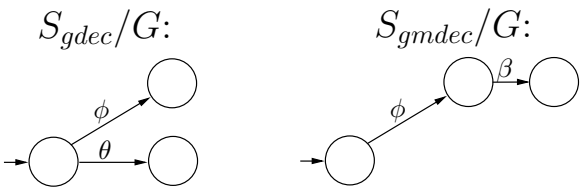

Figure 5.3: S_{gdec}/G and S_{gmdec}/G for Example 1.

5.5.1 Language Properties

Several properties of the language generated by systems using the *gmdec* control protocol are now investigated. A sufficient safety condition is shown after demonstrating several necessary lemmas. Let $\overline{K}$ represent the legal language to be achieved by controlling the system G. As usual, $\mathcal{L}(G)$ represents the possible behavior of G.

Lemma 1 $PS_i(s) \subseteq PS_i^+(s)$

Proof: $PS_i(s)$ represents all the strings i believes could have occurred where all events are controlled. $PS_i^+(s)$ represents all the strings i believes could have occurred where all events after the last observed event are uncontrolled.

$\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi} \subseteq \Sigma_{uoi}$.

$\left[\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \subseteq \Sigma_{uoi}^*$.

$\left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \subseteq (PS_i(s')P_i(\sigma)\Sigma_{uoi}^*)$.

$\left(PS_i(s')P_i(\sigma)\left[\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \cap \mathcal{L}(G) \subseteq$.

$(PS_i(s')P_i(\sigma)\Sigma_{uoi}^*) \cap \mathcal{L}(G)$.

$PS_i(s) \subseteq PS_i^+(s)$. ■

Lemma 2 $PS_i^+(s) \subseteq P_i^{-1}(P_i(s))$

Proof: $PS_i^+(s)$ represents all the strings i believes could have occurred where past control actions are taken into account. $P_i^{-1}(P_i(s))$ represents all the strings i believes could have occurred where the control action after the last observable event is not taken into account. This lemma is proved by induction on the length of the string s.

Base:

$|s| = 0$, or $s = \varepsilon$.

$PS_i^+(\varepsilon) = \Sigma_{uoi}^* \cap \mathcal{L}(G)$.

$P_i^{-1}(P_i(\varepsilon)) = \Sigma_{uoi}^*$.

$\Sigma_{uoi}^* \cap \mathcal{L}(G) \subseteq \Sigma_{uoi}^*$.

$PS_i^+(\varepsilon) \subseteq P_i^{-1}(P_i(\varepsilon))$.

Induction hypothesis:

for $|s'| = n - 1$.

$PS_i^+(s') \subseteq P_i^{-1}(P_i(s'))$.

Induction step:

for $|s| = n$.

Let $s = s'\sigma$ so $|s'| = n - 1$.

$P_i^{-1}(P_i(s)) = P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^*$.

$PS_i^+(s) = (PS_i(s')P_i(\sigma)\Sigma_{uoi}^*) \cap \mathcal{L}(G)$.

$PS_i(s') \subseteq PS_i^+(s')$.

$PS_i(s')P_i(\sigma)\Sigma_{uoi}^* \subseteq PS_i^+(s')P_i(\sigma)\Sigma_{uoi}^* \subseteq P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^*$.

$(PS_i(s')P_i(\sigma)\Sigma_{uoi}^*) \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^*$.

$PS_i^+(s) \subseteq P_i^{-1}(P_i(s))$ which completes the proof by induction. ■

If $\overline{K}$ is controllable and co-observable, then $\overline{K} = \mathcal{L}(S_{gmdec}/G)$. This statement is proved later in this chapter because the proof depends on properties comparing the languages $\mathcal{L}(S_{gmdec}/G)$ and $\mathcal{L}(S_{gdec}/G)$. It is assumed without loss of generality that $\overline{K}$ is controllable as an automaton generating the supremal controllable sublanguage of the language specified by a finite state automaton can be computed in polynomial time. Unfortunately, there is no known algorithm to compute a maximal controllable and co-observable sublanguage of $\overline{K}$ if one exists. However, if $\overline{K}$ is controllable but not co-observable, *gmdec* can be used to attempt to achieve as large a sublanguage of $\overline{K}$ as possible, thereby satisfying the main safety condition.

A sufficient condition for the safety of the *gmdec* protocol when used with a system G and a language specification $\overline{K}$ that is not co-observable is provided here and is the same as the one in [68] for *gdec*. It is based on the fact that if $(\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd})$ holds then no permissive events will lead to illegal behavior. To review, all events that lead to a marked state in the $\mathcal{M}$ automaton violate C&P co-observability for the system if $\Sigma = \Sigma_{ce}$. If all events that violate C&P co-observability from the $\mathcal{M}$ automaton are assigned to be anti-permissive, those events will never lead to illegal behavior in the original system.

Anti-permissive events are enabled only when a controller is sure they will not lead to illegal behavior. Therefore, anti-permissive events, by their very nature in general decentralized control, never lead to illegal behavior. All permissive events not identified by the $\mathcal{M}$ automaton will also not lead to illegal behavior because they do not violate co-observability. There will always be at least one controller for those valid permissive events that will know to and can disable those events in time if they could lead to illegal behavior. Furthermore, because $\overline{K}$ is assumed to be controllable, uncontrollable events will never lead to from legal to illegal states.

Theorem 11 $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{gmdec}/G) \subseteq \overline{K}$

Proof: It is sufficient to prove that $(s \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow \left(s \in \overline{K}\right)$ given the condition $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd}$. Without loss of generality, $\overline{K}$ is assumed to be non-empty and controllable. This lemma is proved by induction on the length of the string s.

Base:

$|s| = 0$, or $s = \varepsilon$.

Because $\overline{K}$ can be assumed to be nonempty, so $\varepsilon \in \overline{K}$ and $\varepsilon \in \mathcal{L}(S_{gmdec}/G)$.

Therefore, $(\varepsilon \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow \left(\varepsilon \in \overline{K}\right)$.

Induction hypothesis:

$|s'| = n - 1$.

$(s' \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow \left(s' \in \overline{K}\right)$.

Induction step:

Let $s = s'\sigma$ so $|s'| = n - 1$ and $|s| = n$.

The induction step is broken down into three cases, $\sigma \in \Sigma_{uc}$, $\sigma \in \Sigma_{ce}$, $\sigma \in \Sigma_{cd}$.

Case 1 *:* $\sigma \in \Sigma_{uc}$

For all $s' \in \mathcal{L}(G)$ and $i \in I$, $\Sigma_{uc} \subseteq \gamma_i(s')$, so $\Sigma_{uc} \subseteq S_{gmdec}(s')$.

Therefore, if $s'\sigma \in \mathcal{L}(G)$, then $s'\sigma \in \mathcal{L}(S_{gmdec}/G)$.

Also, because $\overline{K}$ is controllable, $s'\sigma \in \mathcal{L}(G)$ and $s' \in \overline{K}$, then $s'\sigma \in \overline{K}$.

so, if $\sigma \in \Sigma_{uc}, (s \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow \left(s \in \overline{K}\right)$.

The next two cases are proved by contradiction.

Case 2 *:* $\sigma \in \Sigma_{ce}$

Assume there exists a string $s' \in \mathcal{L}(S_{gmdec}/G) \cap \overline{K}$ and $\sigma \in \Sigma_{ce}$ such that $s'\sigma \in \mathcal{L}(S_{gmdec}/G)$ and $s'\sigma \notin \overline{K}$.

Since $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd}$ and $\Sigma_{cd} \cap \Sigma_{ce} = \emptyset$, $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \cap \Sigma_{ce} = \emptyset$.

Therefore, $\overline{K}$ is C&P co-observable w.r.t. $\mathcal{L}(G), \Sigma_{o1}, \Sigma_{ce1}, \ldots, \Sigma_{on}, \Sigma_{cen}$.

By definition of C&P co-observability:

$\left(s' \in \overline{K}\right) \wedge \left(s'\sigma \notin \overline{K}\right) \Rightarrow \left[(\exists i \in I)\left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$.

$PS_i^+(s') \subseteq P_i^{-1}(P_i(s'))$.

$\Rightarrow PS_i^+(s')\sigma \subseteq P_i^{-1}(P_i(s'))\sigma$.

$\Rightarrow PS_i^+(s')\sigma \cap \overline{K} \subseteq P_i^{-1}(P_i(s'))\sigma \cap \overline{K}$.

$\Rightarrow \left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \Rightarrow \left(PS_i^+(s')\sigma \cap \overline{K} = \emptyset\right)$.

Therefore:

$\left(s' \in \overline{K}\right) \wedge \left(s'\sigma \notin \overline{K}\right) \Rightarrow \left[\exists i \in I\left(PS_i^+(s')\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$.

$\Rightarrow \sigma \notin S_{gmdec}\left(s'\right)$.

$\Rightarrow s'\sigma \notin \mathcal{L}(S_{gmdec}/G)$.

This contradicts the assumption that $s'\sigma \in \mathcal{L}(S_{gmdec}/G)$, so if $\sigma \in \Sigma_{ce}$,

$$\left(s \in \mathcal{L}(S_{gmdec}/G)\right) \Rightarrow \left(s \in \overline{K}\right).$$

Case 3 *:* $\sigma \in \Sigma_{cd}$

Assume there exists a string $s' \in \mathcal{L}(S_{gmdec}/G) \cap \overline{K}$ and $\sigma \in \Sigma_{cd}$ such that $s'\sigma \in \mathcal{L}(S_{gmdec}/G)$ and $s'\sigma \notin \overline{K}$. It has been shown that $(\forall i \in I)\ s' \in PS_i^+(s')$. This property follows from the fact that PS_i is a valid state estimate, but for the sake of brevity, this property is not demonstrated here.

Because $s' \in \overline{K}$, $(\forall i \in I)\ s' \in PS_i^+(s') \cap \overline{K}$.

Therefore:$(\forall i \in I)\ s'\sigma \in \left(PS_i^+(s') \cap \overline{K}\right)\sigma$.

Furthermore, $(\forall i \in I)\ (s'\sigma \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow (s'\sigma \in \mathcal{L}(G))$.

So, $(\forall i \in I)\ s'\sigma \in \left(PS_i^+(s') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G)$ and $s'\sigma \notin \overline{K}$.

$\Rightarrow \left[(\forall i \in I)\left(\left(PS_i^+(s') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right)\right]$.

$\Rightarrow \left[(\forall i \in I)\left(\left(PS_i^+(s') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right) \vee (\sigma \notin \Sigma_{cdi})\right].$

$\Rightarrow \sigma \notin S_{gmdec}(s').$

$\Rightarrow s'\sigma \notin \mathcal{L}(S_{gmdec}/G).$

This contradicts the assumption that $s'\sigma \in \mathcal{L}(S_{gmdec}/G)$, so if $\sigma \in \Sigma_{cd}$,

$$(s \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow \left(s \in \overline{K}\right).$$

So, for all cases $(s \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow \left(s \in \overline{K}\right)$, which concludes the induction proof and demonstrates that

$$\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{gmdec}/G) \subseteq \overline{K}.$$

■

Note that the safety condition $(\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd})$ is in a sense a sensor and actuator selection condition. That is, the sets of locally observable events $\{\Sigma_{o1}, \ldots, \Sigma_{on}\}$ and the partition of controllable events $\{\Sigma_{ce}, \Sigma_{cd}\}$ need to be chosen such that for the $\mathcal{M}$ automaton construction, the only state transitions into the dump state are driven by events in Σ_{cd}. This prompts the interesting subproblem of finding the optimal cost sensor/actuator selection such that the safety condition is satisfied. A simpler but computationally similar version of this problem is discussed in Chapter VI.

It is shown in [83] that given a sufficient safety condition for $\mathcal{L}(S^1_{gdec}/G)$ and $\mathcal{L}(S^2_{gdec}/G)$, $\Sigma^2_{cd} \subseteq \Sigma^1_{cd} \Rightarrow \mathcal{L}(S^1_{gdec}/G) \subseteq \mathcal{L}(S^2_{gdec}/G)$. This property states that if more events are assigned to be anti-permissive, the behavior generated by a system controlled with the *gdec* protocol will be smaller. In other words, a more restrictive event partitioning will lead to a smaller generated language using the *gdec* control scheme. It would seem intuitive that this property would also hold for $\mathcal{L}(S_{gmdec}/G)$.

However, this is not always the case. In general, given a sufficient safety condition for $\mathcal{L}(S^1_{gmdec}/G)$ and $\mathcal{L}(S^2_{gmdec}/G)$, $\Sigma^2_{cd} \subseteq \Sigma^1_{cd} \not\Rightarrow \mathcal{L}(S^1_{gmdec}/G) \subseteq \mathcal{L}(S^2_{gmdec}/G)$.

Proposition 7 *Given* $\mathcal{L}(S^1_{gmdec}/G) \subseteq \overline{K}$ *and* $\mathcal{L}(S^2_{gmdec}/G) \subseteq \overline{K}$, $\Sigma^2_{cd} \subseteq \Sigma^1_{cd} \not\Rightarrow \mathcal{L}(S^1_{gmdec}/G) \subseteq \mathcal{L}(S^2_{gmdec}/G)$.

Proof: This proposition is demonstrated by example. Suppose there are two controllers (a and b) for the uncontrolled system G seen in Figure 5.4 and in the example above. As before, $\overline{K} = \mathcal{L}(H)$.

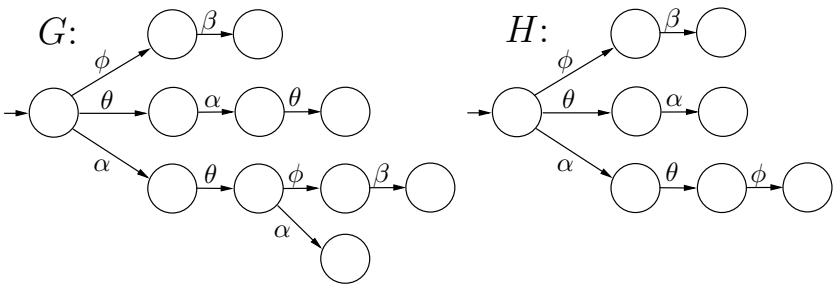

Figure 5.4: The system G and specification H for the proof of Proposition 7.

The controllers have the following properties:

$\Sigma_{oa} = \{\phi\}$, $\Sigma_{ob} = \emptyset$, $\Sigma_{ca} = \{\alpha, d\}$, $\Sigma_{cb} = \{\alpha, \beta, \theta\}$, $\Sigma_{uc} = \{\theta\}$.

First, let's construct the language generated by using the following partition on the controllable events:

$\Sigma^1_{ce} = \{\alpha\}$, $\Sigma^1_{cd} = \{\theta, \beta\}$.

$\mathcal{L}(S^1_{gmdec}/G)$, the language generated using Σ^1_{cd} and Σ^1_{ce} was calculated in the example above. $\mathcal{L}(S^1_{gmdec}/G) = \{\varepsilon, \alpha, \phi\}$.

This system can be seen in Figure 5.5.

Now the language generated is calculated by using the following partition on the controllable events:

$\Sigma^2_{ce} = \emptyset$, $\Sigma^2_{cd} = \{\alpha, \theta, \beta\}$.

Note that $\Sigma^1_{cd} \subseteq \Sigma^2_{cd}$.

To calculate $\mathcal{L}(S^2_{gmdec}/G)$, the language generated using Σ^2_{cd} and Σ^2_{ce} :

$PS^{+2}_a(\varepsilon) = \{\varepsilon, \theta, \theta\alpha, \theta\alpha\theta, \alpha, \alpha\theta, \alpha\theta\alpha\}$.

$\gamma^2_a(\varepsilon) = \{\phi\}$.

$PS^2_a(\varepsilon) = \{\varepsilon, \theta\}$.

$PS^{+2}_a(\phi) = \{\phi, \phi\beta\}$.

$\gamma^2_a(\phi) = \{\phi, \beta\}$.

$PS^2_a(\phi) = \{\phi, \phi\beta\}$.

$PS^{+2}_b(\varepsilon) = \{\varepsilon, \theta, \theta\alpha, \theta\alpha\theta, \alpha, \alpha\theta, \alpha\theta\alpha, \alpha\theta\phi, \alpha\theta\phi\beta, \phi, \phi\beta\}$.

$\gamma^2_b(\varepsilon) = \{\phi\}$.

$PS^2_b(\varepsilon) = \{\varepsilon, \theta, \theta\alpha, \theta\alpha\theta, \alpha, \alpha\theta, \alpha\theta\alpha, \alpha\theta\phi, \alpha\theta\phi\beta, \phi, \phi\beta\}$.

Therefore, $\mathcal{L}(S^2_{gmdec}/G) = \{\varepsilon, \phi, \phi\beta\}$. This system also can be seen in Figure 5.5.

So, $\mathcal{L}(S^1_{gmdec}/G) \not\subseteq \mathcal{L}(S^2_{gmdec}/G)$ in this case.

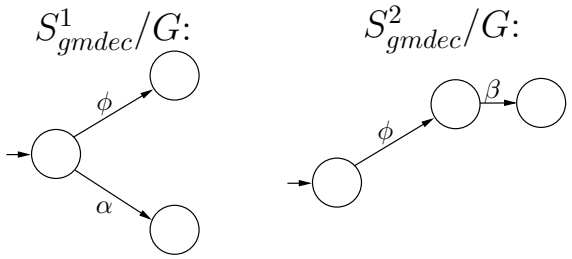

Figure 5.5: Resulting systems when *gmdec* operates on system G of Proposition 7.

Therefore, given $\mathcal{L}(S^1_{gmdec}/G) \subseteq \overline{K}$ and $\mathcal{L}(S^2_{gmdec}/G) \subseteq \overline{K}$,

$$\Sigma^2_{cd} \subseteq \Sigma^1_{cd} \not\Rightarrow \mathcal{L}(S^1_{gmdec}/G) \subseteq \mathcal{L}(S^2_{gmdec}/G). \tag{5.5}$$

■

At first, this result may seem counter-intuitive. A less restrictive control protocol should lead to a larger generated language, but this is obviously not the case for *gmdec*. The reason for this lies in the calculation of the state estimate, $PS_i(s)$. When controller i calculates $PS_i(s)$, $\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}$ are all events that could have occurred without being observed by i after the calculation of each control action. Even though $\Sigma_{cd}^2 \subseteq \Sigma_{cd}^1$, it may be the case that $PS_i^2(s) \not\subseteq PS_i^1(s)$ because $\left(\Sigma_{cd}^2 \subseteq \Sigma_{cd}^1 \not\Rightarrow \left(\gamma_i^1(s) \cup_{j \in I, j \neq i} \Sigma_{cdj}^1\right) \cap \Sigma_{uoi} \subseteq \left(\gamma_i^2(s) \cup_{j \in I, j \neq i} \Sigma_{cdj}^2\right) \cap \Sigma_{uoi}\right)$, as was seen in the example above. If $PS_i^1(s)$ is not contained in $PS_i^2(s)$, anti-permissive events that would be disabled in system 1 could be enabled in system 2. However, a condition could be found that guarantees $PS_i^1(s) \subseteq PS_i^2(s)$ for all i, anti-permissive events that are enabled in system 1 could be disabled in system 2, so $\mathcal{L}(S_{gmdec}^1/G)$ would included in $\mathcal{L}(S_{gmdec}^2/G)$. This curious property is discussed further later in the chapter.

5.5.2 Language Comparisons

Now that some of the basic properties of the *gmdec* control scheme have been introduced, it is interesting to see how the languages generated by *gmdec* compare to the languages generated by the *gdec* control scheme. A controller that accounts for its past control actions should be able to better control a system than a controller that bases its control actions only on the inverse projection of the observed string. Given that a sufficient safety condition has been met, a memory-based control protocol generates a language at least as large as the language generated by a conventional *gdec* protocol.

Theorem 12 *Given that* $\mathcal{L}(S_{gdec}/G)$ *is safe,* $\mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{gmdec}/G)$.

Proof: This proof is based on induction on the length of the strings s to show that given $\mathcal{L}(S_{gdec}/G)$ is safe, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{gmdec}/G))$.

Base:

$|s| = 0$, or $s = \varepsilon$.

Assuming without loss of generality that neither language is empty,

$(\varepsilon \in \mathcal{L}(S_{gdec}/G))$ and $(\varepsilon \in \mathcal{L}(S_{gmdec}/G))$.

Therefore, $(\varepsilon \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (\varepsilon \in \mathcal{L}(S_{gmdec}/G))$.

Induction hypothesis:

$|s'| = n - 1$.

$(s' \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s' \in \mathcal{L}(S_{gmdec}/G))$.

Induction step:

Let $s = s'\sigma$ so $|s'| = n-1$ and $|s| = n$. The induction step is completed in several cases.

Case 1 *:* $\sigma \in \Sigma_{uc}$

$(\sigma \in \Sigma_{uc}) \Rightarrow (\sigma \in S_{gdec}(s')) \wedge (\sigma \in S_{gmdec}(s'))$.

Because $\mathcal{L}(S_{gdec}/G)$ is controllable:

$(s' \in \mathcal{L}(S_{gdec}/G)) \wedge (s'\sigma \in \mathcal{L}(G)) \Leftrightarrow (s'\sigma \in \mathcal{L}(S_{gdec}/G))$.

Because $\mathcal{L}(S_{gmdec}/G)$ is controllable:

$(s' \in \mathcal{L}(S_{gmdec}/G)) \wedge (s'\sigma \in \mathcal{L}(G)) \Leftrightarrow (s'\sigma \in \mathcal{L}(S_{gmdec}/G))$.

So, in this case, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{gmdec}/G))$.

Case 2 *:* $\sigma \in \Sigma_{cd}$

It was shown above that $PS_i^+(s') \subseteq P_i^{-1}(P_i(s'))$.

$\Rightarrow PS_i^+(s') \cap \overline{K} \subseteq P_i^{-1}(P_i(s')) \cap \overline{K}$.

$\Rightarrow \left(PS_i^+(s') \cap \overline{K}\right) \sigma \subseteq \left(P_i^{-1}(P_i(s')) \cap \overline{K}\right) \sigma.$

$\Rightarrow \left(PS_i^+(s') \cap \overline{K}\right) \sigma \cap \mathcal{L}(G) \subseteq \left(P_i^{-1}(P_i(s')) \cap \overline{K}\right) \sigma \cap \mathcal{L}(G).$

$$\Rightarrow \left(\begin{array}{c} \left(\left(P_i^{-1}(P_i(s')) \cap \overline{K}\right) \sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \Rightarrow \\ \left(\left(PS_i^+(s') \cap \overline{K}\right) \sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \end{array} \right).$$

$(s'\sigma \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (\sigma \in S_{gdec}(s')).$

$\Rightarrow (\exists i \in I) \left[\left(\left(P_i^{-1}(P_i(s')) \cap \overline{K}\right) \sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \wedge (\sigma \in \Sigma_{cdi})\right].$

$\Rightarrow (\exists i \in I) \left[\left(\left(PS_i^+(s') \cap \overline{K}\right) \sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \wedge (\sigma \in \Sigma_{cdi})\right].$

$\Rightarrow (\sigma \in S_{gmdec}(s')).$

$\Rightarrow (s'\sigma \in \mathcal{L}(S_{gmdec}/G)).$

So, in this case, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{gmdec}/G))$.

Case 3 *:* $(\sigma \in \Sigma_{ce})$

$\forall i \in I : s' \in PS_i^+(s')$, so $\forall i \in I : s'\sigma \in PS_i^+(s')\sigma$.

Suppose $(s'\sigma \in \mathcal{L}(S_{gdec}/G))$.

$\Rightarrow s'\sigma \in \overline{K}$ by the safety condition

$\Rightarrow \forall i \in I : s'\sigma \in PS_i^+(s')\sigma \cap \overline{K}.$

$\Rightarrow \forall i \in I : PS_i^+(s')\sigma \cap \overline{K} \neq \emptyset.$

$\Rightarrow \forall i \in I : \left(PS_i^+(s')\sigma \cap \overline{K} \neq \emptyset\right).$

$\Rightarrow (\sigma \in S_{gmdec}(s')).$

$\Rightarrow (s \in \mathcal{L}(S_{gmdec}/G)).$

So, in this case, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{gmdec}/G))$.

$(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{gmdec}/G))$ holds in all cases and this completes the induction proof that given $\mathcal{L}(S_{gdec}/G)$ is safe, $\mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{gmdec}/G)$. ■

If $\mathcal{L}(S_{gdec}/G) \not\subseteq K$ and after the string s illegal behavior were about to occur, it may be the case that *gdec* would enable illegal events when *gmdec* would leave those

events disabled which is why the safety condition is required for the above proof. An intuitive corollary to the theorem just demonstrated is that if $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd}$, then the memory based control protocol will generate a safe language no smaller than that generated by the standard general decentralized control protocol of [83].

Corollary 4 $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{gmdec}/G) \subseteq \overline{K}$.

Proof: The proof of this corollary is a direct consequence of the following:

$\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{gmdec}/G) \subseteq \overline{K}$.

$\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{gdec}/G) \subseteq \overline{K}$.

$\mathcal{L}(S_{gdec}/G) \subseteq \overline{K} \Rightarrow \mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{gmdec}/G)$. ■

For all of the comparisons between $\mathcal{L}(S_{gdec}/G)$ and $\mathcal{L}(S_{gmdec}/G)$, it has not been demonstrated that $\mathcal{L}(S_{gdec}/G)$ is strictly smaller than $\mathcal{L}(S_{gmdec}/G)$. This property does not hold, in general. There may be some problem instances where *gmdec* and *gdec* generate the same language.

Corollary 5 $\left(\mathcal{L}(S_{gdec}/G) = \overline{K}\right) \Rightarrow \left(\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G)\right)$

Proof: If $\mathcal{L}(S_{gdec}/G) = \overline{K}$ then $\mathcal{L}(S_{gdec}/G)$ is safe and $(\mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{gmdec}/G))$ holds by Theorem 12.

It remains to be shown that $(\mathcal{L}(S_{gmdec}/G) \subseteq \mathcal{L}(S_{gdec}/G))$. This property is shown by contradiction. Suppose there exists $t = t'\sigma$ such that

$$\left(t' \in \mathcal{L}(S_{gmdec}/G)\right) \wedge \left(t' \in \mathcal{L}(S_{gdec}/G)\right) \wedge \left(t \in \mathcal{L}(S_{gmdec}/G)\right) \wedge \left(t \notin \mathcal{L}(S_{gdec}/G)\right).$$

This proof is broken down into three cases based on the event σ.

Case 1 *:* $\sigma \in \Sigma_{uc}$

Because σ is uncontrollable, $(\sigma \in S_{gmdec}(t')) \wedge (\sigma \in S_{gdec}(t'))$.

$(t \in \mathcal{L}(S_{gmdec}/G)) \Rightarrow (t \in \mathcal{L}(G))$, but $(t \notin \mathcal{L}(S_{gdec}/G))$ and $\mathcal{L}(S_{gdec}/G)$ is controllable. This is a contradiction, so the assumptions do not hold in this case.

Case 2 *:* $\sigma \in \Sigma_{cd}$

$(t' \in \mathcal{L}(S_{gmdec}/G)) \wedge (t \in \mathcal{L}(S_{gmdec}/G))$.

$\Rightarrow \left(\exists i \in I : \left(\left(PS_i^+(t') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \wedge (\sigma \in \Sigma_{cdi})\right)$.

It must also be true that $(t'\sigma \in \mathcal{L}(G)) \wedge \left(t'\sigma \notin \overline{K}\right)$.

However,

$\forall i \in I : \left(\left(t' \in PS_i^+(t')\right) \wedge \left(t' \in \overline{K}\right)\right) \vee (\sigma \notin \Sigma_{cdi})$.

$\Rightarrow \forall i \in I : \left(t' \in \left(PS_i^+(t') \cap \overline{K}\right)\right) \vee (\sigma \notin \Sigma_{cdi})$.

$\Rightarrow \forall i \in I : \left(t'\sigma \in \left(PS_i^+(t') \cap \overline{K}\right)\sigma\right) \vee (\sigma \notin \Sigma_{cdi})$.

$\Rightarrow \forall i \in I : \left(t'\sigma \in \left(PS_i^+(t') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G)\right) \vee (\sigma \notin \Sigma_{cdi})$.

$\Rightarrow \forall i \in I : \left(\left(PS_i^+(t') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right) \vee (\sigma \notin \Sigma_{cdi})$.

This contradicts the assumptions which do not hold in this case.

Case 3 *:* $\sigma \in \Sigma_{ce}$

$(t' \in \mathcal{L}(S_{gmdec}/G)) \wedge (t \in \mathcal{L}(S_{gmdec}/G))$.

$\Rightarrow \left(\forall i \in I : \left(PS_i^+(t')\sigma \cap \overline{K} \neq \emptyset\right) \vee (\sigma \notin \Sigma_{cei})\right)$.

$(t' \in \mathcal{L}(S_{gdec}/G)) \wedge (t \notin \mathcal{L}(S_{gdec}/G))$.

$\Rightarrow \left(\exists i \in I : \left(P_i^{-1}(P_i(t'))\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right)$.

$\forall i \in I : PS_i^+(t') \subseteq P_i^{-1}(P_i(t'))$.

$\Rightarrow \forall i \in I : PS_i^+(t')\sigma \subseteq P_i^{-1}(P_i(t'))\sigma$.

$\Rightarrow \forall i \in I : PS_i^+(t')\sigma \cap \overline{K} \subseteq P_i^{-1}(P_i(t'))\sigma \cap \overline{K}$.

$$\Rightarrow \left(\begin{array}{c} \left(\exists i \in I : \left(P_i^{-1}(P_i(t'))\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right) \Rightarrow \\ \left(\exists i \in I : \left(PS_i^+(t')\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right) \end{array} \right).$$

This contradicts the assumptions, which do not hold in this case and in all cases. Therefore $(\mathcal{L}(S_{gmdec}/G) \subseteq \mathcal{L}(S_{gdec}/G))$ which implies $(\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G))$.

■

The above corollary can also be used to show that if $\overline{K}$ is controllable and co-observable, then $\overline{K} = \mathcal{L}(S_{gmdec}/G)$.

Corollary 6 *($\overline{K}$ controllable and co-observable)*$\Rightarrow \left(\overline{K} = \mathcal{L}(S_{gmdec}/G)\right)$

Proof: From [83], ($\overline{K}$ controllable and co-observable)$\Rightarrow \left(\overline{K} = \mathcal{L}(S_{gdec}/G)\right)$

As has already seen in Corollary 5 that

$$\left(\mathcal{L}(S_{gdec}/G) = \overline{K}\right) \Rightarrow \left(\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G)\right).$$

Therefore, ($\overline{K}$ controllable and co-observable)$\Rightarrow \left(\overline{K} = \mathcal{L}(S_{gmdec}/G)\right)$. ■

As has already been shown, under some conditions $\mathcal{L}(S_{gdec}/G)$ and $\mathcal{L}(S_{gmdec}/G)$ are equal. However, what if $\left(\mathcal{L}(S_{gdec}/G) \neq \overline{K}\right)$? Will the language generated by *gmdec* always be larger than the language generated by *gdec* in this case? In short, no. The state estimate $PS_i(s)$ used for the *gmdec* control protocol might gain an advantage over $P^{-1}(P_i(s))$ only when $PS_i(s)$ can consider an anti-permissive event not feasible to be enabled globally when calculating the unobservable reach of the controlled system state.

Permissive events are disabled only when the local controller knows for sure they will lead to illegal behavior, so those events are never disabled when they lead to possibly legal behavior, and so, permissive events are never considered impossible in estimating the unobservable reach of the system under control.

A sufficient condition for when the *gmdec* control scheme performs the same as the *gdec* controller is shown below. The proofs of the lemmas and theorem are

based on the intuition that permissive events could be considered possible by the state estimate even though they may lead to illegal behavior. Therefore, for the *gmdec* control protocol, the state estimate $PS_i(s)$ can only gain knowledge that $P_i^{-1}(P_i(s))$ does not have in the context of the control protocol only when there are unobservable anti-permissive events that are privately controlled and disabled so that those events are known to be disabled globally. This prompts the lemmas seen below that show $\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right) \Rightarrow \left(P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i^+(s) \cap \overline{K}\right) \wedge \left(P_i^{-1}(P_i(s))\sigma \cap \overline{K} = PS_i^+(s)\sigma \cap \overline{K}\right)$. The intersection of the state estimates with $\overline{K}$ demonstrates the "context" of the *gmdec* control protocol, i.e., it is used to prevent the system transitioning from legal states to illegal states and hence it is only concerned with behavior that occurs when the system is in a legal state.

Lemma 3 *Given two sets of strings,* $A, B : AB \cap \overline{K}B \cap \overline{K} = AB \cap \overline{K}$.

Proof: This proof is broken down into two cases. First it is shown that

$$\left(\left(AB \cap \overline{K}B \cap \overline{K}\right) \subseteq \left(AB \cap \overline{K}\right)\right).$$

and then it is shown that

$$\left(\left(AB \cap \overline{K}\right) \subseteq \left(AB \cap \overline{K}B \cap \overline{K}\right)\right).$$

Case 1 : $\left(AB \cap \overline{K}B \cap \overline{K}\right) \subseteq \left(AB \cap \overline{K}\right)$

This case is trivial and should be obvious.

Case 2 : $\left(\left(AB \cap \overline{K}\right) \subseteq \left(AB \cap \overline{K}B \cap \overline{K}\right)\right)$

This case is proved by contradiction. There exists a string $t = t's$ such that $(t' \in A) \wedge (s \in B)$. Suppose:

$\big(t \in AB \cap \overline{K}\big) \wedge \big(t \notin AB \cap \overline{K}B \cap \overline{K}\big)$.

$\Rightarrow \big(t \in \overline{K}\big) \wedge \big(t \notin \overline{K}B\big)$.

$\Rightarrow \big(t' \in \overline{K}\big) \wedge \big(t \notin \overline{K}B\big)$.

$\Rightarrow \big(t's \in \overline{K}B\big) \wedge \big(t \notin \overline{K}B\big)$.

$\Rightarrow \big(t's \in \overline{K}B\big) \wedge \big(t's \notin \overline{K}B\big)$.

This is a contradiction and therefore $\big(\big(AB \cap \overline{K}\big) \subseteq \big(AB \cap \overline{K}B \cap \overline{K}\big)\big)$. This completes both cases of the proof that $AB \cap \overline{K}B \cap \overline{K} = AB \cap \overline{K}$. ■

Lemma 4 $\left(\begin{array}{c} \big(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\big) \Rightarrow \\ \big(P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i(s) \cap \overline{K}\big) \end{array} \right)$

Proof: This proof is broken down into two parts. First it is shown that

$$\big(P_i^{-1}(P_i(s)) \cap \overline{K} \subseteq PS_i(s) \cap \overline{K}\big).$$

and then it is shown that

$$\big(P_i^{-1}(P_i(s)) \cap \overline{K} \supseteq PS_i(s) \cap \overline{K}\big).$$

Case 1 *: To show* $\big(PS_i(s) \cap \overline{K} \subseteq P_i^{-1}(P_i(s)) \cap \overline{K}\big)$:

It has already been shown that $PS_i(s) \subseteq P_i^{-1}(P_i(s))$.

$PS_i(s) \subseteq P_i^{-1}(P_i(s))$.

$\Rightarrow PS_i(s) \cap \overline{K} \subseteq P_i^{-1}(P_i(s)) \cap \overline{K}$.

so, this case holds.

Case 2 *: To show* $\big(P_i^{-1}(P_i(s)) \cap \overline{K} \subseteq PS_i(s) \cap \overline{K}\big)$:

To start off with some preliminary simplification:

$$\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) = \left(\begin{array}{c} \left\{\sigma \in \Sigma_{cdi} : \left(PS_i^+(s) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ \left\{\sigma \in \Sigma_{cei} : PS_i^+(s)\sigma \cap \overline{K} \neq \emptyset\right\} \cup \\ \Sigma_{uc} \cup \Sigma_{cd}^{-i} \cup \Sigma_{ce} \setminus \Sigma_{cei} \end{array} \right).$$

For the sake of simplicity, the following functions are defined to represent a supervisor's local anti-permissive control action and local permissive control action respectively.

Definition 13

$$\gamma_i^d(s) = \left\{\sigma \in \Sigma_{cdi} : \left(PS_i^+(s) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \tag{5.6}$$

Definition 14

$$\gamma_i^e(s) = \left\{\sigma \in \Sigma_{cei} : PS_i^+(s)\sigma \cap \overline{K} \neq \emptyset\right\} \tag{5.7}$$

$\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}$ can now be rewritten in the following manner:

$\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi} = \left(\gamma_i^e(s) \cup \gamma_i^d(s) \cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei} \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}.$

$= \left(\left(\gamma_i^e(s) \cup \gamma_i^d(s) \cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}\right) \cap \Sigma_{uoi}\right) \cup \left(\Sigma_{cd}^{-i} \cap \Sigma_{uoi}\right).$

$= \left(\left(\gamma_i^e(s) \cup \gamma_i^d(s) \cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei}\right) \cap \Sigma_{uoi}\right) \cup \left(\Sigma_{cd} \cap \Sigma_{uoi}\right)$ by the hypothesis.

$= \left(\gamma_i^e(s) \cup \gamma_i^d(s) \cup \Sigma_{uc} \cup \Sigma_{cd} \cup \Sigma_{ce} \setminus \Sigma_{cei}\right) \cap \Sigma_{uoi}.$

$= \left(\gamma_i^e(s) \cup \Sigma_{uc} \cup \left(\gamma_i^d(s) \cup \Sigma_{cd}\right) \cup \Sigma_{ce} \setminus \Sigma_{cei}\right) \cap \Sigma_{uoi}.$

$= \left(\gamma_i^e(s) \cup \Sigma_{uc} \cup \Sigma_{cd} \cup \Sigma_{ce} \setminus \Sigma_{cei}\right) \cap \Sigma_{uoi}.$

$= \left(\gamma_i^e(s) \cup \left(\Sigma_{uc} \cup \Sigma_{cd} \cup \Sigma_{ce}\right) \setminus \Sigma_{cei}\right) \cap \Sigma_{uoi}.$

$= \left(\gamma_i^e(s) \cup \Sigma \setminus \Sigma_{cei}\right) \cap \Sigma_{uoi}.$

$= \left(\Sigma \setminus \left(\Sigma_{cei} \setminus \gamma_i^e(s)\right)\right) \cap \Sigma_{uoi}.$

Now, $PS_i(\varepsilon) \cap \overline{K}$ and $PS_i(s) \cap \overline{K}$ can be rewritten as follows:

$PS_i(\varepsilon) \cap \overline{K} = \left[\left(\Sigma \setminus \left(\Sigma_{cei} \setminus \gamma_i^e(s)\right)\right) \cap \Sigma_{uoi}\right]^* \cap \overline{K}.$

$PS_i(s) \cap \overline{K} = \left(PS_i(s')P_i(\sigma)\left[\left(\Sigma \setminus \left(\Sigma_{cei} \setminus \gamma_i^e(s)\right)\right) \cap \Sigma_{uoi}\right]^*\right) \cap \overline{K}.$

Now, to demonstrate $\left(P_i^{-1}(P_i(s)) \cap \overline{K} \subseteq PS_i(s) \cap \overline{K}\right)$, a proof based on induction on the string s is used to show $\left(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(s) \cap \overline{K}\right)$.

Base:

$|s| = 0$, or $s = \varepsilon$.

The base case $\left(t \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(\varepsilon) \cap \overline{K}\right)$ is proved by induction on the length of string t.

Base (of the Base):

$|t| = 0$, or $t = \varepsilon$.

$PS_i(\varepsilon) \cap \overline{K} = [(\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(\varepsilon))) \cap \Sigma_{uoi}]^* \cap \overline{K}$.

$P_i^{-1}(P_i(\varepsilon)) \cap \overline{K} = \Sigma_{uoi}^* \cap \overline{K}$.

Both $PS_i(\varepsilon) \cap \overline{K}$ and $P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}$ are prefix closed and without loss of generality, both languages are assumed to be nonempty. This implies $\left(\varepsilon \in PS_i(\varepsilon) \cap \overline{K}\right) \wedge \left(\varepsilon \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right)$. Therefore $\left(\varepsilon \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \Rightarrow \left(\varepsilon \in PS_i(\varepsilon) \cap \overline{K}\right)$ and the base case of the base holds.

Induction hypothesis (of the Base):

$|t'| = m - 1$.

$\left(t' \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \Rightarrow \left(t' \in PS_i(\varepsilon) \cap \overline{K}\right)$.

Induction step (of the Base):

Let $t = t'\sigma$ so $|t'| = m - 1$ and $|t| = m$.

The induction step (of the Base) is proved by contradiction.

Assume $\left(t \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \wedge \left(t \notin PS_i(\varepsilon) \cap \overline{K}\right)$.

$\Rightarrow \left(t \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \wedge \left(t \in \overline{K}\right) \wedge \left(t \notin PS_i(\varepsilon) \cap \overline{K}\right)$.

$\Rightarrow \left(t \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \wedge \left(t' \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \wedge (t \notin PS_i(\varepsilon))$.

$\Rightarrow \left(t \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \wedge (t \notin PS_i(\varepsilon)) \wedge \left(t' \in PS_i(\varepsilon) \cap \overline{K}\right)$.

$$\Rightarrow \left(\begin{array}{c} \left(t \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \wedge (t \notin PS_i(\varepsilon)) \wedge \left(t' \in PS_i(\varepsilon) \cap \overline{K}\right) \wedge \\ (\sigma \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(\varepsilon))) \cap \Sigma_{uoi}) \wedge (\sigma \in \Sigma_{uoi}) \end{array} \right)$$

because σ must considered impossible in the unobservable reach of the system by the initial assumptions.

$$\Rightarrow \left(\begin{array}{c} \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge (t \notin PS_i(\varepsilon)) \wedge (t' \in PS_i(\varepsilon)) \wedge \\ (\sigma \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(\varepsilon))) \cap \Sigma_{uoi}) \wedge (\sigma \in \Sigma_{uoi}) \end{array} \right).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge (t \notin PS_i(\varepsilon)) \wedge (t' \in PS_i(\varepsilon)) \wedge (\sigma \in \Sigma_{cei} \setminus \gamma_i^e(\varepsilon)).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge (t \notin PS_i(\varepsilon)) \wedge (\sigma \in \Sigma_{cei} \setminus \gamma_i^e(\varepsilon)).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge (\sigma \in \Sigma_{cei} \text{ and not enabled by } i).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge \left(PS_i^+(\varepsilon)\sigma \cap \overline{K} = \emptyset\right).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge \left(\left(\Sigma_{uoi}^* \cap \mathcal{L}(G)\right) \sigma \cap \overline{K} = \emptyset\right).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge \left(\Sigma_{uoi}^* \sigma \cap \mathcal{L}(G)\sigma \cap \overline{K} = \emptyset\right).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge \left(\Sigma_{uoi}^* \sigma \cap \mathcal{L}(G)\sigma \cap \mathcal{L}(G) \cap \overline{K} = \emptyset\right).$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge \left(\Sigma_{uoi}^* \sigma \cap \mathcal{L}(G) \cap \overline{K} = \emptyset\right) \text{ by Lemma 3.}$$

$$\Rightarrow \left(\Sigma_{uoi}^* \sigma \cap \overline{K} \neq \emptyset\right) \wedge \left(\Sigma_{uoi}^* \sigma \cap \overline{K} = \emptyset\right).$$

This is a contradiction, so the assumption does not hold, which completes the proof by contradiction. Therefore,

$$\left(t \in P_i^{-1}(P_i(\varepsilon)) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(\varepsilon) \cap \overline{K}\right)$$

which completes the proof by induction for the base case of

$$\left(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(s) \cap \overline{K}\right).$$

Induction hypothesis for the proof of:

$\left(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(s) \cap \overline{K}\right)$:

$|s'| = n - 1$.

$\left(t \in P_i^{-1}(P_i(s')) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(s') \cap \overline{K}\right)$.

Induction step for the proof of:

$\left(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(s) \cap \overline{K}\right)$:

Let $s = s'\sigma$ such that $|s'| = n-1$ and $|s| = n$.

$P_i^{-1}(P_i(s)) \cap \overline{K} = P_i^{-1}(P_i(s'))P_i(\sigma)\Sigma_{uoi}^* \cap \overline{K}$.

$PS_i(s) \cap \overline{K} = (PS_i(s')P_i(\sigma)\left[(\gamma_i^e(s) \cup \Sigma \setminus \Sigma_{cei}) \cap \Sigma_{uoi}\right]^*) \cap \overline{K}$.

The induction step needs to be proved for two cases, $P_i(\sigma) = \varepsilon$, and $P_i(\sigma) = \sigma$.

Case 3 *:* $P_i(\sigma) = \varepsilon$

This case is trivial. If $P_i(\sigma) = \varepsilon$, then $P_i^{-1}(P_i(s)) \cap \overline{K} = P_i^{-1}(P_i(s')) \cap \overline{K}$ and $PS_i(s) \cap \overline{K} = PS_i(s') \cap \overline{K}$. By simple substitution with the induction hypothesis, $\left(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Rightarrow \left(t \in PS_i(s) \cap \overline{K}\right)$.

Case 4 *:* $P_i(\sigma) = \sigma$

Let $u = u'\sigma$, $v = v'\alpha$ and $t = uv$ such that $P_i(s') = P_i(u')$ and $P_i(v) = \varepsilon$. A u and v exist for all t such that $t \in P_i^{-1}(P_i(s)) \cap \overline{K}$.

$t \in P_i^{-1}(P_i(s)) \cap \overline{K}$.

$\Rightarrow u'\sigma v \in P_i^{-1}(P_i(s'))\sigma\Sigma_{uoi}^* \cap \overline{K}$.

$t \in PS_i(s) \cap \overline{K}$.

$\Rightarrow \left(u'\sigma v \in PS_i(s')\sigma\left[\left(\gamma_i(s) \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \overline{K}\right)$.

This case is proved by induction on the length of the string v.

Base:

$|v| = 0$, or $v = \varepsilon$.

$u'\sigma v = u'\sigma$.

$\left(u'\sigma \in P_i^{-1}(P_i(s'))\sigma \cap \overline{K}\right)$.

$\Rightarrow \left(u'\sigma \in P_i^{-1}(P_i(s'))\sigma \cap \overline{K}\right) \wedge \left(u'\sigma \in \overline{K}\right)$.

$\Rightarrow \left(u'\sigma \in P_i^{-1}(P_i(s'))\sigma \cap \overline{K}\right) \wedge \left(u' \in \overline{K}\right)$.

$\Rightarrow \left(u'\sigma \in P_i^{-1}(P_i(s'))\sigma \cap \overline{K}\right) \wedge \left(u'\sigma \in \overline{K}\sigma\right)$.

$\Rightarrow \left(u'\sigma \in P_i^{-1}(P_i(s'))\sigma \cap \overline{K}\sigma \cap \overline{K}\right)$.

$\Rightarrow \left(u'\sigma \in P_i^{-1}(P_i(s'))\sigma \cap \overline{K}\sigma\right)$.

$\Rightarrow \left(u'\sigma \in \left(P_i^{-1}(P_i(s')) \cap \overline{K}\right)\sigma\right)$.

$\Rightarrow \left(u' \in \left(P_i^{-1}(P_i(s')) \cap \overline{K}\right)\right)$.

$\Rightarrow \left(u' \in \left(PS_i(s') \cap \overline{K}\right)\right)$ because $|u'| < n$ and the (original) induction hypothesis.

$\Rightarrow \left(u'\sigma \in \left(PS_i(s') \cap \overline{K}\right)\sigma\right)$.

$\Rightarrow \left(u'\sigma \in PS_i(s')\sigma \cap \overline{K}\sigma\right)$.

$\Rightarrow \left(u'\sigma \in PS_i(s')\sigma \cap \overline{K}\sigma \cap \overline{K}\right)$.

$\Rightarrow \left(u'\sigma \in PS_i(s')\sigma \cap \overline{K}\right)$ by the Lemma 3.

So the base case of the (original) induction step holds.

Induction Hypothesis (of the induction step):

$|v'| = q - 1$.

$\left(u'\sigma v' \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Rightarrow \left(u'\sigma v' \in PS_i(s) \cap \overline{K}\right)$.

The induction step (of the Base) is proved by contradiction.

Let $v = v'\alpha$ so $|v'| = q - 1$ and $|v| = q$.

Assume $\left(u'\sigma v \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \wedge \left(u'\sigma v \notin PS_i(s) \cap \overline{K}\right)$.

$\Rightarrow \left(u'\sigma v \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \wedge \left(u'\sigma v \in \overline{K}\right) \wedge (u'\sigma v \notin PS_i(s))$.

$$\Rightarrow \left(\begin{array}{c} \left(u'\sigma v \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \wedge (u'\sigma v \notin PS_i(s)) \wedge \\ \left(u'\sigma v' \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \end{array} \right).$$

$$\Rightarrow \left(\begin{array}{c} \left(u'\sigma v \in P_i^{-1}(P_i(s)) \cap \overline{K}\right) \wedge (u'\sigma v \notin PS_i(s)) \wedge \\ \left(u'\sigma v' \in PS_i(s) \cap \overline{K}\right) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right)$$ because σ must considered possible in the unobservable reach of the system by the initial assumptions.

$$\Rightarrow \left(\begin{array}{c} \left(P_i^{-1}(P_i(s))\alpha \cap \overline{K} \neq \emptyset\right) \wedge (u'\sigma v \notin PS_i(s)) \wedge \\ (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right).$$

$$\Rightarrow \left(\begin{array}{c} \left(P_i^{-1}(P_i(s'))\sigma\Sigma_{uoi}^*\alpha \cap \overline{K} \neq \emptyset\right) \wedge (u'\sigma v \notin PS_i(s)) \wedge \\ (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right).$$

$$\Rightarrow \left(\begin{array}{c} \left(\exists w \in P_i^{-1}(P_i(s'))\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\right) \wedge (u'\sigma v \notin PS_i(s)) \wedge \\ (u'\sigma v' \in PS_i(s)) \\ \wedge (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right),$$

$$\Rightarrow \left(\begin{array}{c} \left(\exists w \in P_i^{-1}(P_i(s'))\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\right) \wedge \\ (u'\sigma v \notin PS_i(s)) \wedge (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right) \text{ by the Lemma 3.}$$

$$\Rightarrow \left(\begin{array}{c} \left(\exists w \in \left(P_i^{-1}(P_i(s')) \cap \overline{K}\right)\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\right) \wedge \\ (u'\sigma v \notin PS_i(s)) \wedge (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right).$$

$$\Rightarrow \left(\begin{array}{c} \left(\exists w \in \left(PS_i(s')) \cap \overline{K}\right)\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\right) \wedge \\ (u'\sigma v \notin PS_i(s)) \wedge (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right).$$

$$\Rightarrow \left(\begin{array}{c} \left(\exists w \in PS_i(s'))\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\right) \wedge \\ (u'\sigma v \notin PS_i(s)) \wedge (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right) \text{ by the induction hypothesis.}$$

$$\Rightarrow \left(\begin{array}{c} \left(\exists w \in PS_i(s'))\sigma\Sigma_{uoi}^*\alpha \cap \overline{K}\right) \wedge \\ (u'\sigma v \notin PS_i(s)) \wedge (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma_i^e(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right) \text{ by the Lemma 3.}$$

$$\Rightarrow \left(\begin{array}{c} \big(PS_i(s'))\sigma\Sigma^*_{uoi}\alpha \cap \overline{K} \neq \emptyset\big) \wedge \\ (u'\sigma v \notin PS_i(s)) \wedge (u'\sigma v' \in PS_i(s)) \wedge \\ (\alpha \notin (\Sigma \setminus (\Sigma_{cei} \setminus \gamma^e_i(s))) \cap \Sigma_{uoi}) \wedge (\alpha \in \Sigma_{uoi}) \end{array} \right).$$

$$\Rightarrow \left(\begin{array}{c} \big(PS_i(s'))\sigma\Sigma^*_{uoi}\alpha \cap \overline{K} \neq \emptyset\big) \wedge (u'\sigma v \notin PS_i(s)) \wedge \\ (u'\sigma v' \in PS_i(s)) \wedge (\alpha \in \Sigma_{cei} \backslash \gamma^e_i(s)) \end{array} \right).$$

$$\Rightarrow \big(PS_i(s'))\sigma\Sigma^*_{uoi}\alpha \cap \overline{K} \neq \emptyset\big) \wedge (u'\sigma v \notin PS_i(s)) \wedge (\alpha \in \Sigma_{cei} \backslash \gamma^e_i(s)).$$

$$\Rightarrow \big(PS_i(s'))\sigma\Sigma^*_{uoi}\alpha \cap \overline{K} \neq \emptyset\big) \wedge (\alpha \in \Sigma_{cei} \text{ and not enabled by } i).$$

$$\Rightarrow \big(PS_i(s'))\sigma\Sigma^*_{uoi}\alpha \cap \overline{K} \neq \emptyset\big) \wedge \big(PS^+_i(s)\alpha \cap \overline{K} = \emptyset\big).$$

$$\Rightarrow \big(PS_i(s'))\sigma\Sigma^*_{uoi}\alpha \cap \overline{K}\alpha \cap \overline{K} \neq \emptyset\big) \wedge \big(PS^+_i(s)\alpha \cap \overline{K} = \emptyset\big) \text{ by the Lemma 3.}$$

$$\Rightarrow \big(PS_i(s'))\sigma\Sigma^*_{uoi}\alpha \cap \mathcal{L}(G)\alpha \cap \overline{K} \neq \emptyset\big) \wedge \big(PS^+_i(s)\alpha \cap \overline{K} = \emptyset\big).$$

$$\Rightarrow \big((PS_i(s'))\sigma\Sigma^*_{uoi} \cap \mathcal{L}(G))\,\alpha \cap \overline{K} \neq \emptyset\big) \wedge \big(PS^+_i(s)\alpha \cap \overline{K} = \emptyset\big).$$

$$\Rightarrow \big(PS^+_i(s)\alpha \cap \overline{K} \neq \emptyset\big) \wedge \big(PS^+_i(s)\alpha \cap \overline{K} = \emptyset\big).$$

This is a contradiction, so the assumption does not hold, which completes the proof by contradiction. Therefore, $\big(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\big) \Rightarrow \big(t \in PS_i(s) \cap \overline{K}\big)$ which completes the proof by induction for the induction hypothesis of

$$\big(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\big) \Rightarrow \big(t \in PS_i(s) \cap \overline{K}\big).$$

This completes the induction proof that

$$\Big(\big(t \in P_i^{-1}(P_i(s)) \cap \overline{K}\big) \Rightarrow \big(t \in PS_i(s) \cap \overline{K}\big)\Big).$$

It has now been shown that

$$\big(P_i^{-1}(P_i(s)) \cap \overline{K} \subseteq PS_i(s) \cap \overline{K}\big) \tag{5.8}$$

$$\big(P_i^{-1}(P_i(s)) \cap \overline{K} \supseteq PS_i(s) \cap \overline{K}\big). \tag{5.9}$$

This holds if and only if

$$P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i(s) \cap \overline{K} \tag{5.10}$$

which completes the proof of this lemma. ■

Lemma 5 $\left(\begin{array}{c} \left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right) \Rightarrow \\ \left(P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i^+(s) \cap \overline{K}\right) \end{array} \right)$

Proof: It has already been shown that:

$\left(\begin{array}{c} \left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right) \Rightarrow \\ \left(P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i(s) \cap \overline{K}\right) \end{array} \right)$.

$\Rightarrow \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Sigma_{uoi}^* = \left(PS_i(s) \cap \overline{K}\right) \Sigma_{uoi}^*$.

$\Rightarrow P_i^{-1}(P_i(s))\Sigma_{uoi}^* \cap \overline{K}\Sigma_{uoi}^* = PS_i(s)\Sigma_{uoi}^* \cap \overline{K}\Sigma_{uoi}^*$.

$\Rightarrow P_i^{-1}(P_i(s))\Sigma_{uoi}^* \cap \overline{K}\Sigma_{uoi}^* \cap \overline{K} = PS_i(s)\Sigma_{uoi}^* \cap \overline{K}\Sigma_{uoi}^* \cap \overline{K}$.

$\Rightarrow P_i^{-1}(P_i(s))\Sigma_{uoi}^* \cap \overline{K} = PS_i(s)\Sigma_{uoi}^* \cap \overline{K}$ by the Lemma 3.

$\Rightarrow P_i^{-1}(P_i(s))\Sigma_{uoi}^* \cap \overline{K} = PS_i(s)\Sigma_{uoi}^* \cap \mathcal{L}(G) \cap \overline{K}$.

$\Rightarrow P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i^+(s) \cap \overline{K}$.

This completes the proof of this lemma. ■

Lemma 6 $\left(\begin{array}{c} \left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right) \Rightarrow \\ \left(P_i^{-1}(P_i(s))\sigma \cap \overline{K} = PS_i^+(s)\sigma \cap \overline{K}\right) \end{array} \right)$

Proof: This proof is broken down into two cases.

Case 1 *:* $P_i(s) = \varepsilon$

$P_i^{-1}(P_i(s))\sigma \cap \overline{K} = \Sigma_{uoi}^*\sigma \cap \overline{K}$.

$PS_i^+(s)\sigma \cap \overline{K} = \left(\Sigma_{uoi}^* \cap \mathcal{L}(G)\right)\sigma \cap \overline{K}$.

$PS_i^+(s)\sigma \cap \overline{K} = \left(\Sigma_{uoi}^* \cap \mathcal{L}(G)\right)\sigma \cap \mathcal{L}(G) \cap \overline{K}$.

$PS_i^+(s)\sigma \cap \overline{K} = \Sigma_{uoi}^*\sigma \cap \mathcal{L}(G)\sigma \cap \mathcal{L}(G) \cap \overline{K}$.

$PS_i^+(s)\sigma \cap \overline{K} = \Sigma_{uoi}^*\sigma \cap \mathcal{L}(G) \cap \overline{K}$ by the Lemma 3.

$PS_i^+(s)\sigma \cap \overline{K} = \Sigma_{uoi}^*\sigma \cap \overline{K}$.

So in this case, $\left(P_i^{-1}(P_i(s))\sigma \cap \overline{K} = PS_i^+(s)\sigma \cap \overline{K}\right)$.

Case 2 *:* $P_i(s) \neq \varepsilon$

It has already been shown that:

$$\left(\begin{array}{c} \left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right) \Rightarrow \\ \left(P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i(s) \cap \overline{K}\right) \end{array} \right).$$

$\Rightarrow \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right) \Sigma_{uoi}^* \sigma = \left(PS_i(s) \cap \overline{K}\right) \Sigma_{uoi}^* \sigma.$

$\Rightarrow P_i^{-1}(P_i(s)) \Sigma_{uoi}^* \sigma \cap \overline{K} \Sigma_{uoi}^* \sigma = PS_i(s) \Sigma_{uoi}^* \sigma \cap \overline{K} \Sigma_{uoi}^* \sigma.$

$\Rightarrow P_i^{-1}(P_i(s)) \Sigma_{uoi}^* \sigma \cap \overline{K} \Sigma_{uoi}^* \sigma \cap \overline{K} = PS_i(s) \Sigma_{uoi}^* \sigma \cap \overline{K} \Sigma_{uoi}^* \sigma \cap \overline{K}.$

$\Rightarrow P_i^{-1}(P_i(s)) \Sigma_{uoi}^* \sigma \cap \overline{K} = PS_i(s) \Sigma_{uoi}^* \sigma \cap \overline{K}$ by the Lemma 3.

$\Rightarrow P_i^{-1}(P_i(s)) \Sigma_{uoi}^* \sigma \cap \overline{K} = PS_i(s) \Sigma_{uoi}^* \sigma \cap \mathcal{L}(G) \cap \overline{K}.$

$\Rightarrow P_i^{-1}(P_i(s)) \Sigma_{uoi}^* \sigma \cap \overline{K} = PS_i(s) \Sigma_{uoi}^* \sigma \cap \mathcal{L}(G) \sigma \cap \mathcal{L}(G) \cap \overline{K}$ by the Lemma 3.

$\Rightarrow P_i^{-1}(P_i(s)) \Sigma_{uoi}^* \sigma \cap \overline{K} = \left(PS_i(s) \Sigma_{uoi}^* \cap \mathcal{L}(G)\right) \sigma \cap \mathcal{L}(G) \cap \overline{K}.$

$\Rightarrow P_i^{-1}(P_i(s)) \Sigma_{uoi}^* \sigma \cap \overline{K} = \left(PS_i(s) \Sigma_{uoi}^* \cap \mathcal{L}(G)\right) \sigma \cap \overline{K}.$

$\Rightarrow P_i^{-1}(P_i(s)) \sigma \cap \overline{K} = PS_i^+(s) \sigma \cap \overline{K}.$

This completes the proof of this lemma. ■

With the above lemmas, the theorem below follows from simple manipulation of the definitions.

Theorem 13 $$\left(\begin{array}{c} \left[(\forall i \in I) \left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right] \Rightarrow \\ \mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G) \end{array} \right).$$

Proof: It has already been shown that:

$$\left(\begin{array}{c} \left[(\forall i \in I) \left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right] \Rightarrow \\ \left(P_i^{-1}(P_i(s)) \cap \overline{K} = PS_i^+(s) \cap \overline{K}\right) \end{array} \right).$$

and $$\left(\begin{array}{c} \left[(\forall i \in I) \left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right] \Rightarrow \\ \left(P_i^{-1}(P_i(s)) \sigma \cap \overline{K} = PS_i^+(s) \sigma \cap \overline{K}\right) \end{array} \right).$$

By definition:

$$\gamma_i^{gdec}(s) = \left(\begin{array}{c} \left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ \left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s))\sigma \cap \overline{K} \neq \emptyset\right\} \cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei} \end{array} \right).$$

$$\gamma_i^{gmdec}(s) = \left(\begin{array}{c} \left\{\sigma \in \Sigma_{cdi} : \left(PS_i^{+}(s) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ \left\{\sigma \in \Sigma_{cei} : PS_i^{+}(s)\sigma \cap \overline{K} \neq \emptyset\right\} \cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei} \end{array} \right).$$

$$= \left(\begin{array}{c} \left\{\sigma \in \Sigma_{cdi} : \left(P_i^{-1}(P_i(s)) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\} \cup \\ \left\{\sigma \in \Sigma_{cei} : P_i^{-1}(P_i(s))\sigma \cap \overline{K} \neq \emptyset\right\} \cup \Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei} \end{array} \right) \text{ by substitution.}$$

$$= \gamma_i^{gdec}(s).$$

So, $\gamma_i^{gdec}(s) = \gamma_i^{gdec}(s)$ for all $i \in I$.

$\Rightarrow S_{gdec}(s) = S_{gmdec}(s)$.

$\Rightarrow \mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G)$.

This completes the proof. ■

It has previously been shown that $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd}^2 \subseteq \Sigma_{cd}^1 \not\Rightarrow \mathcal{L}(S_{gmdec}^1/G) \subseteq \mathcal{L}(S_{gmdec}^2/G)$, but if $(\mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G))$ holds, then the implication will also hold. The previous theorem now prompts the following corollary.

Corollary 7 *Given* $\left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right]$ *and* $\Sigma_{cd}^2 \subseteq \Sigma_{cd}^1$, $\mathcal{L}(S_{gmdec}^1/G) \subseteq \overline{K}$, $\mathcal{L}(S_{gmdec}^2/G) \subseteq \overline{K}$. *Therefore* $\mathcal{L}(S_{gmdec}^1/G) \subseteq \mathcal{L}(S_{gmdec}^2/G)$

5.6 VLP-GM Algorithm

An interesting deficiency of the *gdec* and *gmdec* control protocols discussed above is that the local control actions are not always maximal with respect to a controller's state estimate. When computing a control action, *gmdec* looks at the uncontrolled and unobservable reach of the system from the current set of estimated states. It might occur that some of the events disabled by *gmdec* could be legally enabled. This issue is illustrated by the example below.

Example 2 *Consider the uncontrolled system G and desired controlled system H seen in Figure 5.6 such that $\mathcal{L}(H) = \overline{K}$.*

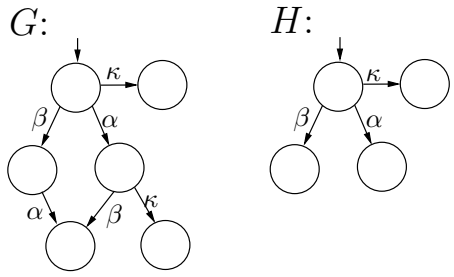

Figure 5.6: Uncontrolled system G and desired system H for Example 2.

The system has two controllers and the following properties:

$\Sigma_o = \emptyset$.

$\Sigma_{cd1} = \emptyset$, $\Sigma_{ce1} = \{\beta\}$, $\Sigma_{cd2} = \{\alpha, \kappa\}$, $\Sigma_{ce2} = \emptyset$.

The gmdec controller introduced above would have local control actions $\gamma_1(\varepsilon) = \{\beta\}$ and $\gamma_2(\varepsilon) = \{\beta\}$. The global control action would be $S_{gmdec}(\varepsilon) = \{\beta\}$.

However, there is a valid control action the second controller might take that is strictly larger than $\gamma_2(\varepsilon) = \{\beta\}$. Suppose controller 2 were to take the control action $\gamma_2(\varepsilon) = \{\beta, \kappa\}$.

Controller 2 knows that event α is globally disabled because only controller 2 can enable α. Therefore, controller 2 knows κ is a valid event to enable. This control action will result in a generated language that is larger than the language generated by gmdec.

The resulting controlled systems can be seen in Figure 5.7. The controlled system with the modified control action for controller 2 is denoted by S_{mod}/G.

Obviously, in this example, the gmdec control protocol does not generate maximal local control actions for its given state estimate.

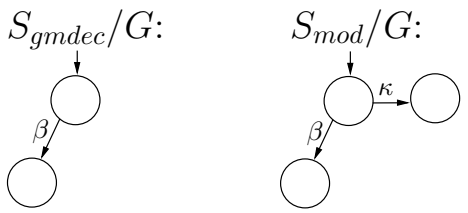

Figure 5.7: S_{gmdec}/G and S_{gdec}/G for Example 2.

Note that the *gmdec* and *gdec* control protocols locally enable all of their events at the same instance, but what if a control protocol were to locally enable its events one-by-one? It has already been seen in the example that the *gmdec* control protocol might not enable all events locally possible. Might a new control protocol that enables events one-by-one be able to generate local control actions that are locally maximal?

For this new scheme, a local controller could initially enable all events that are required to be enabled no matter the situation: uncontrollable events and permissive events the local controller cannot control. A local controller could then test to enable locally controllable events sequentially. The controller would select an event to test for enabling and if the event is admissible as an enabled event, that event would be added to the control action and controller would move on to the next event. As events are tested to be enabled, the unobservable controlled reach is likewise expanded to include the behavior of the newly enabled events. To determine the order in which events are tested to be enabled, the controllers could use an ordered list of controllable events that might reflect the relative desirability of enabling those events. Notice that this briefly outlined the control algorithm is a greedy algorithm; as events are enabled locally, they are not disabled until the next control action is calculated.

A similar approach to developing a maximal control protocol for permissive centralized control under partial observation was developed by Ben Hadj-Alouane, et.al. [10] and called VLP-PO for "Variable Lookahead Protocols under Partial Observation". VLP-PO uses an iterative algorithm to generate maximal control protocols that enable as many events as possible at each control step. VLP-PO also takes an event ordering into account when determining the enabled events by attempting to enable events one-by-one in a predetermined order. Furthermore VLP-PO is a "greedy" control protocol in that once an event is enabled, it is never disabled until a new control action is calculated.

In this section of the chapter, the VLP-PO algorithm is extended to the general decentralized architecture for a new algorithm called VLP-GM meaning "Variable Lookahead Protocols for General decentralized Memory-based control". VLP-GM calculates a supervisor's local control action based on a state estimate similar to $PS_i(s)$ introduced in section 3 above.

VLP-GM is an iterative greedy procedure like VLP-PO in that events are enabled one-by-one, but once an event is enabled, it is never disabled until another event is observed and a new control action is calculated. VLP-GM necessarily accounts for the possibility that the i^{th} controller might not be the only controller with authority over some the controllable events similar to *gmdec* above. When a controller i uses VLP-GM to calculate local control actions, it considers the possibility that other local controllers may enable anti-permissive events that the i^{th} controller would disable. Furthermore, when a controller attempts to enable an event, the controller needs to be sure that the unobservable behavior of all events previously enabled would still be valid after that new event is enabled.

Because of the prioritization of controllable events, when an event is enabled, it could cause some already enabled to become illegal. This point is illustrated in the following example.

Example 3 *For this example, consider the branch of the uncontrolled system seen in Figure 5.8.*

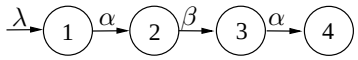

Figure 5.8: A branch of the automaton representing the system in Example 3.

Suppose λ is observable to controller i, but α and β are not. Suppose that after λ occurs, controller i knows that the system could only be in system state 1. Suppose further that all states are legal except for state 4 and that controller i places a high priority on enabling the event α. Let β be a permissive event and α be an anti-permissive event. Under the gmdec control protocol, controller i's action would be to enable β and disable α, but this would not be the case for VLP-GM. Because controller i places a high priority on α, α would be tested to be enabled first.

Initially, β would not be enabled, so it would be valid for i to enable α. After α is enabled, the controller would then test β. In itself, β would be a valid event to enable because it leads to the legal state 3, but by enabling β, α could then lead to an illegal state. Therefore, β could not be enabled in this situation using VLP-GM, but α could be.

As with the previously discussed *gmdec* control protocol, for VLP-GM introduced below, PS_i is an iteratively updated state estimate and represents all states that the local controller estimates the system could be in after the control action is calcu-

lated. On initialization, before any control action is taken, PS_i should be set to $\{\varepsilon\}$. As before, it is assumed that the controllers operate instantaneously with respect to system operation so the controllers can be calculate control actions before any unobservable events occur. The set NS_i represents all strings that could have occurred if there were no unobservable events after the last observable event. Notice that for initialization purposes, the last event to occur could be considered the empty event, ε.

After initialization, the VLP-GM algorithm for i needs to be run only once after events observable to i occur because it is assumed that the local controllers have no knowledge of the observations or control actions of the other controllers. The set γ_{init} represents the initial control action of i, i.e., all events that need to be enabled no matter what control action is taken. *EventOrdering* is an ordered list of the controllable events and represents the relative importance of enabling those events The symbol γ_i represents the local control action taken by i.

Algorithm 1 $VLP-GM\,(\sigma \in \Sigma_{oi} \cup \{\varepsilon\})$

$NS_i = PS_i\sigma \cap \mathcal{L}(G)$;

$\gamma_{init} = (\Sigma_{uc} \cup \Sigma_{ce} \setminus \Sigma_{cei})$

$\gamma_i = ControlAction(\gamma_{init}, NS_i, EventOrdering)$;

$PS_i = NS_i \left[\left(\gamma_i \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$;

$RETURN\ \gamma_i$;

END

Inside VLP-GM, the algorithm *ControlAction* calculates the next control action to be taken by i after the event σ observable to i occurs. *ControlAction* can be seen below and attempts to greedily enable the events ranked highest in *EventOrdering*

first. Note that because of the ordering of the events in $EventOrdering$, a controller could in a manner "steer" the system by allowing events with a higher priority more chances to become enabled. The $EventOrdering$ list does not necessarily need to have the same ordering for all controllers in a set of decentralized controllers running VLP-GM and differences in order may reflect some local control priorities. It is always assumed that $EventOrdering$ is constant for all local controllers, but the results of this chapter hold for the more general event lists.

Seen below, the $ControlAction$ algorithm is the greedy iterative part of VLP-GM that calculates what events can be enabled from an ordered lest of events $EList$. NS_i is the same variable as above in the VLP-GM algorithm and represents what states the system could be in before any unobservable events occur. The set ACT_i represents all events that $ControlAction$ has already enabled and naturally ACT_i is initialized to γ_{init} in VLP-GM. After the $ControlAction$ algorithm has completed, the final value for ACT_i is used to update γ_i. $ControlAction$ is greedy in that once an event is enabled and added to ACT_i, that event is never disabled until the next control action is calculated.

$ControlAction$ operates by cycling through all events in the event list, $EList$, from high priority to low. If an event is enabled, it is added to ACT_i and removed from $EList$ so it will not be considered again. If a high priority event is not enabled, that event is skipped but left on the list for later consideration and the next lowest priority event is tested.

If the lower priority event is enabled, it is added to ACT_i and removed from $EList$, but then $ControlAction$ cycles back to the top of the $EList$ to see if any high priority events can be enabled due to the enabling of the lower priority event.

Algorithm 2 $ControlAction(\gamma_{init}, NS_i, EList)$

$ACT_i = \gamma_{init}$;

$EList = (EventOrdering - \gamma_{init}) \cap \Sigma_{ci}$;

$Pt = 1$;

$While(Pt \leq |EList|)\ do$

{

$If\ Admissible(NS_i, EList.Pt, ACT_i)$

{

$ACT_i = ACT_i \cup \{EList.Pt\}$;

$EList = EList - EList.Pt$;

$Pt = 1$;

}

$Else$

$Pt++$;

}

$RETURN\ ACT_i$;

END

If an event is a "don't care" it is not enabled by VLP-GM, so an event that is initially "don't care" could become viable as more events are enabled. Also, enabling a lower priority event may cause a previously unenabled permissive higher priority event to become legal, so *ControlAction* needs to continually check its unenabled high priority events as more low priority events are enabled. Consider the following example.

Example 4 *In Figure 5.9, let G be the uncontrolled system and H be the desired controlled system for this example. The system could be controlled by several con-*

trollers, but only consider the actions of the i^{th} controller are considered. Suppose ρ, τ and μ are permissive events and that controller i can observe no events. Suppose further that ρ has a higher priority than μ. When VLP-GM executes for controller i, the high-priority permissive event ρ is tested first and not enabled because μ is initially disabled and ρ will never lead to a legal state in this system. Next, the lower priority event μ is tested and deemed valid so it is added to ACT_i. Now ρ leads to at least one legal system state in the unobservable reach of the system using the control action $ACT_i \cup \{\rho\}$. When ρ is retested, it can now be enabled and added to ACT_i even though previously it was inadmissible as an enabled event because of its "don't care" status.

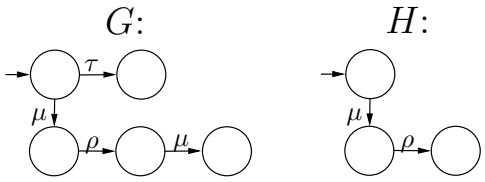

Figure 5.9: The uncontrolled system and desired system of Example 4.

When VLP-GM enables events, VLP-GM needs to make sure that not only that the event in itself is valid to be enabled, but VLP-GM also needs to make sure that if an event is enabled that event won't cause other, already enabled events to become illegal as with the system in Figure 5.9 above.

The function *Admissible* determines if an individual event σ should be enabled given a set of states that the control operation will be starting from, NS_i, and the set of events currently enabled by i, ACT_i.

The algorithm *Admissible* tests to see that all events in $(ACT_i \cup \{\sigma\}) \cap \Sigma_{ci}$ are still validly enabled if σ were to be enabled. To do this, *Admissible* looks at the

state estimate RS_i^+ which is constructed from NS_i by calculating the unobservable reach after NS_i if $\left(ACT_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i}\right)$ were enabled. RS_i^+ represents what behavior could occur in the system if σ were added to ACT_i.

Events in $ACT_i \cup \{\sigma\}$ should be assumed to enabled when calculating RS_i^+. To be on the safe side, controller i also assumes that all events in Σ_{cd}^{-i} are enabled globally when calculating RS_i^+ because another controller could enable those events without the knowledge of i. After RS_i^+ has been calculated, the system looks to see that all events in $(ACT_i \cup \{\sigma\}) \cap \Sigma_{cei}$ lead at least once to legal behavior and that all events in $(ACT_i \cup \{\sigma\}) \cap \Sigma_{cdi}$ always lead to legal behavior. If this condition holds, then σ can be legally added to ACT_i.

It is important to note that the *Admissible* algorithm does not label "don't care" events as admissible. If a "don't care" event were enabled, it would add nothing to the control action and could possibly hinder other events from being enabled. This is reflected in the line $\neg\left[\emptyset \subset \left(RS_i^+ \cap \overline{K}\right)\alpha \cap \mathcal{L}(G) \subseteq \overline{K}\right]$ for testing anti-permissive events in *Admissible*. If $\left(\emptyset = \left(RS_i^+ \cap \overline{K}\right)\alpha \cap \mathcal{L}(G)\right)$ were true for an anti-permissive event α then α would never lead from a legal state to another legal state using the current control action. This test for "don't care" events is an improvement over the *gdec* and *gmdec* control schemes because they could enable "don't care" events unnecessarily.

Admissible returns true if σ can be enabled and false if σ should not be enabled. The algorithm for *Admissible* can be seen below.

Algorithm 3 $Admissible(NS_i, \sigma, ACT_i)$

$RS_i^+ = NS_i\left[\left(ACT_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$;

$Output = True$;

$For\ all\ \alpha \in (ACT_i \cup \{\sigma\}) \cap \Sigma_{ci}$

{

$If\ \alpha\ is\ anti-permissive$

{

$If\ \neg\left[\emptyset \subset \left(RS_i^+ \cap \overline{K}\right)\alpha \cap \mathcal{L}(G) \subseteq \overline{K}\right]$

$Output = False;$

}

$Else\ (\alpha\ is\ permissive)$

{

$If\ \left[\left(RS_i^+\alpha \cap \overline{K}\right) = \emptyset\right]$

$Output = False;$

}

}

$RETURN\ Output;$

END

It is now demonstrated that the VLP-GM protocol generates a local maximal control protocol.

Theorem 14 *Excluding "don't care" events, the local control action generated by VLP-GM at s* (γ_i) *is maximal.*

Proof: The proof of this theorem is a direct consequence of the construction of the VLP-GM algorithm. At every system state s, every event in $\Sigma_{ci} \backslash ACT_i$ is tested for inclusion in the final ACT_i and is rejected. If an event could be included in ACT_i and is not, then that event is either not admissible or it would alter the admissibility of an event already in ACT_i, so there is no control protocol strictly larger than VLP-GM.

■

It can also be noted that supremal local control actions for VLP-GM do not exist in general. This point is illustrated in the following example.

Example 5 *For this example, consider a branch of the uncontrolled system seen in Figure 5.10.*

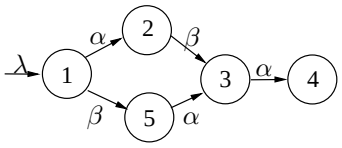

Figure 5.10: A branch of the automaton representing the system in Example 5.

Suppose λ is observable to controller i, but α and β are not. Suppose that after λ occurs, controller i knows that the system could only be in system state 1. Suppose further that all states are legal except for state 4. Let α and β be anti-permissive events that only i can control. VLP-GM would make two different control decision for i depending on EventOrdering. If α were the high priority event, then α would be the only event enabled by i out of $\{\alpha, \beta\}$, but if β were the high priority event, then β would be the only event enabled by i out of $\{\alpha, \beta\}$. It can therefore be seen that two maximal control protocols exist for controller i and the choice of EventOrdering is not trivial because supremal local VLP-GM control actions do not exist in general.

Another added advantage of VLP-GM is that when control actions are computed in an online manner, the algorithm has polynomial time complexity at each step. To see this, suppose the system G is a finite automata. This can be done because as stated earlier in the thesis regular legal and possible languages are discussed exclusively in this thesis. Let all states of the system G be included in the set X and as usual, Σ and Σ_{ci} represents all system events and events controllable by i respectively.

For this discussion of computational complexity, the states of the system in VLP-GM are no longer the strings generated by the system, but the states of the automata. Using standard algorithms for finite automata, the $NS_i = PS_i\sigma \cap \mathcal{L}(G)$ step, when converted to an equivalent expression for automata is of the order $O(|X|)$ and the $PS_i = NS_i \left[\left(\gamma_i \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$step, also when converted to an equivalent expression for automata is of the order $O(|X|\,|\Sigma_{uoi}|)$.

For the *ControlAction* algorithm, in the worst case, the algorithm will always enable only the last event on $EList$, and will continue to cycle through $EList$ until the list. In the worst case there will be $|EList| * (|EList| - 1)/2$ iterations of the while loop. $|EList| = |\Sigma_{ci}|$, so $|EList| * (|EList| - 1)/2 \in O\left(|\Sigma_{ci}|^2\right)$ and the code inside the while-loop will be executed on the order of $O\left(|\Sigma_{ci}|^2\right)$ times. Other than the *Admissible* function, all steps in *ControlAction* are of $O(1)$.

In the *Admissible* function, the step

$RS_i^+ = NS_i \left[ACT_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i} \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$, when converted to a finite automata expression, as with the similar steps for VLP-GM discussed earlier, is of the order $O(|X|\,|\Sigma_{uoi}|)$. There is another loop in the *Admissible* function that is iterated at most $|\Sigma_{ci}|$ times. The step to check the admissibility of permissive events, $\left[\left(PS_i^+\alpha \cap \overline{K}\right) = \emptyset\right]$, when converted to equivalent expressions for finite automata, is of the order $O(|X|)$ because the concatenation and intersection operations are of the order $O(|X|)$ and performed sequentially, while the test for the empty set is an elementary operation. Similarly, the step to check the admissibility of anti-permissive events, $\left[\emptyset \subset \left(PS_i^+ \cap \overline{K}\right)\alpha \cap \mathcal{L}(G) \subseteq \overline{K}\right]$, is also of the order $O(|X|)$ because the concatenation, intersection and inclusion test operations are of the order $O(|X|)$ and performed sequentially, while the test for the empty set is an elementary operation. Therefore, the *Admissible* function is of the order $O(\max(|X|\,|\Sigma_{uoi}|, |X|\,|\Sigma_{ci}|))$ and

ControlAction is computed on the order of $O\left(|\Sigma_{ci}|^2 \max\left(|X|\,|\Sigma_{uoi}|, |X|\,|\Sigma_{ci}|\right)\right) = O\left(\max\left(|X|\,|\Sigma_{uoi}|\,|\Sigma_{ci}|^2, |X|\,|\Sigma_{ci}|^3\right)\right)$. Finally, the VLP-GM algorithm takes order $O\left(\max\left(|X|\,|\Sigma_{uoi}|\,|\Sigma_{ci}|^2, |X|\,|\Sigma_{ci}|^3, |X|\right)\right)$ to complete. Therefore, the VLP-GM can be run in polynomial time.

5.6.1 Language Properties

Now that the VLP-GM control protocol has been introduced, several properties of languages generated by this control protocol are discussed. To start, a sufficient safety condition is provided for VLP-GM given $\overline{K}$, a legal sublanguage of $\mathcal{L}(G)$ where G is the uncontrolled system.

The proof of the safety theorem seen below for VLP-GM is very similar to the safety theorem introduced above for *gmdec* and relies on the fact that $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right)$ identifies all events that violate C&P co-observability when $\Sigma_c = \Sigma_{ce}$. First, a preliminary lemma needs to be proved that shows the incremental state estimates used by VLP-GM are indeed at least as small as the traditional inverse projection state estimate. This lemma is similar to Lemma 2 above where it was shown that $PS_i^+(s) \subseteq P_i^{-1}(P_i(s))$.

Lemma 7 *For all Σ_A, $NS_i\left[\Sigma_A \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s))$ where NS_i is the set calculated in the VLP-GM protocol by the i^{th} controller the last time VLP-GM was run and s is the state of the system.*

Proof: This lemma is proved by induction on the number of events observed by i.

It is assumed that the system has passed initialization, so $\varepsilon \in \overline{s}$.

Base:

$|P_i(s)| = 0$.

$s \in \Sigma_{uoi}^*$. Then $NS_i = \{\varepsilon\}$ and $P_i^{-1}(P_i(s)) = \Sigma_{uoi}^*$.

$[\Sigma_A \cap \Sigma_{uoi}]^* \subseteq \Sigma^*_{uoi}$, so $NS_i\,[\Sigma_A \cap \Sigma_{uoi}]^* \subseteq P_i^{-1}(P_i(s))$ holds in this case.

Induction Hypothesis:

$|P_i\,(s)| = n - 1$.

$NS_i\,[\Sigma_A \cap \Sigma_{uoi}]^* \subseteq P_i^{-1}(P_i(s))$ holds when s contains n events observable to i.

Induction Step:

$|P_i\,(s)| = n$.

Let $s = s'\alpha u$ where s contains n events observable to i, α is observable to i and $u \in \Sigma^*_{uoi}$. $|P_i\,(s')| = n-1$. Before α has occurred, $PS_i = NS_i\left[\left(\gamma_i \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$.

It should be noted that due to the induction hypothesis, $PS_i \subseteq P_i^{-1}(P_i(s'))$.

After α has occurred, NS_i is updated to $NS_i = PS_i\alpha \cap \mathcal{L}(G)$.

$PS_i \subseteq P_i^{-1}(P_i(s'))$.

$PS_i\alpha \subseteq P_i^{-1}(P_i(s'))\alpha$.

$PS_i\alpha \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s'))\alpha$.

$NS_i \subseteq P_i^{-1}(P_i(s'))\alpha$.

$NS_i\Sigma^*_{uoi} \subseteq P_i^{-1}(P_i(s'))\alpha\Sigma^*_{uoi}$.

$NS_i\,[\Sigma_A \cap \Sigma_{uoi}]^* \subseteq NS_i\Sigma^*_{uoi}$.

$NS_i\,[\Sigma_A \cap \Sigma_{uoi}]^* \cap \mathcal{L}(G) \subseteq NS_i\,[\Sigma_A \cap \Sigma_{uoi}]^*$.

$P_i^{-1}(P_i(s'))\alpha\Sigma^*_{uoi} = P_i^{-1}(P_i(s'\alpha u)) = P_i^{-1}(P_i(s))$.

so, $NS_i\,[\Sigma_A \cap \Sigma_{uoi}]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s))$ holds when s contains n events observable to i, which completes the proof by induction. ■

The previous lemma is used in the safety proof below due to the condition that VLP-GM enables events iteratively and so the $PS_i^+(s) \subseteq P_i^{-1}(P_i(s))$ property could not have been used previously in proving the safety of *gmdec*. Except for this distinction, the safety proof below is very similar to the safety proof for VLP-GM. Without loss of generality $\overline{K}$ is assumed to be controllable.

Theorem 15 $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{VLPGM}/G) \subseteq \overline{K}$

Proof: To prove this theorem, it is shown that given $\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd}$, $s \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow s \in \overline{K}$. This statement is proved by induction on the string s.

Base:

$|s| = 0, s = \varepsilon$.

$\mathcal{L}(S_{VLPGM}/G)$ and $\overline{K}$ are non-empty, so $\varepsilon \in \mathcal{L}(S_{VLPGM}/G)$ and $\varepsilon \in \overline{K}$. Therefore, $\varepsilon \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow \varepsilon \in \overline{K}$ is valid and the base case holds.

Induction hypothesis:

$|s| = n - 1$.

Assume $s \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow s \in \overline{K}$ holds.

Induction step:

$|s| = n$.

Let $s = s'\sigma$.

The induction step is broken down into three exhaustive cases based on the classification of σ: $(\sigma \in \Sigma_{uc})$, $(\sigma \in \Sigma_{cd})$, $(\sigma \in \Sigma_{ce})$.

Case 1 *:* $\sigma \in \Sigma_{uc}$

$s'\sigma \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow (s'\sigma \in \mathcal{L}(G)) \wedge (s' \in \mathcal{L}(S_{VLPGM}/G))$.

$\Rightarrow (s'\sigma \in \mathcal{L}(G)) \wedge \left(s' \in \overline{K}\right)$ by the induction hypothesis.

$\overline{K}$ is assumed to be controllable, so $(s'\sigma \in \mathcal{L}(G)) \wedge \left(s' \in \overline{K}\right) \Rightarrow \left(s'\sigma \in \overline{K}\right)$.

Which completes the proof of this case that $s \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow s \in \overline{K}$.

Case 2 *:* $\sigma \in \Sigma_{cd}$

Assume there exists a string $s' \in \mathcal{L}(S_{VLPGM}/G) \cap \overline{K}$ and $\sigma \in \Sigma_{ce}$ such that $s'\sigma \in \mathcal{L}(S_{VLPGM}/G)$ and $s'\sigma \notin \overline{K}$.

Intuitively, after s' has occurred:

$s' \in NS_i \left[\left(\gamma_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$ for all controllers.

Let $RS_i^+ = NS_i \left[\left(\gamma_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$ after s' has occurred σ first needs to be tested if it is valid to be enabled by any controller.

Because $s' \in \overline{K}$, for all $i \in I$, $s' \in RS_i^+ \cap \overline{K}$.

Therefore: $s'\sigma \in \left(RS_i^+ \cap \overline{K}\right)\sigma$.

Furthermore, $(s'\sigma \in \mathcal{L}(S_{VLPGM}/G)) \Rightarrow (s'\sigma \in \mathcal{L}(G))$.

So, $s'\sigma \in \left(RS_i^+(s') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G)$ and $s'\sigma \notin \overline{K}$ for all $i \in I$.

If σ were enabled, there must be some controller such that:

$\left(\left(RS_i^+ \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \wedge (\sigma \in \Sigma_{cdi})$ holds, but:

$\left[(\forall i \in I)\left(\left(RS_i^+ \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right)\right]$.

$\Rightarrow \left[(\forall i \in I)\left(\left(RS_i^+ \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right) \vee (\sigma \notin \Sigma_{cdi})\right]$, so σ is not valid to be enabled by any controller after s' has occurred.

$\Rightarrow \sigma \notin S_{VLPGM}(s')$.

$\Rightarrow s'\sigma \notin \mathcal{L}(S_{VLPGM}/G)$.

This contradicts the assumption that $s'\sigma \in \mathcal{L}(S_{VLPGM}/G)$, so if $\sigma \in \Sigma_{cd}$,

$$(s \in \mathcal{L}(S_{VLPGM}/G)) \Rightarrow \left(s \in \overline{K}\right).$$

Case 3 *:* $\sigma \in \Sigma_{ce}$

This case is proved by contradiction.

Suppose $s'\sigma \in \mathcal{L}(S_{VLPGM}/G)$, $s' \in \mathcal{L}(S_{VLPGM}/G)$, $s'\sigma \notin \overline{K}$, $s' \in \overline{K}$.

Since $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd}$ and $\Sigma_{cd} \cap \Sigma_{ce} = \emptyset$, $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \cap \Sigma_{ce} = \emptyset$.

Therefore, $\overline{K}$ is C&P co-observable w.r.t. $\mathcal{L}(G), \Sigma_{o1}, \Sigma_{ce1}, \ldots, \Sigma_{on}, \Sigma_{cen}$.

By definition of C&P co-observability:

$\left(s' \in \overline{K}\right) \wedge \left(s'\sigma \notin \overline{K}\right) \Rightarrow \left[(\exists i \in I)\left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$.

After $s\prime$ has occurred, for all controllers:

$NS_i\left[\left(\gamma_i \cup \{\sigma\} \cup \left(\Sigma_{cd}^{-i}\right)\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s'))$.

Let $RS_i^+ = NS_i\left[\left(\gamma_i \cup \{\sigma\} \cup \left(\Sigma_{cd}^{-i}\right)\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$. The event σ first needs to be tested if it is valid to be enabled by at least one controller.

$RS_i^+ \subseteq P_i^{-1}(P_i(s'))$ by the Lemma 7.

$\Rightarrow RS_i^+\sigma \subseteq P_i^{-1}(P_i(s'))\sigma$.

$\Rightarrow RS_i^+\sigma \cap \overline{K} \subseteq P_i^{-1}(P_i(s'))\sigma \cap \overline{K}$.

$\Rightarrow \left[\left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \Rightarrow \left(RS_i^+\sigma \cap \overline{K} = \emptyset\right)\right]$.

$\left(s' \in \overline{K}\right) \wedge \left(s'\sigma \notin \overline{K}\right) \Rightarrow \left[(\exists i \in I)\left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$.

$\Rightarrow \left[(\exists i \in I)\left(RS_i^+\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$, so σ is invalid to be enabled by at least one controller.

$\Rightarrow \sigma \notin S_{VLPGM}(s')$.

$\Rightarrow s'\sigma \notin \mathcal{L}(S_{VLPGM}/G)$.

This contradicts the assumption that $s'\sigma \in \mathcal{L}(S_{VLPGM}/G)$, so if $\sigma \in \Sigma_{ce}$,

$$(s \in \mathcal{L}(S_{VLPGM}/G)) \Rightarrow \left(s \in \overline{K}\right)$$

which completes the proof by induction that $(s \in \mathcal{L}(S_{VLPGM}/G)) \Rightarrow \left(s \in \overline{K}\right)$, so

$$\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{VLPGM}/G) \subseteq \overline{K}. \tag{5.11}$$

■

As an interesting side note, the VLP-GM protocol may use maximal local control actions to generate sublanguages of $\overline{K}$, but does the VLP-GM protocol necessarily generate maximal controllable and co-observable sublanguages of $\overline{K}$? In general, no. Consider the following theorem which is demonstrated by example.

Theorem 16 *The VLP-GM control scheme cannot in general generate a maximal*

controllable and co-observable sublanguage of $\overline{K}$ over a system G if a sufficient safety condition is satisfied.

Proof: To prove this theorem, let us examine the following example. Consider the uncontrolled system G and desired controlled system H seen in Figure 5.11.

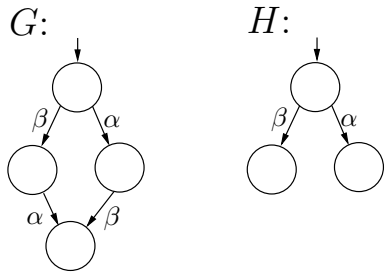

Figure 5.11: Uncontrolled system G and desired system H for Theorem 15.

As before, $\mathcal{L}(H) = \overline{K}$.

Suppose two controllers using the VLP-GM control scheme attempt to achieve $\overline{K}$ over G with the following properties:

$\Sigma_{o1} = \emptyset$.

$\Sigma_{ce1} = \emptyset$.

$\Sigma_{cd1} = \{\alpha, \beta\}$.

$\Sigma_{o2} = \emptyset$.

$\Sigma_{ce2} = \emptyset$.

$\Sigma_{cd2} = \{\alpha, \beta\}$.

$\{\alpha, \beta\} = \Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} = \{\alpha, \beta\}$, so this system is safe.

After using the VLP-GM protocol to calculate the control actions, for all permutations of the event orderings, $\gamma_1(\varepsilon) = \gamma_2(\varepsilon) = \emptyset$, so $S_{VLPGM}(\varepsilon) = \emptyset$ and $\mathcal{L}(S_{VLPGM}/G) = \{\varepsilon\}$.

Two maximal controllable and co-observable sublanguages exist for this problem; $\mathcal{L}_1 = \{\varepsilon, \alpha\}$ and $\mathcal{L}_2 = \{\varepsilon, \beta\}$. The VLP-GM protocol does not generate a maximal controllable and co-observable sublanguage for this problem for any event ordering, so the VLP-GM control scheme cannot in general generate a maximal controllable and co-observable sublanguage of $\overline{K}$ over a system G if a sufficient safety condition is satisfied. ■

With the above theorem, an interesting open problem is to demonstrate a sufficient condition such that a locally maximal control protocol will generate a maximal controllable and co-observable sublanguage.

5.6.2 Language Comparison

Now that the VLP-GM control algorithm has been introduced and it has been shown that the local enabling protocols of VLP-GM are maximal for the controllers' given information, one would think that the languages generated using the VLP-GM control algorithm would automatically be at least as large as the languages generated by the *gmdec* and *gdec* protocols. However, this is not always the case. No general statements can be made comparing the safe languages generated by VLP-GM with those generated by *gmdec* and *gdec*. There are some problem instances where VLP-GM generates larger safe languages than *gdec* and gmdec, but there are also problem instances where the language generated by VLP-GM is incomparable with the languages generated by *gdec* and *gmdec*. Surprisingly, there are also examples where VLP-GM generates languages strictly smaller than the languages generated by *gdec* and *gmdec* due to the fact that VLP-GM is only locally maximal in its control decisions. Three examples are now shown to demonstrate these three cases.

Example 6 *VLP-GM generates a language larger than the languages generated by*

gdec and gmdec.

Consider the uncontrolled system G and desired controlled system H seen in Figure 5.12.

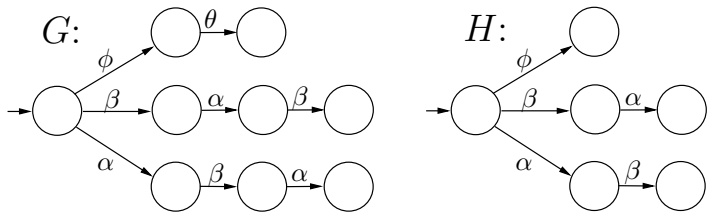

Figure 5.12: The uncontrolled system G and desired system H of Example 6.

The system has two controllers and the following properties:

$\Sigma_o = \emptyset$, $\Sigma_{uc} = \emptyset$, $\Sigma_{c1} = \{\alpha, \beta, \theta\}$, $\Sigma_{c2} = \{\theta, \phi\}$, $\Sigma_{cd} = \{\alpha, \beta\}$, $\Sigma_{ce} = \{\phi, \theta\}$.

$\Sigma_{cd1} = \{\alpha, \beta\}$, $\Sigma_{ce1} = \{\theta\}$, $\Sigma_{cd2} = \emptyset$, $\Sigma_{ce2} = \{\theta, \phi\}$,

$EventOrdering = \{\alpha, \beta, \theta, \phi\}$.

Note that $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) = \{\alpha, \beta\}$, *so this system is safe.*

The languages generated S_{gmdec} *and* S_{gdec} *are first investigated.*

$$(\Sigma_o = \emptyset) \Rightarrow \left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right] \Rightarrow \left(\mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G)\right).$$

$\gamma_1^{gmdec}(\varepsilon) = \gamma_1^{gdec}(\varepsilon) = \{\phi\}$, $\gamma_2^{gmdec}(\varepsilon) = \gamma_2^{gdec}(\varepsilon) = \{\phi\}$.

Therefore, $S_{gmdec}(\varepsilon) = S_{gdec}(\varepsilon) = \{\phi\}$ *and* $\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G) = \{\varepsilon, \phi\}$.

Now to investigate the language generated by VLP-GM.

$\gamma_1^{VLPGM}(\varepsilon) = \{\alpha, \phi\}$, $\gamma_2^{VLPGM}(\varepsilon) = \{\phi\}$.

Therefore, $S_{VLPGM}(\varepsilon) = \{\alpha, \phi\}$ *and* $\mathcal{L}(S_{VLPGM}/G) = \{\varepsilon, \alpha, \phi\}$.

$\mathcal{L}(S_{VLPGM}/G) = \{\varepsilon, \alpha, \phi\}$, $\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G) = \{\varepsilon, \phi\}$.

So, for this problem, $\mathcal{L}(S_{VLPGM}/G)$ *is larger than* $\mathcal{L}(S_{gmdec}/G)$ *or* $\mathcal{L}(S_{gdec}/G)$.

In the previous example, $\mathcal{L}(S_{VLPGM}/G)$ is larger than $\mathcal{L}(S_{gmdec}/G)$ or $\mathcal{L}(S_{gdec}/G)$, which is what was hoped to have happened. Notice that in the previous example, $\left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right]$, but $\mathcal{L}(S_{VLPGM}/G)$ is larger than $\mathcal{L}(S_{gmdec}/G)$ or $\mathcal{L}(S_{gdec}/G)$. This therefore implies that $\left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right]$ is not a sufficient condition for $\mathcal{L}(S_{VLPGM}/G)$ to be equal to $\mathcal{L}(S_{gdec}/G)$. It would seem that the property of VLP-GM to be locally maximal when enabling events would lead to larger languages than *gdec* and *gmdec*. However, as will be seen in the next example, this is not always the case.

Example 7 *VLP-GM generates a language incomparable with the languages generated by gdec and gmdec.*

Consider the uncontrolled system G and desired controlled system H seen in Figure 5.13.

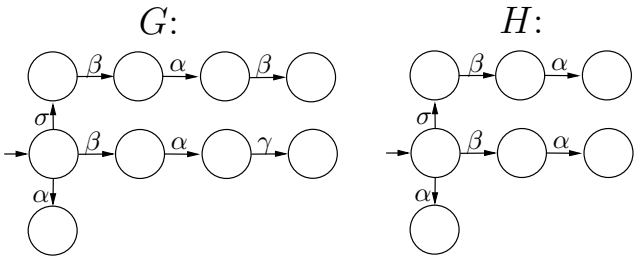

Figure 5.13: The uncontrolled system G and desired system H of Example 7.

The system has two controllers and the following properties:

$\Sigma_o = \emptyset$, $\Sigma_{uc} = \{\sigma\}$, $\Sigma_{c1} = \{\alpha, \beta\}$, $\Sigma_{c2} = \{\gamma\}$, $\Sigma_{cd} = \{\beta, \gamma\}$, $\Sigma_{ce} = \{\alpha\}$.

$\Sigma_{cd1} = \{\beta\}$, $\Sigma_{ce1} = \{\alpha\}$, $\Sigma_{cd2} = \{\gamma\}$, $\Sigma_{ce2} = \emptyset$, $EventOrdering = \{\gamma, \beta, \alpha\}$.

$\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) = \{\beta, \gamma\}$, *so this system is safe.*

The languages generated S_{gmdec} *and* S_{gdec} *first need to be investigated.*

$$(\Sigma_o = \emptyset) \Rightarrow \left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right] \Rightarrow \left(\mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G)\right).$$

$\gamma_1^{gmdec}(\varepsilon) = \gamma_1^{gdec}(\varepsilon) = \{\sigma, \alpha\}$, $\gamma_2^{gmdec}(\varepsilon) = \gamma_2^{gdec}(\varepsilon) = \{\sigma, \alpha\}$.

Therefore, $S_{gmdec}(\varepsilon) = S_{gdec}(\varepsilon) = \{\sigma, \alpha\}$ *and* $\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G) = \{\varepsilon, \alpha, \sigma\}$.

Now to investigate the language generated by VLP-GM.

$\gamma_1^{VLPGM}(\varepsilon) = \{\sigma, \beta\}$, $\gamma_2^{VLPGM}(\varepsilon) = \{\sigma, \alpha\}$.

Therefore, $S_{VLPGM}(\varepsilon) = \{\sigma, \beta\}$ *and* $\mathcal{L}(S_{VLPGM}/G) = \{\varepsilon, \beta, \sigma, \sigma\beta\}$.

$\mathcal{L}(S_{VLPGM}/G) = \{\varepsilon, \beta, \sigma, \sigma\beta\}$, $\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G) = \{\varepsilon, \alpha, \sigma\}$.

So, for this problem instance, $\mathcal{L}(S_{VLPGM}/G)$ *is incomparable with* $\mathcal{L}(S_{gmdec}/G)$ *and* $\mathcal{L}(S_{gdec}/G)$.

In the example above both the *gmdec* and *gdec* local controllers enable α because α is permissive and leads to at least one legal state in the controllers' unobservable reach. β is not enabled by *gmdec* or *gdec* because β leads to an illegal state in controller one's unobservable reach and β is an anti-permissive event. On the other hand, for VLP-GM, β is tested before α by controller one under the VLP-GM scheme. Therefore, β gets enabled by controller one. When controller one tests event α, α is inadmissible because if α, were enabled, β would no longer be a valid enabled event by controller one. Because α is a permissive event and controller one disables it, α becomes disabled globally. This example should show that VLP-GM can generate a language incomparable to the languages generated by *gmdec* or *gdec*; controllers using the VLP-GM protocol can enable high priority events that would normally be disabled by the other control schemes and the high-priority events could cause low-priority events that are normally enabled by *gmdec* or *gdec* to be disabled because enabling the low-priority evens would make the high-priority events to become inadmissible.

Now, to show an example where even though the languages are safe, the controlled language $\mathcal{L}(S_{VLPGM}/G)$ is strictly smaller than $\mathcal{L}(S_{gmdec}/G)$ and $\mathcal{L}(S_{gdec}/G)$.

Example 8 *VLP-GM generates a language strictly smaller than the languages generated by gdec and gmdec.*

Suppose there are two controllers (1 and 2) for the uncontrolled system G seen in Figure 5.14. Suppose further that $\overline{K} = \mathcal{L}(H)$.

The controllers have the following properties:

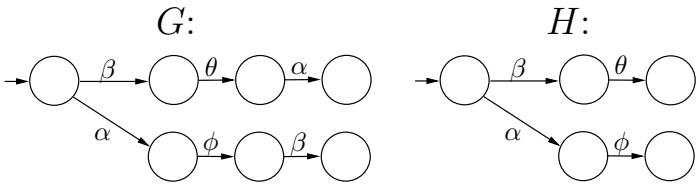

Figure 5.14: The uncontrolled system G and desired system H of Example 8.

$\Sigma_{o1} = \{\theta\}$ $\Sigma_{o2} = \{\phi\}$ $\Sigma_{c1} = \{\alpha, \beta\}$ $\Sigma_{c2} = \{\beta\}$ $\Sigma_{uc} = \{\phi, \theta\}$ $\Sigma_{ce} = \{\alpha\}$ $\Sigma_{cd} = \{\beta\}$ $EList = \{\beta, \alpha\}$.

First, the language $\overline{K}$ *is controllable and co-observable. Therefore, by [83] and Corollary 4,* $\overline{K} = \{\varepsilon, \beta, \beta\theta, \alpha, \alpha\phi\} = \mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{gmdec}/G)$.

For the language generated by the VLP-GM control scheme:

$\gamma_1^{VLPGM}(\varepsilon) = \{\phi, \theta, \beta\}$.

$\gamma_1^{VLPGM}(\beta\theta) = \{\phi, \theta\}$.

$\gamma_2^{VLPGM}(\varepsilon) = \{\phi, \theta, \alpha, \beta\}$.

$S_{VLPGM}(\varepsilon) = \{\phi, \theta, \beta\}$.

$S_{VLPGM}(\beta\theta) = \{\phi, \theta, \beta\}$.

$\mathcal{L}(S_{VLPGM}/G) = \{\varepsilon, \beta, \beta\theta\}$.

From this example, it is evident that $\mathcal{L}(S_{VLPGM}/G) \subseteq \mathcal{L}(S_{gdec}/G)$. *This is due to the fact that the initial control action of 1, VLP-GM tests and enables event* β *before event* α *so that when* α *is tested,* α *is not valid to be enabled.*

It should also be noted in the last example that the specification language is controllable and co-observable, but $\mathcal{L}(S_{VLPGM}/G) \neq \overline{K}$. This should demonstrate a further deficiency in VLP-GM. Not only will VLP-GM sometimes generate smaller languages than *gdec*, VLP-GM will not always be able to achieve controllable and co-observable sublanguages exactly.

Even though for some cases the controlled behavior $\mathcal{L}(S_{VLPGM}/G)$ may be strictly smaller than $\mathcal{L}(S_{gdec}/G)$ and $\mathcal{L}(S_{gmdec}/G)$, the extra control to steer the system granted by the VLP-GM scheme makes it advantageous to use at times. In the last example, VLP-GM tries too hard to enable β at all local controllers at the expense of no controller ever enabling α. This can be explained by the observation that VLP-GM is trying too hard to enable the high-priority events at all local sites at the expense of never enabling the low priority events.

The previous three examples showed that no general statements can be made comparing $\mathcal{L}(S_{VLPGM}/G)$ with $\mathcal{L}(S_{gmdec}/G)$ and $\mathcal{L}(S_{gdec}/G)$. There are problem instances where VLP-GM generates a larger language than *gmdec* or *gdec* as there may be problem instances where VLP-GM generates a smaller language than *gmdec* or *gdec*. Except for the case where $\overline{K}$ is controllable and co-observable, it remains an open problem to show when it might be better to use VLP-GM versus *gdec* or *gmdec*. An improvement upon VLP-GM is now attempted using new algorithm that will always generates control actions locally at least as well as *gdec*.

5.7 VLP-GM2 Algorithm

It may be surprising and disappointing that the languages generated by VLP-GM may be smaller than the languages generated by gdec, but is it possible to modify the VLP-GM algorithm to get improved behavior? What if elements of the *gmdec* protocol were to combined with elements of the VLP-GM algorithm? Would this combination always give a larger generated language when controlling systems? As a controller using this hypothetical protocol is given state estimates, the controller would need to calculate safe control actions. The controller could start by enabling all events that *gmdec* would enable for that state estimate. After removing all the "don't care" events from the *gmdec* control action that are guaranteed not to modify the controlled behavior, the controller could then iteratively enable more events in the same manner as VLP-GM. Might this control method have some desirable maximality properties? This new outlined combination of control protocols is called VLP-GM2 which can be seen below.

Algorithm 4 $VLP-GM2\left(\sigma \in \Sigma_{oi} \cup \{\varepsilon\}\right)$

$NS_i = PS_i\sigma \cap \mathcal{L}(G)$;

$\gamma_{gmdec} = \gamma_i^{gmdec}\left(NS_i\right)$;

$\gamma_{filt} = controlFilter\left(NS_i, \gamma_{gmdec}, \Sigma_{ci}\right)$

$\gamma_i = ControlAction(\gamma_{filt}, NS_i, EventOrdering)$;

$PS_i = NS_i\left[\left(\gamma_i \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$;

$RETURN\ \gamma_i$;

END

VLP-GM2 calculates control action at initialization and on the occurrence of locally observable events. PS_i is a state estimate very similar to the PS_i used in

VLP-GM. On start-up, PS_i is set to $\{\varepsilon\}$. Also as before, NS_i represents where the system might be before any unobservable events occur. The state estimate NS_i is passed to γ_i^{gmdec} to calculate the initial control action - all events that *gmdec* would enable if $PS_i^+(s) = NS_i\Sigma_{uoi}^* \cap \mathcal{L}(G)$ where s is the string of events generated by the system. Because some of these events are "don't care" they will not add any extra behavior to the system. Therefore, these events need to be filtered from the local control action in the *controlFilter* algorithm discussed later. The filtered initial control action is returned in the variable γ_{filt} and passed to *ControlAction*. *ControlAction* then attempts to enable more events in the flavor of the VLP-GM algorithm such that γ_{filt} is the initial control action and more events are iteratively tested and possibly added to this active event set as discussed earlier.

The following algorithm calculates the control action of *gmdec* given the current state estimate before unobservable events occur, NS_i

Algorithm 5 $\gamma_i^{gmdec}(NS_i)$

$\gamma_i^{uc} = \Sigma_{uc} \cup \Sigma_{ce} \backslash \Sigma_{cei}$;

$PS_i^{gmdec} = NS_i\Sigma_{uoi}^* \cap \mathcal{L}(G)$

$\gamma_i^d = \left\{\sigma \in \Sigma_{cdi} : \emptyset \subset \left(PS_i^{gmdec} \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right\}$

$\gamma_i^e = \left\{\sigma \in \Sigma_{cei} : PS_i^{gmdec}\sigma \cap \overline{K} \neq \emptyset\right\}$

$RETURN\ \left(\gamma_i^{uc} \cup \gamma_i^d \cup \gamma_i^e\right)$

The following *controlFilter* algorithm filters out all of the "don't care" events from the γ control action. This ensures that all events that are locally enabled could add to the behavior of the system. Notice that $NS_i\left[\left(\gamma \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* = NS_i\left[\left(\gamma_{filt} \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*$. This statement is given without proof, but can be easily shown because $(\gamma - \gamma_{filt})$ contains only "don't care" events for the given state

estimate. The algorithm tests if an event σ should be filtered out by searching over all states in $NS_i \left[\left(\gamma \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*$ and making sure that σ leads at least once to another possible state. When implemented to control a finite automata system, the *controlFilter* algorithm is in $O(|X| \max(|\Sigma_{uoi}|, |\Sigma_{ci}|))$.

Algorithm 6 *controlFilter* $(NS_i, \gamma, \Sigma_{ci})$

$\gamma_{filt} = \left\{\sigma \in \Sigma_{ci} | NS_i \left[\left(\gamma \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \sigma \cap \mathcal{L}(G) \neq \emptyset\right\} \cup \Sigma_{uc} \cup \Sigma_{ce} \backslash \Sigma_{cei}$;

RETURN γ_{filt};

As would be expected, VLP-GM2 can be run in polynomial time when the algorithm is used to update a control action on the occurrence of a locally observable event. The *controlFilter* algorithm is in $O(|X| \max(|\Sigma_{uoi}|, |\Sigma_{ci}|))$. As discussed earlier, γ_i^{gmdec} is also in $O(|X| \max(|\Sigma_{uoi}|, |\Sigma_{ci}|))$ and updating NS_i and PS_i are in $O(|X||\Sigma_{uoi}|)$. As for VLP-GM, *ControlAction* is in $O(|X||\Sigma_{ci}|^2 \max(|\Sigma_{uoi}|, |\Sigma_{ci}|))$.

Therefore, VLP-GM2 is in $O(|\Sigma_{ci}|^2 \max(|X||\Sigma_{uoi}|, |X||\Sigma_{ci}|))$. As would be expected, VLP-GM2 would be exponential in $|X|$ if used to calculate all possible local control actions because control actions would be determined by an observer automaton.

Theorem 17 *Excluding "don't care" events, the local control action generated by VLP-GM2 at s is maximal for its provided state estimate.*

Proof: The proof of this theorem is a direct consequence of the construction of the VLP-GM2 algorithm. At every system state s, every event in $\Sigma_{ci} \backslash ACT_i$ has been tested for inclusion in the final ACT_i and is rejected. If an event could be included in ACT_i and is not, then that event is either not admissible or it would alter the admissibility of an event already in ACT_i, so there is no control protocol strictly larger than VLP-GM2. ■

It should be noted that VLP-GM2 is similar to VLP-GM because both of these control protocols enable a locally maximal set of events possible for its given state estimate. Although it is not shown, supremal control actions can be shown to not exist in general for VLP-GM2 by an example such as 5.

As will be shown later, by initially enabling all events that *gmdec* does, VLP-GM2 partially avoids the pitfall of trying too hard to enable the high priority events locally so that VLP-GM2 generates controlled languages always at least as large as *gdec* if *gdec* generates a safe language.

5.7.1 Language Properties

Now to explore some properties of the languages generated by using the VLP-GM2 algorithm to control a system G with specification language $\overline{K}$. First a sufficient safety condition is shown that is identical to the safety conditions of the previously discussed control protocols in this chapter after some a necessary lemma used for comparing the state estimators of *gdec* and VLP-GM2.

Lemma 8 *For all* Σ_A, $NS_i\left[\Sigma_A \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s))$ *where* NS_i *is the set calculated in the VLP-GM2 protocol by the* i^{th} *controller the last time VLP-GM was run and* s *is the state of the system.*

Proof: This lemma is proved by induction on the number of events observed by i.

It is assumed that the system has passed initialization, so $\varepsilon \in \overline{s}$

Base:

$|P_i(s)| = 0$.

$s \in \Sigma_{uoi}^*$. Then $NS_i = \{\varepsilon\}$ and $P_i^{-1}(P_i(s)) = \Sigma_{uoi}^*$.

$\left[\Sigma_A \cap \Sigma_{uoi}\right]^* \subseteq \Sigma_{uoi}^*$, so $NS_i\left[\Sigma_A \cap \Sigma_{uoi}\right]^* \subseteq P_i^{-1}(P_i(s))$ holds in this case.

Induction Hypothesis:

$|P_i(s)| = n - 1$.

$NS_i[\Sigma_A \cap \Sigma_{uoi}]^* \subseteq P_i^{-1}(P_i(s))$ holds when s contains n events observable to i.

Induction Step:

$|P_i(s)| = n$.

Let $s = s'\alpha u$ where s contains n events observable to i, α is observable to i and $u \in \Sigma_{uoi}^*$. $|P_i(s')| = n-1$. Before α has occurred, $PS_i = NS_i\left[\left(\gamma_i \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$.

It should be noted that due to the induction hypothesis, $PS_i \subseteq P_i^{-1}(P_i(s'))$.

After α has occurred,NS_i is updated to $NS_i = PS_i\alpha \cap \mathcal{L}(G)$.

$PS_i \subseteq P_i^{-1}(P_i(s'))$.

$PS_i\alpha \subseteq P_i^{-1}(P_i(s'))\alpha$.

$PS_i\alpha \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s'))\alpha$.

$NS_i \subseteq P_i^{-1}(P_i(s'))\alpha$.

$NS_i\Sigma_{uoi}^* \subseteq P_i^{-1}(P_i(s'))\alpha\Sigma_{uoi}^*$.

$NS_i[\Sigma_A \cap \Sigma_{uoi}]^* \subseteq NS_i\Sigma_{uoi}^*$.

$NS_i[\Sigma_A \cap \Sigma_{uoi}]^* \cap \mathcal{L}(G) \subseteq NS_i[\Sigma_A \cap \Sigma_{uoi}]^*$.

$P_i^{-1}(P_i(s'))\alpha\Sigma_{uoi}^* = P_i^{-1}(P_i(s'\alpha u)) = P_i^{-1}(P_i(s))$.

so, $NS_i[\Sigma_A \cap \Sigma_{uoi}]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s))$ holds when s contains n events observable to i, which completes the proof by induction. ■

As before, it is assumed without loss of generality that $\overline{K}$ is controllable but not necessarily co-observable. A sufficient safety condition for VLP-GM2 can now be proved.

Theorem 18 $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{VLPGM2}/G) \subseteq \overline{K}$

Proof: To prove this theorem, it is shown that given $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd}$, $s \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow s \in \overline{K}$. This statement is proved by induction on the string s.

Base:

$|s| = 0, s = \varepsilon$.

Without loss of generality, $\mathcal{L}(S_{VLPGM2}/G)$ and $\overline{K}$ are assumed to be non-empty, so $\varepsilon \in \mathcal{L}(S_{VLPGM}/G)$ and $\varepsilon \in \overline{K}$. Therefore, $\varepsilon \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow \varepsilon \in \overline{K}$ is valid and the base case holds.

Induction hypothesis:

$|s| = n - 1$.

Assume $s \in \mathcal{L}(S_{VLPGM}/G) \Rightarrow s \in \overline{K}$ holds.

Induction step:

$|s| = n$.

Let $s = s'\sigma$.

The induction step is broken down into three exhaustive cases based on the classification of σ: $(\sigma \in \Sigma_{uc})$, $(\sigma \in \Sigma_{cd})$, $(\sigma \in \Sigma_{ce})$.

Case 1 *:* $\sigma \in \Sigma_{uc}$

$s'\sigma \in \mathcal{L}(S_{VLPGM2}/G) \Rightarrow (s'\sigma \in \mathcal{L}(G)) \wedge (s' \in \mathcal{L}(S_{VLPGM2}/G))$.

$\Rightarrow (s'\sigma \in \mathcal{L}(G)) \wedge \left(s' \in \overline{K}\right)$ by the induction hypothesis

$\overline{K}$ is assumed to be controllable, so $(s'\sigma \in \mathcal{L}(G)) \wedge \left(s' \in \overline{K}\right) \Rightarrow \left(s'\sigma \in \overline{K}\right)$.

Which completes the proof of this case that $s \in \mathcal{L}(S_{VLPGM2}/G) \Rightarrow s \in \overline{K}$.

Case 2 *:* $\sigma \in \Sigma_{cd}$

Assume there exists a string $s' \in \mathcal{L}(S_{VLPGM2}/G) \cap \overline{K}$ and $\sigma \in \Sigma_{cd}$ such that $s'\sigma \in \mathcal{L}(S_{VLPGM2}/G)$, $s'\sigma \in \mathcal{L}(S/G)$ and $s'\sigma \notin \overline{K}$

First suppose σ was enabled by the *gmdec* step of VLP-GM2.

Intuitively, after s' has occurred, $s' \in NS_i$.

$\Rightarrow (\forall i \in I)\,[s' \in NS_i\Sigma^*_{uoi}]$.

$\Rightarrow (\forall i \in I)\,[s' \in NS_i\Sigma^*_{uoi} \cap \mathcal{L}(G)]$.

$\Rightarrow (\forall i \in I)\left[s' \in PS_i^{gmdec}\right]$.

$\Rightarrow (\forall i \in I)\left[s' \in \left(PS_i^{gmdec} \cap \overline{K}\right)\right]$.

$\Rightarrow (\forall i \in I)\left[s'\sigma \in \left(PS_i^{gmdec} \cap \overline{K}\right)\sigma\right]$.

$\Rightarrow (\forall i \in I)\left[s'\sigma \in \left(PS_i^{gmdec} \cap \overline{K}\right)\sigma \cap \mathcal{L}(G)\right]$.

$\Rightarrow (\forall i \in I)\left[\left(PS_i^{gmdec} \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right]$.

Thus, it is obviously impossible that σ is enabled by any controller at the *gmdec* step of VLP-GM2.

Now suppose σ was enabled by the *AdditionalControlAction* step of VLP-GM2.

Intuitively, after s' has occurred, $s' \in NS_i\left[\left(\gamma_i \cup \{\sigma\} \cup \left(\Sigma_{cd}^{-i}\right)\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$ for all controllers.

Let $RS_i^+ = NS_i\left[\left(\gamma_i \cup \{\sigma\} \cup \left(\Sigma_{cd}^{-i}\right)\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$ after s' has occurred. We need to test to see if σ is valid to be enabled by any controller.

Because $s' \in \overline{K}$, for all $i \in I$, $s' \in RS_i^+ \cap \overline{K}$.

Therefore: $s'\sigma \in \left(RS_i^+ \cap \overline{K}\right)\sigma$.

So, $s'\sigma \in \left(RS_i^+(s') \cap \overline{K}\right)\sigma \cap \mathcal{L}(G)$ and $s'\sigma \notin \overline{K}$ for all $i \in I$.

If σ were enabled, there must be some controller such that

$$\left(\left(RS_i^+ \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \wedge (\sigma \in \Sigma_{cdi})\,.$$

holds, but:

$\left[(\forall i \in I)\left(\left(RS_i^+ \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right)\right]$.

$\Rightarrow \left[(\forall i \in I)\left(\left(RS_i^+ \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \not\subseteq \overline{K}\right) \vee (\sigma \notin \Sigma_{cdi})\right]$, so σ is not valid to be enabled by any controller at the *AdditionalControlAction* step of VLP-GM2 after s' has occurred.

$\Rightarrow \sigma \notin S_{VLPGM2}(s')$.

$\Rightarrow s'\sigma \notin \mathcal{L}(S_{VLPGM2}/G)$.

This contradicts the assumption that $s'\sigma \in \mathcal{L}(S_{VLPGM2}/G)$, so if $\sigma \in \Sigma_{cd}$,

$$(s \in \mathcal{L}(S_{VLPGM2}/G)) \Rightarrow \left(s \in \overline{K}\right).$$

Case 3 *:* $\sigma \in \Sigma_{ce}$

This case is proved by contradiction. Suppose $s'\sigma \in \mathcal{L}(S_{VLPGM2}/G)$, $s' \in \mathcal{L}(S_{VLPGM2}/G)$, $s'\sigma \notin \overline{K}$, $s' \in \overline{K}$.

Since $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \subseteq \Sigma_{cd}$ and $\Sigma_{cd} \cap \Sigma_{ce} = \emptyset$, $\Sigma_{ter}(\mathcal{L}_m(\mathcal{M})) \cap \Sigma_{ce} = \emptyset$.

Therefore, $\overline{K}$ is C&P co-observable w.r.t. $\mathcal{L}(G), \Sigma_{o1}, \Sigma_{ce1}, \ldots, \Sigma_{on}, \Sigma_{cen}$.

By definition of C&P co-observability,

$$\left(s' \in \overline{K}\right) \wedge \left(s'\sigma \notin \overline{K}\right) \Rightarrow \left[(\exists i \in I)\left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right].$$

First suppose σ was enabled by the *gmdec* step of VLP-GM2.

$NS_i[\Sigma_A \cap \Sigma_{uoi}]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s))$ for all Σ_A by Lemma 8.

$\Rightarrow NS_i\Sigma_{uoi}^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s))$.

$\Rightarrow PS_i^{gmdec} \subseteq P_i^{-1}(P_i(s))$.

$\Rightarrow PS_i^{gmdec}\sigma \subseteq P_i^{-1}(P_i(s'))\sigma$.

$\Rightarrow PS_i^{gmdec}\sigma \cap \overline{K} \subseteq P_i^{-1}(P_i(s'))\sigma \cap \overline{K}$.

$\Rightarrow \left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \Rightarrow \left(PS_i^{gmdec}\sigma \cap \overline{K} = \emptyset\right)$.

Therefore, $\left(s' \in \overline{K}\right) \wedge \left(s'\sigma \notin \overline{K}\right) \Rightarrow \left[\exists i \in I\left(PS_i^{gmdec}\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$.

Thus, it is obviously impossible that σ is enabled by any controller at the *gmdec* step of VLP-GM2.

Now suppose σ was enabled by the *AdditionalControlAction* step of VLP-GM2.

After $s\prime$ has occurred, for all controllers: $NS_i\left[\gamma_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i} \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s'))$.

Let $RS_i^+ = NS_i\left[\left(\gamma_i \cup \{\sigma\} \cup \Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G)$. The event σ needs to be tested to see if it is valid to be enabled by at least one controller.

$RS_i^+ \subseteq P_i^{-1}(P_i(s'))$ by the Lemma 8.

$\Rightarrow RS_i^+\sigma \subseteq P_i^{-1}(P_i(s'))\sigma$.

$\Rightarrow RS_i^+\sigma \cap \overline{K} \subseteq P_i^{-1}(P_i(s'))\sigma \cap \overline{K}$.

$\Rightarrow \left[\left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \Rightarrow \left(RS_i^+\sigma \cap \overline{K} = \emptyset\right)\right]$.

$\left(s' \in \overline{K}\right) \wedge \left(s'\sigma \notin \overline{K}\right) \Rightarrow \left[(\exists i \in I)\left(P_i^{-1}(P_i(s'))\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$.

$\Rightarrow \left[(\exists i \in I)\left(RS_i^+\sigma \cap \overline{K} = \emptyset\right) \wedge (\sigma \in \Sigma_{cei})\right]$, so σ is invalid to be enabled by at least one controller in the *AdditionalControlAction* step of VLP-GM2.

$\Rightarrow \sigma \notin S_{VLPGM2}(s')$.

$\Rightarrow s'\sigma \notin \mathcal{L}(S_{VLPGM2}/G)$.

This contradicts the assumption that $s'\sigma \in \mathcal{L}(S_{VLPGM2}/G)$, so if $\sigma \in \Sigma_{ce}$,

$$\left(s \in \mathcal{L}(S_{VLPGM2}/G)\right) \Rightarrow \left(s \in \overline{K}\right).$$

This completes the proof by induction. Therefore,

$$\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd} \Rightarrow \mathcal{L}(S_{VLPGM2}/G) \subseteq \overline{K}. \tag{5.12}$$

■

Now that a sufficient safety condition has been shown for VLP-GM2, it is demonstrated that VLP-GM2 always generates a language at least as large as the one generated by *gdec*.

Theorem 19 *Given that* $\mathcal{L}(S_{gdec}/G)$ *is safe,* $\mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{VLPGM2}/G)$.

Proof: This proof is based on induction on the length of the strings s to show that given $\mathcal{L}(S_{gdec}/G)$ is safe, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{VLPGM2}/G))$.

Base:

$|s| = 0$, or $s = \varepsilon$.

It is assumed that $(\varepsilon \in \mathcal{L}(S_{gdec}/G))$ and $(\varepsilon \in \mathcal{L}(S_{VLPGM2}/G))$ without loss of generality so that neither language is empty.

Therefore, $(\varepsilon \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (\varepsilon \in \mathcal{L}(S_{VLPGM2}/G))$.

Induction hypothesis:

$|s'| = n - 1$.

$(s' \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s' \in \mathcal{L}(S_{VLPGM2}/G))$.

Induction step:

Let $s = s'\sigma$ so $|s'| = n-1$ and $|s| = n$. The induction step is completed in several cases.

Case 1 *:* $\sigma \in \Sigma_{uc}$

$(\sigma \in \Sigma_{uc}) \Rightarrow (\sigma \in S_{gdec}(s')) \wedge (\sigma \in S_{VLPGM2}(s'))$.

Because $\mathcal{L}(S_{gdec}/G)$ is controllable,

$$(s' \in \mathcal{L}(S_{gdec}/G)) \wedge (s'\sigma \in \mathcal{L}(G)) \Leftrightarrow (s'\sigma \in \mathcal{L}(S_{gdec}/G)) .$$

Because $\mathcal{L}(S_{VLPGM2}/G)$ is controllable, $(s' \in \mathcal{L}(S_{VLPGM2}/G)) \wedge (s'\sigma \in \mathcal{L}(G)) \Leftrightarrow (s'\sigma \in \mathcal{L}(S_{VLPGM2}/G))$.

So, in this case, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{VLPGM2}/G))$.

Case 2 *:* $\sigma \in \Sigma_{cd}$

Previously, it was shown that $NS_i\left[\Sigma_A \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(G) \subseteq P_i^{-1}(P_i(s))$ for any Σ_A.

Let $PS_i^{gmdec} = NS_i\Sigma_{uoi}^* \cap \mathcal{L}(G)$.

$PS_i^{gmdec} \subseteq P_i^{-1}(P_i(s))$.

$\Rightarrow PS_i^{gmdec} \cap \overline{K} \subseteq P_i^{-1}(P_i(s')) \cap \overline{K}$.

$\Rightarrow \left(PS_i^{gmdec} \cap \overline{K}\right)\sigma \subseteq \left(P_i^{-1}(P_i(s')) \cap \overline{K}\right)\sigma$.

$\Rightarrow \left(PS_i^{gmdec} \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \left(P_i^{-1}(P_i(s')) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G)$.

$\Rightarrow \left(\left(\left(P_i^{-1}(P_i(s')) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \Rightarrow \left(\left(PS_i^{gmdec} \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right)\right)$.

$(s'\sigma \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (\sigma \in S_{gdec}(s'))$.

$\Rightarrow (\exists i \in I)\left[\left(\left(P_i^{-1}(P_i(s')) \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \wedge (\sigma \in \Sigma_{cdi})\right]$.

$\Rightarrow (\exists i \in I)\left[\left(\left(PS_i^{gmdec} \cap \overline{K}\right)\sigma \cap \mathcal{L}(G) \subseteq \overline{K}\right) \wedge (\sigma \in \Sigma_{cdi})\right]$.

$\Rightarrow (\exists i \in I)\left(\sigma \in \gamma_i^{gmdec}\left(NS_i\right)\right)$.

Furthermore,

$$(s' \in \mathcal{L}(S_{VLPGM2}/G)) \wedge \left(s' \in NS_i\left[\left(\gamma_{gmdec}\Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \wedge (s'\sigma \in \mathcal{L}(S_{gdec}/G)).$$

$\Rightarrow \left(s'\sigma \in NS_i\left[\left(\gamma_{gmdec}\Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(S/G)\right)$, so σ is not removed by the control filtering operation.

$\Rightarrow (\sigma \in S_{VLPGM2}(s'))$.

$\Rightarrow (s'\sigma \in \mathcal{L}(S_{VLPGM2}/G))$.

So, in this case, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{VLPGM2}/G))$.

Case 3 *:* $(\sigma \in \Sigma_{ce})$

After the system has been operating and the string s' has occurred, $\forall i \in I : s' \in NS_i\Sigma_{uoi}^* \cap \mathcal{L}(G)$, so $\forall i \in I : s'\sigma \in PS_i^{gmdec}\sigma$

Suppose $(s'\sigma \in \mathcal{L}(S_{gdec}/G))$.

$\Rightarrow s'\sigma \in \overline{K}$ by the safety condition.

$\Rightarrow \forall i \in I : s'\sigma \in PS_i^{gmdec}\sigma \cap \overline{K}$.

$\Rightarrow \forall i \in I : PS_i^{gmdec}\sigma \cap \overline{K} \neq \emptyset.$

$\Rightarrow \forall i \in I : \left(PS_i^{gmdec}\sigma \cap \overline{K} \neq \emptyset\right).$

$\Rightarrow (\forall i \in I)\left(\sigma \in \gamma_i^{gmdec}\left(NS_i\right)\right).$

Furthermore,

$$\left(s' \in \mathcal{L}(S_{VLPGM2}/G)\right) \wedge \left(s' \in NS_i\left[\left(\gamma_{gmdec}\Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^*\right) \wedge \left(s'\sigma \in \mathcal{L}(S_{gdec}/G)\right).$$

$\Rightarrow \left(s'\sigma \in NS_i\left[\left(\gamma_{gmdec}\Sigma_{cd}^{-i}\right) \cap \Sigma_{uoi}\right]^* \cap \mathcal{L}(S/G)\right)$, so σ is not removed by the control filtering operation.

$\Rightarrow (\sigma \in S_{VLPGM2}(s')).$

$\Rightarrow (s \in \mathcal{L}(S_{VLPGM2}/G)).$

So, in this case, $(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{VLPGM2}/G))$.

$(s \in \mathcal{L}(S_{gdec}/G)) \Rightarrow (s \in \mathcal{L}(S_{VLPGM2}/G))$ holds in all cases and this completes the induction proof that given $\mathcal{L}(S_{gdec}/G)$ is safe, $\mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{VLPGM2}/G)$. ■

This theorem can be used to prove another corollary very similar to one for gmdec. Notably, is $\overline{K}$ is controllable and co-observable then $\overline{K}$ will be achieved exactly by VLP-GM2.

Corollary 8 $\left(\overline{K}\right.$ *is controllable and co-observable*$\left.\right)$

$$\Rightarrow \left(\mathcal{L}(S_{VLPGM2}/G) = \overline{K}\right) \wedge \left(\mathcal{L}(S_{VLPGM2}/G) = \mathcal{L}(S_{gdec}/G)\right)$$

Proof: $\left(\overline{K}\right.$ is controllable and co-observable$\left.\right)$.

$\Rightarrow \left(\Sigma_{ter}\left(\mathcal{L}_m\left(\mathcal{M}\right)\right) \subseteq \Sigma_{cd}\right) \wedge \left(\mathcal{L}(S_{gdec}/G) = \overline{K}\right).$

$\Rightarrow \left(\mathcal{L}(S_{gdec}/G) \subseteq \mathcal{L}(S_{VLPGM2}/G) \subseteq \overline{K}\right) \wedge \left(\mathcal{L}(S_{gdec}/G) = \overline{K}\right).$

$\Rightarrow \left(\mathcal{L}(S_{VLPGM2}/G) = \overline{K}\right) \wedge \left(\mathcal{L}(S_{gdec}/G) = \mathcal{L}(S_{VLPGM2}/G)\right).$ ■

It should be apparent to the reader that many language properties are shared by *gmdec* and VLP-GM2 as would be expected by their similar constructions.

However, the languages generated by these two control protocols are not always equal, as would be expected. For one, *gmdec* is not afforded a degree of steering as with VLP-GM2 and furthermore, *gmdec* is never guaranteed to have a locally maximal control protocol like VLP-GM2. A natural question to ask when one considers the "local maximality" properties of VLP-GM2 is whether VLP-GM2 generates a language always at least as large as the language generated by *gmdec*. Sadly, this is not always the case. As is shown in the next three examples, $\mathcal{L}(S_{VLPGM2}/G)$ may be strictly larger than $\mathcal{L}(S_{gmdec}/G)$, $\mathcal{L}(S_{VLPGM2}/G)$ and $\mathcal{L}(S_{gmdec}/G)$ may be incomparable and $\mathcal{L}(S_{VLPGM2}/G)$ may be strictly smaller than $\mathcal{L}(S_{gmdec}/G)$.

Example 9 $\mathcal{L}(S_{gmdec}/G) \subset \mathcal{L}(S_{VLPGM2}/G)$

Remember Example 6 above. For this example $\mathcal{L}(S_{gmdec}/G) = \{\varepsilon, \phi\}$.

$\gamma_1^{VLPGM2}(\varepsilon) = \{\alpha, \phi\}$, $\gamma_2^{VLPGM2}(\varepsilon) = \{\phi\}$.

Therefore, $S_{VLPGM2}(\varepsilon) = \{\alpha, \phi\}$ *and* $\mathcal{L}(S_{VLPGM2}/G) = \{\varepsilon, \alpha, \phi\}$.

$\mathcal{L}(S_{VLPGM2}/G) = \{\varepsilon, \alpha, \phi\}$, $\mathcal{L}(S_{gmdec}/G) = \mathcal{L}(S_{gdec}/G) = \{\varepsilon, \phi\}$.

So, for this problem instance, $\mathcal{L}(S_{VLPGM2}/G)$ *is strictly larger than* $\mathcal{L}(S_{gmdec}/G)$.

Notice that in the previous example, if for all $i \in I$ $\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)$, but $\mathcal{L}(S_{VLPGM2}/G)$ is larger than $\mathcal{L}(S_{gmdec}/G)$ or $\mathcal{L}(S_{gdec}/G)$. This therefore implies that $\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)$ is not a sufficient condition for $\mathcal{L}(S_{VLPGM2}/G)$ to be equal to $\mathcal{L}(S_{gdec}/G)$. Because $\mathcal{L}(S_{VLPGM2}/G)$ has been shown to always be at least as large as $\mathcal{L}(S_{gdec}/G)$ when $\mathcal{L}(S_{gdec}/G)$ is safe, if $\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)$ holds for a problem instance, VLP-GM2 would be guaranteed to allow at least as much behavior as *gmdec*. Therefore VLP-GM2 would be a good algorithm to control a system when $\left[(\forall i \in I)\left(\Sigma_{uoi} \cap \Sigma_{cd}^{-i} = \Sigma_{uoi} \cap \Sigma_{cd}\right)\right]$.

Now, to move onto the next example where $\mathcal{L}(S_{VLPGM2}/G)$ and $\mathcal{L}(S_{gmdec}/G)$ are incomparable.

Example 10 *$\mathcal{L}(S_{VLPGM2}/G)$ and $\mathcal{L}(S_{gmdec}/G)$ are incomparable*

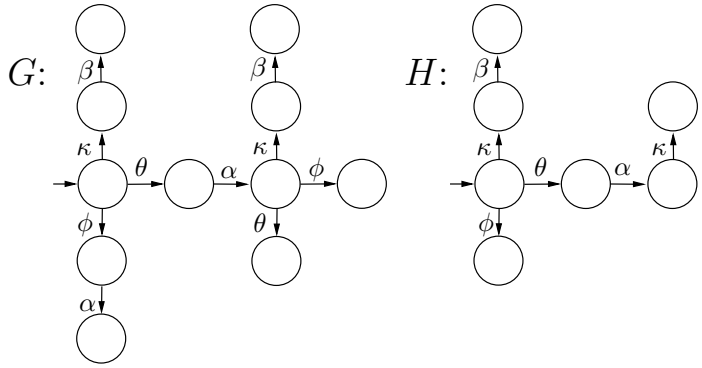

Figure 5.15: The uncontrolled system G and desired system H of Example 10.

Suppose the system G in Figure 5.15 is controlled by the decentralized control systems gmdec and VLP-GM2 with two controllers and the following properties:

$\Sigma_{uc} = \{\kappa\}$, $\Sigma_{cd} = \{\alpha, \beta, \phi, \theta\}$, $\Sigma_{cd} = \emptyset$, $\Sigma_{c1} = \{\alpha, \beta, \phi\}$, $\Sigma_{c2} = \{\theta\}$, $\Sigma_{o1} = \{\kappa\}$, $\Sigma_{o2} = \emptyset$.

$EList = [\phi, \alpha, \beta, \theta]$, $\overline{K} = \mathcal{L}(H)$.

For the gmdec control scheme:

$\gamma_1^{gmdec}(\varepsilon) = \{\kappa\}$.

$\gamma_1^{gmdec}(\kappa) = \{\beta, \kappa\}$.

$\gamma_2^{gmdec}(\varepsilon) = \{\kappa\}$.

Therefore, $\mathcal{L}(S_{gmdec}/G) = \{\varepsilon, \kappa, \kappa\beta\}$.

With the $VLPGM2$ control scheme:

$\gamma_1^{VLPGM2}(\varepsilon) = \{\kappa, \phi\}$.

$\gamma_1^{VLPGM2}(\kappa) = \{\kappa\}$.

$\gamma_2^{VLPGM2}(\varepsilon) = \{\kappa\}$.

Therefore, $\mathcal{L}(S_{VLPGM2}/G) = \{\varepsilon, \kappa, \phi\}$.

Obviously, $\mathcal{L}(S_{VLPGM2}/G)$ *and* $\mathcal{L}(S_{gmdec}/G)$ *are incomparable.*

An even more disappointing result with $\mathcal{L}(S_{VLPGM2}/G)$ is that there may be times when the language generated by VLP-GM2 may be strictly smaller than the language generated by *gmdec*. This possibility is demonstrated by the example below.

Example 11 $\mathcal{L}(S_{VLPGM2}/G) \subset \mathcal{L}(S_{gmdec}/G)$

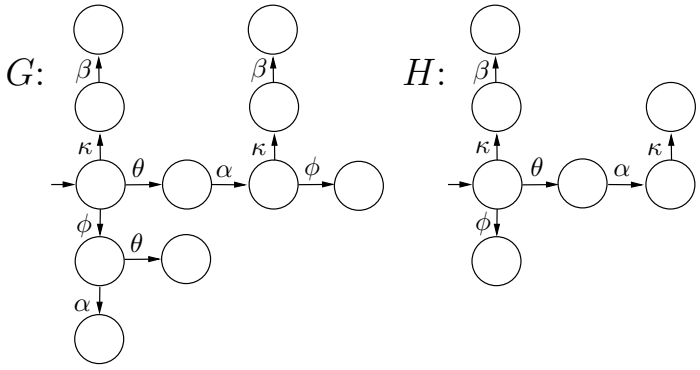

Figure 5.16: The uncontrolled system G and desired system H of Example 11.

Supposes the system G in Figure 5.16 is controlled by decentralized control system with two controllers and the following properties

$\Sigma_{uc} = \{\kappa\}$, $\Sigma_{cd} = \{\alpha, \beta, \phi, \theta\}$, $\Sigma_{cd} = \emptyset$, $\Sigma_{c1} = \{\alpha, \beta, \phi\}$, $\Sigma_{c2} = \{\theta\}$, $\Sigma_{o1} = \{\kappa\}$, $\Sigma_{o2} = \emptyset$.

The control priority is $EList = [\alpha, \phi, \beta, \theta]$.

For the gmdec control scheme:

$\gamma_1^{gmdec}(\varepsilon) = \{\kappa\}$.

$\gamma_1^{gmdec}(\kappa) = \{\beta, \kappa\}$.

$\gamma_2^{gmdec}(\varepsilon) = \{\kappa\}$.

Therefore, $\mathcal{L}(S_{gmdec}/G) = \{\varepsilon, \kappa, \kappa\beta\}$.

With the $VLPGM2$ *control scheme:*

$\gamma_1^{VLPGM2}(\varepsilon) = \{\kappa, \alpha\}$.

$\gamma_1^{VLPGM2}(\kappa) = \{\kappa\}$.

$\gamma_2^{VLPGM2}(\varepsilon) = \{\kappa\}$.

Therefore, $\mathcal{L}(S_{VLPGM2}/G) = \{\varepsilon, \kappa\}$ *and* $\mathcal{L}(S_{VLPGM2}/G) \subset \mathcal{L}(S_{gmdec}/G)$.

In this specific example, under the gmdec control scheme, controller 1 initially disables the α *event and can disregard any events that may occur after* α *could have occurred. This is why controller 1 can disregard the illegal* β *event. For both control schemes, neither controller communicates and controller 1 does not know that the* θ *events have been disabled by controller 2, so controller 1 does not know to disregard all events that could occur after* θ*. Therefore, when* $VLP-GM2$ *enables* α*, even though event* α *will never happen, controller 1 needs to account for the behavior of the illegal* β *event and hence disables* β *globally.*

The languages generated by VLP-GM2 may be smaller than the language generated by *gmdec* for the same reasons that VLP-GM may generate smaller languages than *gdec* or *gmdec*. Because the VLP-GM2 algorithm locally enables more events early on compared to *gmdec*, local controllers using VLP-GM2 have a larger state estimate than *gmdec* after more behavior has occurred. Because of its larger state estimate, VLP-GM2 is less likely to enable as many anti-permissive events as *gmdec* after system behavior has progressed. This could cause VLP-GM2 to generate a language strictly smaller than the language generated by *gmdec*.

5.8 Conclusion

In this chapter an improved state-estimator for general decentralized controllers is introduced. The state estimator improves upon previous work because it takes past control actions into account when calculating the set of likely current states.

This chapter also introduced several online decentralized control protocols that attempt to generate maximal safe behavior with respect to a given specification that may not be co-observable. These control protocols all have a testable sufficient safety condition that can be used to aid in selecting the proper sensors and actuators. That is, the sets of locally observable events $\{\Sigma_{o1}, \ldots, \Sigma_{on}\}$ and the partition of controllable events $\{\Sigma_{ce}, \Sigma_{cd}\}$ need to be chosen such that for the $\mathcal{M}$ automaton construction the only state transitions into the dump state are driven by events in Σ_{cd}. It would be interesting to find a method that minimizes the cost of this sensor/actuator selection. A simplified but computationally similar version of this selection problem is discussed in Chapter VI.

The *gmdec* control protocol is one of the new online protocols and is the result of combining this new state estimator with the *gdec* control scheme of [83]. The *gmdec* control method generates legal languages at least as large as the *gdec* controller when *gdec* is safe, but under some conditions *gmdec* produces languages equal to those generated by the *gdec* controller. In general, for the *gmdec* controller, allowing more events to be labelled as permissive does not imply a larger legal language will be generated.

Besides *gmdec*, another control scheme, VLP-GM, is introduced. VLP-GM is an iterative, greedy algorithm that produces maximal local control protocols. VLP-GM also has a sufficient safety condition similar to one for *gmdec* and *gdec*. The languages

generated by VLP-GM are in general incomparable with the languages generated by *gmdec* or *gdec*. It remains an open problem to develop sufficient conditions when a maximal controllable and co-observable sublanguage can be achieved by general decentralized control and to find sufficient conditions when a control scheme can generate a maximal controllable and co-observable sublanguage on-line.

To attempt to make up for some of the deficiencies of VLP-GM, the VLP-GM2 algorithm was introduced. The VLP-GM2 algorithm combines parts of *gmdec* with VLP-GM to create a "steerable" control algorithm that always performs at least as well as *gdec*. VLP-GM2 has many beneficial properties similar to *gmdec*, but there are times when *gmdec* could generate larger languages than VLP-GM2. It remains an open problem to find when the languages generated by VLP-GM2 are at least as large as *gmdec*.

CHAPTER VI

APPROXIMATING MINIMAL CARDINALITY SENSOR SELECTIONS

6.1 Chapter Overview

This chapter discusses the approximation properties of a minimal cardinality sensor selection that is sufficient for a specification language to be observable with respect to a system. It is assumed that the specification language and system are given as finite state automata. This sensor selection problem most likely does not have provably accurate polynomial time solution approximation algorithms. A method is shown to convert this minimal cardinality sensor selection problem into a type of directed-graph *st*-cut problem. Several polynomial time heuristic algorithms are shown for approximating solutions to this problem. It is then shown how these methods can be used to solve a similar communicating controller problem.

6.2 Approximation Problems and Discrete-Event Systems

When synthesizing a controller for a system to achieve a specification, a control engineer may be able to choose the set of sensors used by the controller. In the framework of supervisory control, a set $\Sigma_o \subseteq \Sigma$ is called a *sufficient sensor selection* with respect to G, H and Σ_c if $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c.

A sufficient sensor selection is also called an observability set. Therefore, if Σ_o is a sufficient sensor selection and $\mathcal{L}(H)$ is controllable with respect to $\mathcal{L}(G)$ and Σ_c, then there exists an admissible controller S such that $\mathcal{L}(S/G) = \mathcal{L}(H)$. Due to economic reasons such as the cost of purchasing and installing sensors, the number of sensors used by a controller should be kept to a minimum as long as the sensor selection is sufficient. The problem of finding this minimal cardinality sufficient sensor selection is called the *minimal cardinality sensor selection problem.*

Problem 1 Minimal Cardinality Sensor Selection: *Given G, H and $\Sigma_c \subseteq \Sigma$, find a sufficient sensor selection Σ_o^{min} such that for any other sufficient sensor selection Σ_o, $|\Sigma_o^{min}| \leq |\Sigma_o|$.*

If there were some cost associated with selecting a sensor (as specified by some cost function $cost : \Sigma \rightarrow \Re^+$), a generalized version of Problem 1 would be to find a sufficient sensor selection with minimal cost. Although this chapter exclusively discusses the cardinality minimization problem, the methods shown here for dealing with this problem can be easily extended to the sensor cost minimization problem.

A simple example of the sensor selection problem is now shown.

Example 12 *Consider the system and specification seen in Figure 6.1. Suppose that $\Sigma_c = \{\alpha\}$. There are several sufficient sensor selections for this specification with respect to the given system: $\{\alpha\}, \{\beta, \gamma\}, \{\gamma, \lambda\}, \{\beta, \lambda\}$. However, $\{\alpha\}$ is the minimal cardinality sufficient sensor selection.*

Now suppose that the cost of using the sensors is non-uniform, such that

$$cost(\alpha) = 7, cost(\beta) = 4, cost(\gamma) = 5, cost(\lambda) = 2.$$

With these sensor costs, the minimal cost sensor selection that makes the specification observable is $\{\beta, \lambda\}$.

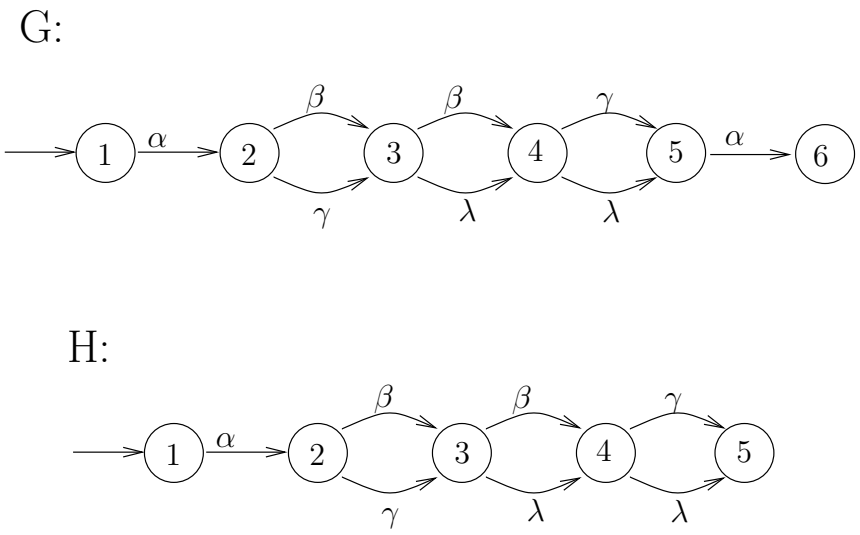

Figure 6.1: The system G and specification H of Example 12.

Because of the NP-completeness of Problem 1 the minimal cardinality sensor selection can not always be found in a computationally efficient manner [84]. However, a sufficient sensor selection Σ_o may still need to be found reasonably efficiently such that the cardinality of Σ_o is as close to $|\Sigma_o^{min}|$ as possible. Fortunately, some NP-complete minimization problems have fairly accurate polynomial time approximation algorithms [2, 77]. This means sufficient and approximate solutions can be found for many computationally difficult problems in a reasonable amount of time. However, not all NP-complete minimization problems are believed to have this property [2, 77].

To better quantify what is meant by an approximation to Problem 1, suppose P is the set of instances of Problem 1. Let $p \in P$ be a specific problem instance corresponding to the system G, the specification H and a set of controllable events Σ_c. Let $\Sigma_o^{min}(p)$ denote the solution of this problem instance. When an approximation algorithm A is given an input p, A returns $\Sigma_o^A(p)$, a sufficient sensor selection with respect to G, H and Σ_c. The measure the utility of the approximation $\Sigma_o^A(p)$ is the

ratio $|\Sigma_o^A(p)|/|\Sigma_o^{min}(p)|$ and this ratio should ideally be as small as possible. Problem 1 is said to have *r-approximation* algorithm if

$$\forall p \in P, |\Sigma_o^A(p)|/|\Sigma_o^{min}(p)| \leq r. \tag{6.1}$$

This r-approximation notation also holds for other approximation problems. For a deeper discussion of these topics, see [2]. An important subclass of computation problems in the NP-complete class is the APX problem class formally defined below.

Definition 15 The APX Problem Class: *[2] A problem P is in APX if it is NP-complete and there is some constant $r \in \Re, r \geq 1$ such that for all problem instances there exists a polynomial time r-approximate algorithm for P.*

6.3 The Complexity of Minimal Cardinality Sensor Selection Approximations

It is now shown that the minimal cardinality sensor selection problem is not in APX using the minimal set cover problem. The minimal set covering problem is a fundamental problem in computer science used to show the computational difficulty of many other problems. For this problem a set $S = \{\gamma_1, \ldots, \gamma_n\}$ is given along with a set of subsets $\mathcal{C} = \{C_1, \ldots, C_m\} \subseteq 2^S$, and the problem is to find a set covering $\mathcal{C}_{min} = \{C_{i_1}, \ldots, C_{i_k}\} \subseteq \mathcal{C}$ such that $C_{i_1} \cup \cdots \cup C_{i_k} = S$ and for any other covering subset $\mathcal{C}' = \{C_{k_1}, \ldots, C_{k_l}\} \subseteq \mathcal{C}$ such that $C_{k_1} \cup \cdots \cup C_{k_l} = S$, $|\mathcal{C}_{min}| \leq |\mathcal{C}'|$. The set $\mathcal{C}_{min}$ is called the minimal set covering. It is known that the minimal set covering problem is NP-complete [18], and this problem is not in APX [2]. This result can be used to show that minimal cardinality sensor selection problem is also not in APX.

Theorem 20 *The minimal cardinality sensor selection problem is not in APX.*

Proof: This theorem is demonstrated using a proof by contradiction. Suppose the minimal cardinality sensor selection problem is in APX. Then there is an algorithm $\mathcal{A}_o$ that when given an instance of the minimal cardinality sensor selection problem, returns an approximation of the minimal cardinality sensor selection whose cardinality is within a constant ratio r of the cardinality of the minimal cardinality sensor selection in polynomial time. It is now shown how algorithm $\mathcal{A}_o$ can be used to construct an algorithm $\mathcal{A}_{sc}$ that when given an instance of the set cover problem, returns an approximation of the minimal set cover whose cardinality is within a constant ratio r of the cardinality of the minimal set cover in polynomial time.

Given an instance of the set cover problem, i.e., a set $S = \{\gamma_1, \ldots, \gamma_n\}$ and a set of subsets $\mathcal{C} = \{C_1, \ldots, C_m\} \subseteq 2^S$, assume without loss of generality that $C_1 \cup \cdots \cup C_m = S$. Put an arbitrary ordering on the subsets of S such that $C_1 < \ldots < C_m$. For an element γ_i let $\mathcal{C}_i = \{C_1^i, \ldots, C_{j_i}^i\}$ represent the subsets that contain γ_i. That is, $\forall C_k^i \in \mathcal{C}_i, \gamma_i \in C_k^i$. Furthermore, assume that $C_1^i \leq \ldots \leq C_{j_i}^i$. Now, for the sets $\{\mathcal{C}_1, \ldots, \mathcal{C}_n\}$ construct the automaton G seen in Figure 6.2 where the ith branch of the initial state represents the ordered list of sets that contain γ_i and α is a symbol not already used.

The G automaton can be constructed in polynomial time with respect to the size of the encoding of the set cover problem, and therefore G also has a polynomial number of states. Note that the automaton G may be nondeterministic, but it can be converted to a deterministic automaton accepting the same language by iteratively merging state transitions with the same label at the same parent node until the automaton is deterministic. That is, if $x_1 \overset{C_j^k}{\mapsto}_G x_2$ and $x_1 \overset{C_j^l}{\mapsto}_G x_3$ such that $C_j^k = C_j^l$ and $x_2 \neq x_3$, then merge the states x_2 and x_3 and remove the $x_1 \overset{C_j^l}{\mapsto}_G x_3$ transition. Because the number of states is bounded by a polynomial, this determinization procedure will

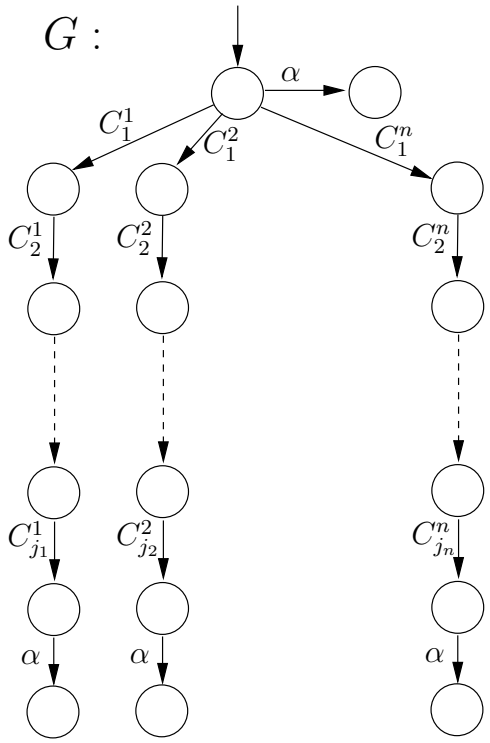

Figure 6.2: The G automaton used in proof of Theorem 20.

halt in a polynomial amount of time.

Let H be a copy of automaton G with all occurrences of α removed except for the α at the initial state. Let G be the system automaton, let H the specification automaton and let $\Sigma_c = \{\alpha\}$.

Suppose there exists a string of events $C_1^i C_2^i \dots C_{j_i}^i$ such that no events in this string are observed. Then the system is unobservable because a controller would not know to disable the α event after $C_1^i C_2^i \dots C_{j_i}^i$ occurred. In this case $\mathcal{L}(H)$ is not observable with respect to $\mathcal{L}(G)$, Σ_c and Σ_o where $\Sigma_o = \{C_o^1, \dots, C_o^p\}$ and there exists some event γ_i such that for all $C_o^q \in \Sigma_o$, $\gamma_i \notin C_o^q$. Therefore $\{C_o^1, \dots, C_o^p\} = \mathcal{C}'$ does not form set cover for S.

Similarly, if every string of events $C_1^i C_2^i \dots C_{j_i}^i$ contains at least one event that is observed, then a controller would always know to disable α after the string of events $C_1^i C_2^i \dots C_{j_i}^i$ occurs. Therefore, for every event γ_i, there must be an event in $C_1^i C_2^i \dots C_{j_i}^i$ that is observable. Hence, if $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_c and Σ_o where $\Sigma_o = \{C_o^1, \dots, C_o^p\}$, then $\{C_o^1, \dots, C_o^p\} = \mathcal{C}'$ forms a set cover for S. This is because for any γ_i, there exists a C_o^q such that $\gamma_i \in C_o^q$.

Then, for the given construction of G and H, a set of events $\Sigma_o \in 2^\Sigma$ is a sufficient sensor selection with respect to G, H and Σ_c if and only if the corresponding set $\mathcal{C}'$ is a set cover for S. Furthermore, the cardinality of the minimal sensor selection is equal to the cardinality of the corresponding minimal set cover. Therefore, $|\Sigma_o^{min}| = |\mathcal{C}_{min}|$.

Suppose algorithm $\mathcal{A}_o$ is run with the construction of G, H and Σ_c and the observability set Σ_o' is returned. It is known that $|\Sigma_o'|/|\Sigma_o^{min}| \leq r$ because of the assumption on $\mathcal{A}_o$. The set Σ_o' can then be used to calculate a set $\mathcal{C}'$ using the construction above such that $|\mathcal{C}'|/|\mathcal{C}_{min}| \leq r$. The problem instance G, H and Σ_c can be constructed in polynomial time and it was assumed the algorithm $\mathcal{A}_o$ can be run in polynomial time. This implies there exists a polynomial time algorithm to find an approximation to the minimal set cover such that the ratio of the cardinality of the approximation to the cardinality of the minimal set cover is bound by a constant r. This implies by definition that the minimal set cover problem is in APX, which forms a contradiction. Therefore, there does not exist an algorithm $\mathcal{A}_o$ that when given an instance of the minimal cardinality sensor selection problem, returns a set whose cardinality is within a constant ratio r of the cardinality of the minimal cardinality sensor selection in polynomial time unless P=NP. Finally, the minimal cardinality sensor selection problem is not in APX. ■

This theorem shows that the minimal cardinality sensor selection problem is difficult to approximate in a time efficient manner. It was also shown in [33] that the sensor selection problem admits no $2^{\log^{(1-\epsilon)} n}$ approximation for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{\text{polylog } n})$. It follows from this result that if solutions to the sensor selection problem can be found with better than a $2^{\log^{(1-\epsilon)} n}$-approximation, then a method has been found for solving NP-complete problems in quasi-polynomial time. This lower bound on the ability to approximate minimal sensor selections is generally considered to be a very poor lower bound in the computer science community because as ϵ approaches 0, then $2^{\log^{(1-\epsilon)} n}$ approaches n. However, because of the fundamental importance of this problem, usable methods need to be developed to approximate the minimal cardinality sensor selections. This prompts the algorithms presented in the rest of this chapter for approximating solutions to the minimal cardinality sensor selection problem.

6.4 A Randomized Descent Approximation Algorithm

A randomized descent algorithm for approximating minimal cardinality sensor selections is now shown. Consider the set of system events Σ and its power set 2^{Σ}. The process of finding the minimum cardinality sufficient sensor selection $\Sigma_o^{min} \subseteq \Sigma$ is effectively a search over the power set of Σ, 2^{Σ}. An interesting property of observable systems is that for any set of observable events $\Sigma_o \subseteq \Sigma$ such that $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c, then for any Σ_o' such that $\Sigma_o \subseteq \Sigma_o' \subseteq \Sigma$, $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o' and Σ_c and $|\Sigma_o| \leq |\Sigma_o'|$. That is, all supersets of sufficient sensor selections are also sufficient sensor selections.

Given a set of events Σ_o such that $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c, it may be possible that there does not exist an event $\sigma \in \Sigma_o$ such that $\mathcal{L}(H)$

is observable with respect to $\mathcal{L}(G)$, $\Sigma_o \setminus \{\sigma\}$ and Σ_c. In this case Σ_o is called a locally minimal sufficient sensor selection. For a given system there may possibly be many locally minimal sufficient sensor selections, but these locally minimal sufficient sensor selections may not all be minimal cardinality sufficient sensor selections. Consider the following example.

Example 13 *Recall the system in Example 12. For this system and specification,* $\Sigma_o = \{\beta, \gamma\}$ *is a sufficient sensor selection if* $\Sigma_c = \{\alpha\}$ *and it is locally minimal, but the minimal cardinality sensor selection is* $\Sigma_o = \{\alpha\}$ *if* $\Sigma_c = \{\alpha\}$.

Consider how the power set 2^{Σ} forms a lattice with respect to the partial ordering of the subsets of Σ. It is assumed that $\mathcal{L}(H)$ is never observable with respect to $\mathcal{L}(G)$, $\emptyset$ and Σ_c (i.e. the trivial case). Let $n = |\Sigma|$. Therefore, for every path on the lattice formed by 2^{Σ} from $\Sigma \supseteq \Sigma_1 \supseteq \cdots \supseteq \Sigma_n \supseteq \emptyset$, there is a boundary observability set Σ_i such that for any $\Sigma_i' \supseteq \Sigma_i$, $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_i' and Σ_c and for any $\Sigma_{i+1}' \subseteq \Sigma_{i+1}$, $\mathcal{L}(H)$ is not observable with respect to $\mathcal{L}(G)$, Σ_{i+1}' and Σ_c.

Therefore, for the sets of all paths $\Sigma \supseteq \Sigma_1 \supseteq \cdots \supseteq \Sigma_n \supseteq \emptyset$ in the lattice formed by 2^{Σ}, the set of boundary sets of these paths forms a frontier between the sets of events that make the system observable and the sets of events that make the system unobservable. The minimum cost observability set is somewhere on this boundary. Note that not all members of this set of boundaries are locally minimal sensor selections.

Example 14 *Figure 6.3 shows that lattice constructed for the system and specification of Example 12. All of the sensor selections above the dotted line are sufficient, but all of the sensor selections below the line are deficient.*

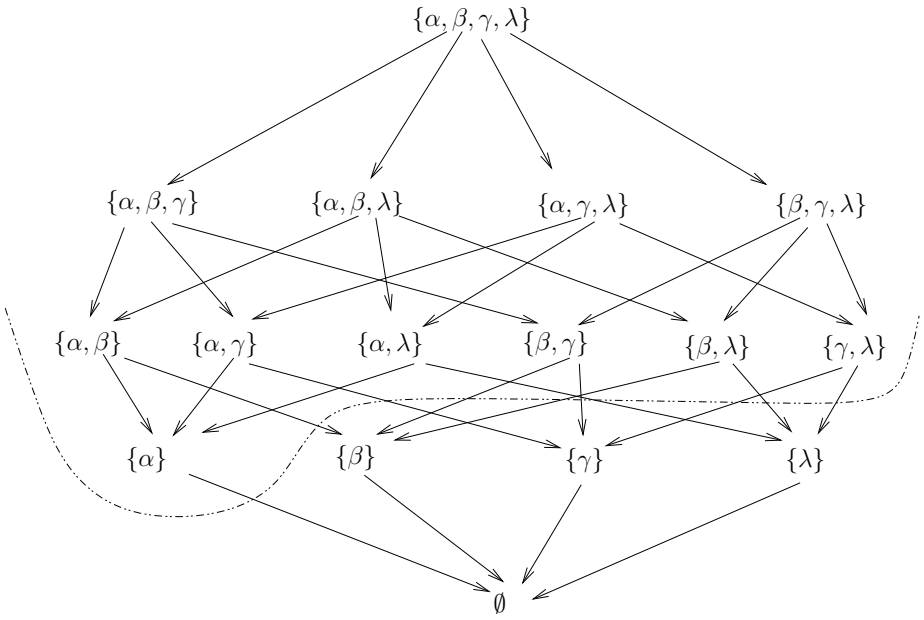

Figure 6.3: The sensor selection lattice of Example 14.

Finding a locally minimal sensor selection for a system is fairly easy. This could be done by initializing the set of observable events to be Σ, and events could iteratively be removed from the observability set until no more events could be removed without the system becoming unobservable. This is exactly what is done in the following randomized algorithm.

Algorithm 7 Randomized Local Minima Search Algorithm (RanLocMin):

Input: G, H, Σ_c.

$\Sigma_{test} \Leftarrow \Sigma$;

$\Sigma_o \Leftarrow \Sigma$;

Repeat:

{

Randomly remove $\sigma \in \Sigma_{test}$ *from* Σ_{test} *and* Σ_o.

If $\mathcal{L}(G)$ *is not observable w.r.t.* $\mathcal{L}(H)$, Σ_c *and* Σ_o, *return* σ *to* Σ_o.

}

Until $\Sigma_{test} = \emptyset$.

Return: Σ_o.

Starting with an observability set Σ, the algorithm randomly chooses an observed event σ and tries making it unobservable. It is assumed that the events have a uniform probability of being chosen on any iteration of the algorithm. If after removing σ the system is no longer observable, then σ needs to be observed no matter what other events are removed. This is because if Σ_a is not an observability set, then any $\Sigma'_a \subseteq \Sigma_a$ can not be an observability set. If after removing σ the system is still observable, then σ is permanently removed from the observability set. This algorithm is iterated until no events can be removed from the observability set. Because no events can be removed from the returned observability set without violating system observability, that observability set is a local minimum.

Suppose Algorithm 7 finds the global minimum with probability p, which may be quite low, but it would be desired to find the global minimum with probability $r \in (p, 1)$. The probability of finding the global minimum using Algorithm 7 could be boosted through iteration as in Algorithm 8 below.

Algorithm 8 Iterated Randomized Minima Search Algorithm (ItRanMin):

Input: G, H, Σ_c.

$\Sigma_o \Leftarrow \Sigma$

Repeat k *times:*

{

$\Sigma_f \Leftarrow RanLocMin(G, H, \Sigma_c)$

if $|\Sigma_f| \leq |\Sigma_o|$, *then* $\Sigma_o \Leftarrow \Sigma_f$

}

Return: Σ_o.

Algorithm 8 makes k calls to Algorithm 7 and hence takes k times as long as Algorithm 7. This prompts the question of what value of k should be chosen such that a global minima is found with probability at least r using Algorithm 8? It would be helpful to have k as small as possible such that a minimal cardinality sensor selection is found with high probability.

If Algorithm 7 finds a global minimum with probability p, this algorithm does not find a global minimum with probability $(1-p)$. Hence, over the k trials of Algorithm 8, the probability that a global minimum is not found is $(1-p)^k$. It is well known that $(1-p)^k \leq \exp(-pk)$ based on the convex analysis that $\exp(-x) - 1 + x \geq 0 \ \forall x \in [0, \infty)$. So, if it is desired that the iterated randomized local minima search algorithm returns a non-global minima with probability at most $(1-p)^k \leq (1-r)$, then k needs to be found such that $\exp(-pk) = (1-r)$, or $k = \frac{\ln\left(\frac{1}{(1-r)}\right)}{p}$.

Unfortunately, p may be very small in the worst case. Consider the situation when $|\Sigma| = n$ and n is even. As with the system in Figure 6.3, for all $\Sigma_o \subset \Sigma$ such that $|\Sigma_o| \geq n/2$, Σ_o is a sufficient sensor selection and no other sensor selections are sufficient except for exactly one subset, $\Sigma_o^{min} \subseteq \Sigma$ where $|\Sigma_o^{min}| = n - 1$. Therefore, for this example, during the operation of Algorithm 7, any set $\Sigma_a \subset \Sigma$ such that $|\Sigma_a| = n/2$ can be selected as a sensor selection by this algorithm.

Notice that for this example there are exactly $(n/2 + 1)$ sets $\Sigma_o' \subset \Sigma$ such that $\Sigma_o^{min} \subset \Sigma_o'$ and $|\Sigma_o'| = n/2$. Therefore, a set Σ_o' as discussed above has probability $\frac{n/2+1}{\binom{n}{n/2}}$ of being used as a sensor selection in Algorithm 7.

Also notice due to the construction of Algorithm 7, for this example if during the operation of Algorithm 7 one of these sets Σ_o' is selected as a sensor selection,

then Algorithm 7 will necessarily return Σ_o^{min} upon termination. Furthermore, if Algorithm 7 does not choose one of these sets Σ_o' as a sensor selection, then Algorithm 7 will not return Σ_o^{min} upon termination. This means that Σ_o^{min} has probability $\frac{n/2+1}{\binom{n}{n/2}}$ of being chosen for this example by Algorithm 7.

Therefore, for this construction, the probability that Σ_o^{min} is found by the randomized local minimum search algorithm is $\frac{n/2+1}{\binom{n}{n/2}}$. Therefore, for n reasonably large:

$$\begin{aligned}
p &= \frac{n/2+1}{\binom{n}{n/2}} \\
&\leq \frac{n}{\binom{n}{n/2}} \\
&= \frac{n}{\left(\frac{n!}{(n/2)!(n/2)!}\right)} \\
&= \frac{n}{n\left(\frac{n-1}{n/2}\cdots\frac{n-i}{(n/2+1)-i}\cdots\frac{n/2+1}{2}\right)} \\
&= \left(\frac{n/2}{n-1}\cdots\frac{(n/2+1)-i}{n-i}\cdots\frac{2}{n/2+1}\right) \\
&\leq (1/2)^{n/2-1}
\end{aligned}$$

Therefore, p is exponential in $-n$ in the worst case and therefore in the worst case, k would have to be exponentially large in order to obtain an arbitrarily high probability of finding a minimum cardinality sensor selection with Algorithm 8. However, worst-case scenarios are rather unusual and this algorithm helps us gain insight into the problem. The more iterations that are taken in Algorithm 8 the closer of an approximation is obtained of the global minimum. In the hypothetical example above with the nearly flat frontier, a very close approximation to the global minimum is obtained after one iteration of the randomized search algorithm.

6.5 The Graph Cutting Problem

Observability can be tested using a method similar to the $\mathcal{M}$ construction seen in Appendix A. This method was originally outlined for the case of observability in [74] (without explicitly giving the $\mathcal{M}$ automaton construction), but a modified version is shown below that can be used to convert sensor selection problems into a special type of graph cutting problem called an "edge colored directed-graph *st*-cut problem".

For this edge colored directed-graph *st*-cut problem, assume an edge-colored directed graph $D = (V, A, C)$ where V is a set of vertices, $A \subseteq V \times V$ are directed edges and $C = \{c_1, \ldots, c_p\}$ is the set of colors. Each edge is assigned a color in C. The directed graph in Figure 6.4 is an example of an edge colored directed graph where the edges are assigned colors $\{\alpha, \beta, \gamma, lambda\}$.

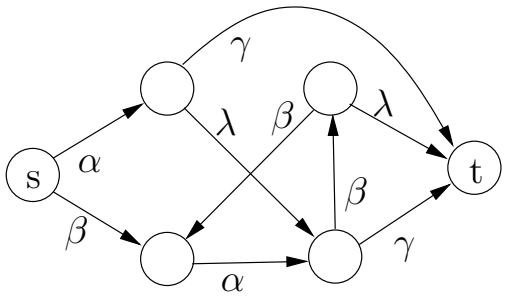

Figure 6.4: An example of an edge colored directed graph.

Let A_i be the edges having color c_i. Given $I \subseteq C$, let $A_I = \cup_{c_i \in I} A_i$. For two nodes $s, t \in V$ such that there is a path of directed edges from s to t, then I is a colored *st*-cut if $(V, (A \setminus A_I), C)$ has no path from s to t. As seen in Figure 6.5, $I = \{\beta, \gamma\}$ is a colored *st*-cut for the graph in Figure 6.4.

The minimal colored cut problem for edge colored directed graphs can now be

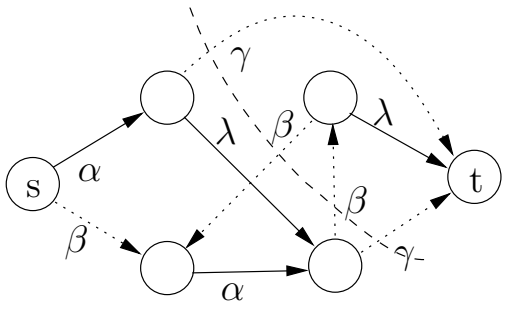

Figure 6.5: The colored cut $I = \{\beta, \gamma\}$ for the graph in Figure 6.4.

defined.

Problem 2 Minimal Colored Cut: *For an edge colored directed graph $D = (V, A, C)$ and two vertices, $s, t \in V$, find a colored st-cut $I^{min} \subseteq C$ such that for any other colored st-cut $I \subseteq C$, $|I^{min}| \leq |I|$.*

It is now shown how to convert an instance of a colored cut problem into an instance of a sensor selection problem. Suppose an edge colored directed graph $D = (V, A, C)$ and two vertices s, t are given. A system G, specification H and controllable event set Σ_c are now constructed from D. For the colors $C = \{c_1, \ldots, c_p\}$, let the event set Σ include a corresponding set of events $\{\sigma_1, \ldots, \sigma_p\}$. such that color c_i is paired with event σ_i. Let γ be another event and define $\Sigma = \{\sigma_1, \ldots, \sigma_p, \gamma\}$. Also define $X^G = V \cup \{s', s'', t'\}$ where s', s'', t' are states not in V. Let $x_0^G = s$. To define the state transition function, let v_1, v_2 be any vertices except s. If $(v_1, v_2) \in A_i$, then $v_1 \overset{\sigma_i}{\mapsto}_G v_2$. If $(s, v_2) \in A_i$, then $s \overset{\sigma_i}{\mapsto}_G v_2$ and $s'' \overset{\sigma_i}{\mapsto}_G v_2$. If $(v_1, s) \in A_i$, then $v_1 \overset{\sigma_i}{\mapsto}_G s''$. For simplicity it is assumed that $(s, s) \notin A$. Also, transitions are added such that $s \overset{\gamma}{\mapsto}_G s'$ and $t \overset{\gamma}{\mapsto}_G t'$. Let H be a copy of G except that $\delta^H(t, \gamma)$ is undefined. Let $\Sigma_c = \{\gamma\}$.

An example of such a system construction for converting a directed graph D to a system and specification G and H is given in Figure 6.6.

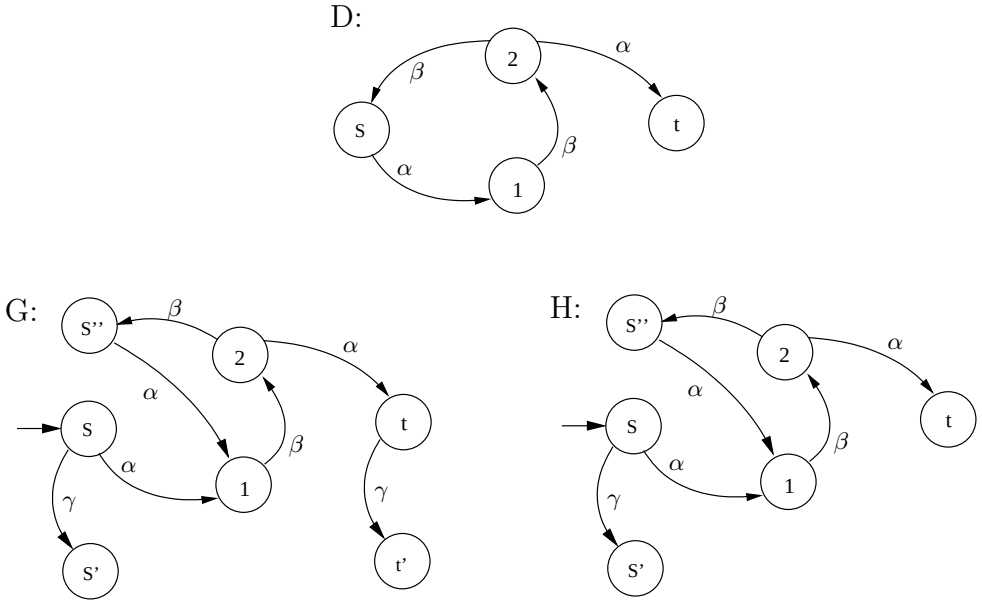

Figure 6.6: A directed graph D and the systems G and H constructed from it.

In the system above, γ must be enabled at s and be disabled at t. There is a control conflict if there is a path in G from s to t where no event is observed. Therefore, as system behavior progresses, if any event is observed, then γ can be disabled. Hence, a set of colors $I = \{c_a, \ldots, c_z\}$ is a colored cut for D if and only if selecting the sensors $\{\sigma_a, \ldots, \sigma_z\}$ corresponding to I makes the system observable. Therefore any approximation algorithm for the sensor selection problem can also be used with the same absolute effectiveness for the colored cut problem.

The converse construction is now shown to convert an instance of Problem 1 to an instance of Problem 2. Suppose $H = (X^H, x_0^H, \Sigma, \delta^H)$, $G = (X^G, x_0^G, \Sigma, \delta^G)$, Σ_o and Σ_c are given and it is desired to test if $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c. This is done by constructing an automaton $\mathcal{M}_{\Sigma_o}(X^{\mathcal{M}_{\Sigma_o}}, x_0^{\mathcal{M}_{\Sigma_o}}, \Sigma^{\mathcal{M}_{\Sigma_o}}, \delta^{\mathcal{M}_{\Sigma_o}})$

that is a modification of $\mathcal{M}$ automaton method for testing observability and co-observability in [67, 74]. The $\mathcal{M}_{\Sigma_o}$ automaton is effectively a nondeterministic simulation of estimates an observer may make of unobservable system behavior with respect to a specification based on imperfect predictions of occurrences of unobservable events ($\Sigma \setminus \Sigma_o$) in the system.

Let Σ' be a copy of the event set Σ where for every event $\sigma \in \Sigma$, there is a corresponding event $\sigma' \in \Sigma'$. The following are then defined, $X^{\mathcal{M}_{\Sigma_o}} := X^H \times X^H \times X^G \cup \{d\}$,$x_0^{\mathcal{M}_{\Sigma_o}} := (x_0^H, x_0^H, x_0^G)$and $\Sigma^{\mathcal{M}_{\Sigma_o}} := \Sigma \cup \Sigma'$.

Suppose a string of events s has been simulated to occur in the system G by $\mathcal{M}_{\Sigma_o}$ and the simulation is at state $(x_1, x_2, x_3) \in X^{\mathcal{M}_{\Sigma_o}}$. State x_3 represents the true state of the system G and x_2 represents the corresponding state of the specification H after s has occurred. States x_2 and x_3 always update simultaneously. However, as was stated above, the observer attempts to predict the occurrence of system events and the state x_1 represents the observer's estimate of the possible state of the specification based on imperfect predictions of the simulated system behavior s.

Furthermore, at state (x_1, x_2, x_3) of the simulation, if an event σ is correctly predicted by the observer in the simulation, there is a transition from (x_1, x_2, x_3) labelled by σ where all of the component states of (x_1, x_2, x_3) update on the occurrence of σ according to the transition rules of H, H and G respectively. A correct prediction may occur for either observable or unobservable events.

However, if an event σ occurs in the system that is not predicted correctly by the observer in the simulation, there is a transition from (x_1, x_2, x_3) labelled by σ' where the x_2, x_3 component states of (x_1, x_2, x_3) update on the occurrence of σ according to the transition rules of H and G respectively. Similarly, if an event σ does not occur in the system but is incorrectly predicted to occur by the observer in the simulation,

there is a transition from (x_1, x_2, x_3) labelled by σ' where the x_1 component state of (x_1, x_2, x_3) updates on the occurrence of σ according to the transition rules of H. Therefore, the $\mathcal{M}_{\Sigma_o}$ simulation is nondeterministic in that unobservable event occurrences cannot be perfectly predicted. In correct predictions only occur for unobservable events.

If the $\mathcal{M}_{\Sigma_o}$ simulation ever reaches a composed state where the observer believes the occurrence of a controllable event is allowed by the specification due to the properties of state x_1, but in reality it is not due to x_2, yet still possible due to x_3, then there is a control conflict. This possibility is captured by the $(*)$ condition such that if the simulation could reach a state (x_1, x_2, x_3) where $(*)$ holds, then illegal controllable behavior could occur in the system without an observer being able to resolve the control conflict.

$$\left.\begin{array}{ll} \delta^H(x_1, \sigma) & \text{is defined if } \sigma \in \Sigma_c \\ \delta^H(x_2, \sigma) & \text{is not defined} \\ \delta^G(x_3, \sigma) & \text{is defined} \end{array}\right\} \qquad (*)$$

The nondeterministic transition relation $\delta^{\mathcal{M}_{\Sigma_o}}$ is now more formally defined as follows.

For $\sigma \notin \Sigma_o$ and its Σ' equivalent, σ',

$$\delta^{\mathcal{M}_{\Sigma_o}}((x_1, x_2, x_3), \sigma') = \left\{ \begin{array}{l} (\delta^H(x_1, \sigma), x_2, x_3) \\ (x_1, \delta^H(x_2, \sigma), \delta^G(x_3, \sigma)) \end{array} \right\}.$$

For $\sigma \in \Sigma$,

$$\delta^{\mathcal{M}_{\Sigma_o}}((x_1, x_2, x_3), \sigma) = \left\{ \begin{array}{ll} (\delta^H(x_1, \sigma), \delta^H(x_2, \sigma), \delta^G(x_3, \sigma)) & \\ d & \text{if } (*) \end{array} \right\}.$$

For $\sigma \in \Sigma$, $\delta^{\mathcal{M}_{\Sigma_o}}(d, \sigma)$ is undefined. The $\mathcal{M}_{\Sigma_o}$ automaton here is modified from the original in [74] in that Σ' transitions replace some Σ transitions. These $\sigma' \in \Sigma'$ transitions correspond to transitions that would not exist if $\sigma \in \Sigma \setminus \Sigma_o$ were to be made observable. The $\mathcal{M}_{\Sigma_o}$ automaton construction prompts the following proposition which follows from the results in [74].

Proposition 8 *The state d is reachable in $\mathcal{M}_{\Sigma_o}$ if and only if $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c.*

An example is now given of an $\mathcal{M}_{\Sigma_o}$ automaton construction.

Example 15 *Recall the system and specification shown in Example 12. The $\mathcal{M}_\emptyset$ automaton constructed for this system and specification with $\Sigma_c = \{alpha\}$ can be seen in Figure 6.7.*

In the $\mathcal{M}_{\Sigma_o}$ simulation, a transition labelled by an event in Σ occurs if an event occurrence in H and G is correctly predicted by the observer even if the event is not observable, and a transition labelled by an event in Σ' occurs if the prediction is not correct. Therefore, if a $\mathcal{M}_{\Sigma_o}$ automaton is constructed and a previously unobservable event σ is made observable, $\mathcal{M}_{\Sigma_o \cup \{\sigma\}}$ could be constructed from $\mathcal{M}_{\Sigma_o}$ by cutting all σ' transitions, and corresponding to the σ event.

Lemma 9 *The automaton $\mathcal{M}_{\Sigma_o}$ can be constructed from $\mathcal{M}_\emptyset$ by iteratively cutting Σ'_o labelled transitions in $\mathcal{M}_\emptyset$.*

Proof: This lemma is shown by a proof by induction on the cardinality of Σ_o.

Base: Suppose $\Sigma_o = \emptyset$. This case is trivial as $\mathcal{M}_{\Sigma_o} = \mathcal{M}_\emptyset$.

Induction hypothesis: For $|\Sigma_o| = n$, the $\mathcal{M}_{\Sigma_o}$ automaton can be constructed from $\mathcal{M}_\emptyset$ by iteratively cutting Σ'_o labelled transitions in $\mathcal{M}_\emptyset$.

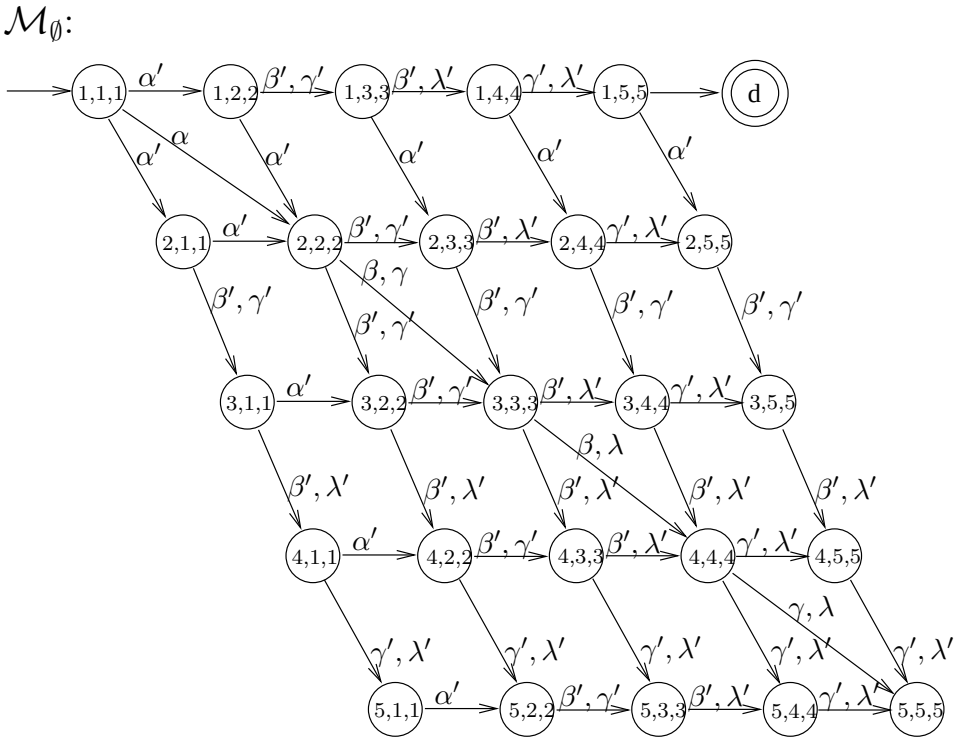

Figure 6.7: The $\mathcal{M}_\emptyset$ machine constructed from G and H of Example 12.

Induction step: Let $|\Sigma_o| = n$. From the induction hypothesis it is known that the $\mathcal{M}_{\Sigma_o}$ automaton can be constructed from $\mathcal{M}_\emptyset$ by iteratively cutting Σ_o' labelled transitions in $\mathcal{M}_\emptyset$.

Let σ be some event in $\Sigma \setminus \Sigma_o$. From the construction of $\mathcal{M}_{\Sigma_o}$ and $\mathcal{M}_{(\Sigma_o \cup \{\sigma\})}$, the only difference in the transition structure of these two automata is that transitions labelled by σ' are absent in $\mathcal{M}_{(\Sigma_o \cup \{\sigma\})}$. Therefore the $\mathcal{M}_{(\Sigma_o \cup \{\sigma\})}$ automaton can be constructed from $\mathcal{M}_{\Sigma_o}$ by cutting all σ' labelled transitions in $\mathcal{M}_{\Sigma_o}$. Hence, the $\mathcal{M}_{(\Sigma_o \cup \{\sigma\})}$ automaton can be constructed from $\mathcal{M}_\emptyset$ by iteratively cutting $\Sigma_o' \cup \{\sigma'\}$ labelled transitions in $\mathcal{M}_\emptyset$. ■

Lemma 9 shows that the sensor selection problem is really a type of colored cut problem. Suppose the automaton $\mathcal{M}_\emptyset$ is considered to be a colored directed graph

as introduced above such that the transition labels are defined to be edge colors. A colored $x_0^{\mathcal{M}_\emptyset} d$-cut for $\mathcal{M}_\emptyset$ where only Σ' transitions are cut corresponds to a sufficient sensor selection for observability to hold. This prompts the following theorem.

Theorem 21 *$\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c if and only if $\Sigma'_o \subseteq \Sigma'$ is a colored $x_0^{\mathcal{M}_\emptyset} d$-cut for $\mathcal{M}_\emptyset$.*

Proof: This proof is demonstrated in two parts. Suppose that $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c. Therefore d is not reachable in $\mathcal{M}_{\Sigma_o}$. From Lemma 9 the automaton $\mathcal{M}_{\Sigma_o}$ can be constructed from $\mathcal{M}_\emptyset$ by iteratively cutting Σ'_o labelled transitions in $\mathcal{M}_\emptyset$. Hence, Σ'_o is a colored $x_0^{\mathcal{M}_\emptyset} d$-cut in $\mathcal{M}_\emptyset$.

Now suppose that $\mathcal{L}(H)$ is not observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c. Therefore d is reachable in $\mathcal{M}_{\Sigma_o}$. From Lemma 9 the automaton $\mathcal{M}_{\Sigma_o}$ can be constructed from $\mathcal{M}_\emptyset$ by iteratively cutting Σ'_o labelled transitions in $\mathcal{M}_\emptyset$. Hence, Σ'_o is a not a colored $x_0^{\mathcal{M}_\emptyset} d$-cut in $\mathcal{M}_\emptyset$. ■

The $\mathcal{M}_\emptyset$ cut problem is not in the same form as in Problem 2 as Σ labelled transitions will never be cut in the $\mathcal{M}_\emptyset$ automaton of Theorem 21 by making events observable. To counter this difference, the following construction is used which performs a form of state condensation and hides the Σ transitions in $\mathcal{M}_\emptyset$.

To start, construct $\mathcal{M}_{\Sigma_o}$ from H, G, Σ_c and Σ_o. Define:

$$X_x^{\mathcal{M}_{\Sigma_o}} = \left\{ y^{\mathcal{M}_{\Sigma_o}} | \exists t \in \Sigma^* \text{ such that } x^{\mathcal{M}_{\Sigma_o}} \overset{t}{\mapsto}_{\mathcal{M}_{\Sigma_o}} y^{\mathcal{M}_{\Sigma_o}} \right\}.$$

Notice the x subscript on $X_x^{\mathcal{M}_{\Sigma_o}}$. The set $X_x^{\mathcal{M}_{\Sigma_o}}$ represents all states that could be reached from $x^{\mathcal{M}_{\Sigma_o}}$ in $\mathcal{M}_{\Sigma_o}$ if only Σ transitions were allowed. These are the same transitions in $\mathcal{M}_{\Sigma_o}$ that could not be cut by making more events observable. Due to this, the states in $X_x^{\mathcal{M}_{\Sigma_o}}$ would be reachable from $x^{\mathcal{M}_{\Sigma_o}}$ according to the transition rules of $\mathcal{M}_{\Sigma_o}$ no matter what events are made observable.

With this in mind, the following nondeterministic automaton $\tilde{\mathcal{M}}_{\Sigma_o}$ is constructed from $\mathcal{M}_{\Sigma_o}$ such that if there is two states $x^{\mathcal{M}_{\Sigma_o}}, y^{\mathcal{M}_{\Sigma_o}}$ and some string of transitions labelled by $s\sigma' \in \Sigma^*\Sigma'$ such that according to the transition rules of $\mathcal{M}_{\Sigma_o}$, $x^{\mathcal{M}_{\Sigma_o}} \overset{s\sigma'}{\mapsto}_{\mathcal{M}_{\Sigma_o}} y^{\mathcal{M}_{\Sigma_o}}$, then according to the transition rules of $\tilde{\mathcal{M}}_{\Sigma_o}$, $x^{\mathcal{M}_{\Sigma_o}} \overset{\sigma'}{\mapsto}_{\tilde{\mathcal{M}}_{\Sigma_o}} y^{\mathcal{M}_{\Sigma_o}}$. This construction effectively condenses all $\mathcal{M}_{\Sigma_o}$ states reachable by Σ transitions. However, it is assumed that $d \notin X_{x_0}^{\mathcal{M}_{\Sigma_o}}$. Let $\tilde{\mathcal{M}}_{\Sigma_o} = (X^{\tilde{\mathcal{M}}_{\Sigma_o}}, x_0^{\tilde{\mathcal{M}}_{\Sigma_o}}, \Sigma^{\tilde{\mathcal{M}}_{\Sigma_o}}, \delta^{\tilde{\mathcal{M}}_{\Sigma_o}})$, where $X^{\tilde{\mathcal{M}}_{\Sigma_o}} := X^H \times X^H \times X^G \cup \{d\}, x_0^{\tilde{\mathcal{M}}_{\Sigma_o}} := (x_0^H, x_0^H, x_0^G)$ and $\Sigma^{\tilde{\mathcal{M}}_{\Sigma_o}} := \Sigma$.

The transition relation $\delta^{\tilde{\mathcal{M}}_{\Sigma_o}}$ is defined as follows. Suppose there exists three states $x^{\mathcal{M}_{\Sigma_o}}, y^{\mathcal{M}_{\Sigma_o}}, z^{\mathcal{M}_{\Sigma_o}} \in X^{\mathcal{M}_{\Sigma_o}}$ and $\sigma \in \Sigma$ such that if $z^{\mathcal{M}_{\Sigma_o}} \in X_x^{\mathcal{M}_{\Sigma_o}}$ and $z^{\mathcal{M}_{\Sigma_o}} \overset{\sigma'}{\mapsto}_{\mathcal{M}_{\Sigma_o}} y^{\mathcal{M}_{\Sigma_o}}$,

$$\delta^{\tilde{\mathcal{M}}_{\Sigma_o}}(x^{\mathcal{M}_{\Sigma_o}}, \sigma) = \left\{ \begin{array}{ll} y^{\mathcal{M}_{\Sigma_o}} & \text{if } d \notin X_y^{\mathcal{M}_{\Sigma_o}} \\ d & \text{if } d \in X_y^{\mathcal{M}_{\Sigma_o}} \end{array} \right\}.$$

An example is now given of an $\tilde{\mathcal{M}}_{\Sigma_o}$ automaton construction.

Example 16 *Recall the system and specification shown in Example 12 and the resulting $\mathcal{M}_{\Sigma_o}$ automaton seen in Figure 6.7. The corresponding $\tilde{\mathcal{M}}_{\emptyset}$ automaton constructed for this system and specification with $\Sigma_c = \{alpha\}$ can be seen in Figure 6.8.*

The $\tilde{\mathcal{M}}_{\Sigma_o}$ automaton is really a colored directed graph where states are vertices, transitions are directed edges and the transition labels are the colors. This prompts one of the main results of this chapter.

Theorem 22 *Given an $\tilde{\mathcal{M}}_{\emptyset}$ automaton constructed from H, G, Σ_c and $\emptyset$ as the set of observable events, $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c if and only if Σ_o is a colored $x_0^{\tilde{\mathcal{M}}_{\emptyset}}d$-cut in the colored directed graph $\tilde{\mathcal{M}}_{\emptyset}$.*

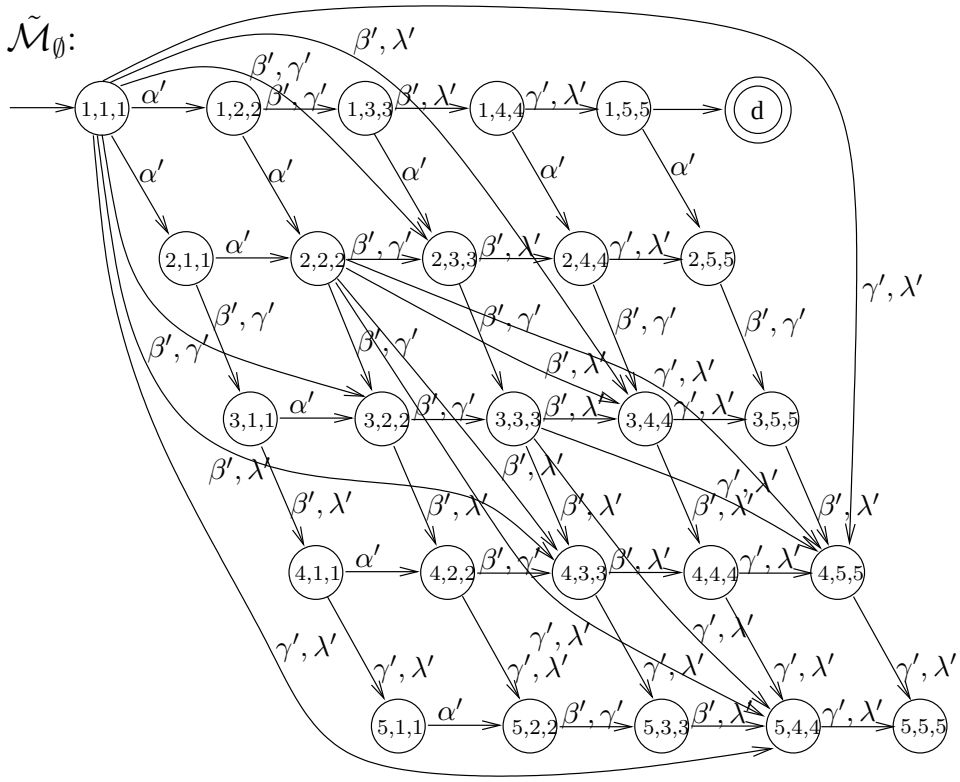

Figure 6.8: The $\tilde{\mathcal{M}}_\emptyset$ machine constructed from G and H of Example 12.

Proof: It has already been shown that $\mathcal{L}(H)$ is observable with respect to $\mathcal{L}(G)$, Σ_o and Σ_c if and only if Σ_o' is a colored $x_0^{\mathcal{M}_\emptyset}d$-cut in the colored directed graph $\mathcal{M}_\emptyset$. Therefore it is sufficient to show that Σ_o is a colored $x_0^{\tilde{\mathcal{M}}_\emptyset}d$-cut in the colored directed graph $\tilde{\mathcal{M}}_\emptyset$ if and only if Σ_o' is a colored $x_0^{\mathcal{M}_\emptyset}d$-cut in the colored directed graph $\mathcal{M}_\emptyset$.

Define a natural projection operation $P' : \Sigma \cup \Sigma' \rightarrow \Sigma'$. Also define the translation operator $\tilde{\Psi} : \Sigma' \rightarrow \Sigma$ such that $\tilde{\Psi}(\sigma') = \sigma$. Both of these functions are extended in the usual manner to be defined over strings. Also define the function $\tilde{P} : \Sigma \cup \Sigma' \rightarrow \Sigma$ that is the composition of $P'(\cdot)$ and $\tilde{\Psi}(\cdot)$, i.e., $\tilde{P}(\sigma) = \tilde{\Psi}(P'(\sigma))$. These functions also have the normally defined inverse operations.

First, suppose that Σ_o' is not a colored $x_0^{\mathcal{M}_\emptyset}d$-cut in the colored directed graph

$\mathcal{M}_\emptyset$. Then there exists a string of transitions labelled by $s \in (\Sigma \cup \Sigma')^*$ such that $x_0^{\mathcal{M}_\emptyset} \overset{s}{\mapsto}_{\mathcal{M}_\emptyset} d$. Due to the construction of $\tilde{\mathcal{M}}_\emptyset$, $x_0^{\tilde{\mathcal{M}}_\emptyset} \overset{\tilde{P}(s)}{\mapsto}_{\tilde{\mathcal{M}}_\emptyset} d$.

Now suppose that Σ'_o is not a colored $x_0^{\tilde{\mathcal{M}}_\emptyset} d$-cut in the colored directed graph $\tilde{\mathcal{M}}_\emptyset$. Then there exists a string of transitions labelled by $s \in \Sigma^*$ such that $x_0^{\tilde{\mathcal{M}}_\emptyset} \overset{s}{\mapsto}_{\tilde{\mathcal{M}}_\emptyset} d$. Due to the construction of $\tilde{\mathcal{M}}_\emptyset$, there exists some string $t \in \tilde{P}^{-1}(s)$ such that $x_0^{\mathcal{M}_\emptyset} \overset{t}{\mapsto}_{\mathcal{M}_\emptyset} d$.

■

With the shown conversions between the graph cutting problem and the sensor selection problem any methods developed to calculate approximate solutions to one problem can be used to calculate approximate solutions to the other problem.

Due the construction of the $\mathcal{M}_{\Sigma_o}$ automaton, it should be apparent that $|X^{\mathcal{M}}| \leq |X^G| * |X^H|^2 + 1$. Furthermore, at each state $x_{\Sigma_o}^{\mathcal{M}} \in X_{\Sigma_o}^{\mathcal{M}}$, the number of state transitions is at most three times the maximum number of output state transitions in any state of G or H. If E^G is the set of state transitions in G and E^H is the number of state transitions in H, let $e = \max\{|E^G|, |E^H|\}$. Therefore $\mathcal{M}_{\Sigma_o}$ can be constructed in time and space in $O(e * |X^G| * |X^H|^2)$ using standard breadth-first digraph construction algorithms. Therefore, because reachability can be tested in polynomial time, the observability of $\mathcal{L}(H)$ with respect to $\mathcal{L}(G)$, Σ_o and Σ_c can be tested in polynomial time [74].

6.6 A Deterministic Greedy Graph Cutting Method

An algorithm is now shown for approximating the solution to the minimal sensor selection problem. This algorithm is based on the $\tilde{\mathcal{M}}_\emptyset$ construction seen above for a system G, a specification H and a set of controllable events Σ_c. After constructing $\tilde{\mathcal{M}}_\emptyset$, events are made observable in order to cut all paths from $x_o^{\tilde{\mathcal{M}}_\emptyset}$ to d in $\tilde{\mathcal{M}}_\emptyset$.

A utility function is now used to decide which events to make observable deter-

mining the relative desirability of cutting a set of transitions associated with an event in $\tilde{\mathcal{M}}_\emptyset$. Starting with a trim version of $\tilde{\mathcal{M}}_\emptyset$, suppose in this automaton it is desirable to find the "probability" $\mathcal{P}(\sigma, \tilde{\mathcal{M}}_\emptyset)$ that a "randomly" selected path from $x_o^{\tilde{\mathcal{M}}_\emptyset}$ to d contains an edge labelled by σ. The term "probability" and "randomly" are used in a loose and intuitive manner in order to develop an understanding for the solution method for this problem while avoiding the explicit definition of a probability distribution function at this time. Naturally it would be desirable to cut transitions associated with events that have the highest probability of occurrence as specified by $\mathcal{P}(\sigma, \tilde{\mathcal{M}}_\emptyset)$. If an event has a high probability of occurring on a run of the $\tilde{\mathcal{M}}_\emptyset$ automaton leading to the d state, then it is desirable to observe occurrences of that event as observing that event would cut the "most" paths to the d state in $\tilde{\mathcal{M}}_\emptyset$ that should be avoided. This prompts the following greedy approximation algorithm.

Algorithm 9 Deterministic Greedy Approximation Algorithm (DetGrAprx)

Input: $G = (X^G, \Sigma, \delta^G, x_0^G)$, $H = (X^H, \Sigma, \delta^H, x_0^H)$, $\Sigma_c \subseteq \Sigma$;

$\Sigma_o \leftarrow \emptyset$;

$i \leftarrow 1$;

Construct $\tilde{\mathcal{M}}_{\Sigma_o}$;

$\tilde{\mathcal{M}}_{\Sigma_o}^T \leftarrow Trim(\tilde{\mathcal{M}}_{\Sigma_o})$;

While $\mathcal{L}_m(\tilde{\mathcal{M}}_{\Sigma_o}^T) \neq \emptyset$;

{

$\sigma_i \leftarrow \arg\max_{\sigma \in \Sigma \setminus \Sigma_o} \left(\mathcal{P}\left(\sigma, \tilde{\mathcal{M}}_{\Sigma_o}^T\right)\right)$;

$\rho_i \leftarrow \mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o}^T\right)$;

$\Sigma_o \leftarrow \Sigma_o \cup \{\sigma_i\}$;

$k \leftarrow i$;

$i \leftarrow i + 1$;

Construct $\tilde{\mathcal{M}}_{\Sigma_o}$*;*

$\tilde{\mathcal{M}}^T_{\Sigma_o} \leftarrow Trim(\tilde{\mathcal{M}}_{\Sigma_o})$*;*

}

Return Σ_o*;*

The relative probabilities $\{\rho_1, \ldots, \rho_k\}$ associated with the events $\{\sigma_1, \ldots, \sigma_k\}$ as the events are selected are stored for later analysis of the accuracy of the found approximation $|\Sigma_o|$. It now needs to be shown how $\mathcal{P}\left(\sigma, \tilde{\mathcal{M}}^T_{\Sigma_o}\right)$ is calculated. This is done by converting $\tilde{\mathcal{M}}^T_{\Sigma_o}$ into a stochastic automaton. At each state $x \in X^{\tilde{\mathcal{M}}^T_{\Sigma_o}}$, suppose there are κ_x output transitions. Note that there may be multiple transitions with the same label. Assign the probability $\frac{1}{\kappa_x}$ to each output transition of x. That probability assignment models that all the output transitions of a state have the same probability of being followed. Therefore, using standard methods from stochastic systems theory [25], $\mathcal{P}(\sigma, \tilde{\mathcal{M}}^T_{\Sigma_o})$ denotes the probability that a random walk in the stochastic version of $\tilde{\mathcal{M}}^T_{\Sigma_o}$ with uniform probability assignments starting at $x_o^{\tilde{\mathcal{M}}^T_{\Sigma_o}}$, traverses a σ transition on its way to d. It should be noted that this probability can be computed in polynomial time using standard methods [25].

Algorithm 9 iteratively chooses events with the highest probability of occurrence in $\tilde{\mathcal{M}}^T_{\Sigma_o}$ over the sets of all $x_o^{\tilde{\mathcal{M}}^T_{\Sigma_o}} d$ paths, adds that event to Σ_o and removes all transitions associated with that event. The $\tilde{\mathcal{M}}^T_{\Sigma_o}$ automaton is trimmed as events are made observable until there is no path from the initial state to the marked state d. Therefore, as Σ_o is updated, the next $\tilde{\mathcal{M}}^T_{\Sigma_o}$ can be calculated in polynomial time. Algorithm 9 runs in polynomial time with respect to the size of the encodings of G and H and the algorithm iterates at most $k \leq |\Sigma|$ times.

The deterministic greedy algorithm is now analyzed to obtain a bound on the ratio of the cardinality of the sensor selection Σ_o returned by Algorithm 9 to the

cardinality of the minimal observability set. This analysis relies on the $\{\rho_1, \ldots, \rho_k\}$ probabilities saved during the operation of Algorithm 9. The set $\Sigma_o^{\min_i}$ denotes the minimum cardinality observability set that could be chosen at iteration i given that events in Σ_o^i are already selected to be observed. Naturally, $\Sigma_o^{\min_1} = \Sigma_o^{\min}$.

Lemma 10 *In Algorithm 9, on the ith iteration,*

$$\frac{1}{\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \leq |\Sigma_o^{\min_i}|$$

Proof: Let $\Sigma_o^{\min_i} = \{\gamma_i^1, \ldots \gamma_i^{k_i}\}$. Also, for $\gamma_i^j, \tilde{\mathcal{M}}_{\Sigma_o^i}^T,\ j \in \{1, \ldots, k_i\}$, let values of $\alpha_i^j \in \left(0, \mathcal{P}\left(\gamma_i^j, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)\right]$ be chosen such that $\sum_{j=1}^{k_i} \alpha_i^j = 1$. Because Algorithm 9 is a greedy algorithm, σ_i has the highest probability of any event not previously chosen for observation on round i, i.e.,

$$\begin{aligned}
& \forall j \in \{1, \ldots, k_i\} \left(\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right) \geq \mathcal{P}\left(\gamma_i^j, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)\right) \\
\Rightarrow\ & \forall j \in \{1, \ldots, k_i\} \left(\frac{1}{\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \leq \frac{1}{\mathcal{P}\left(\gamma_i^j, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)}\right) \\
\Rightarrow\ & \forall j \in \{1, \ldots, k_i\} \left(\frac{1}{\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \leq \frac{1}{\alpha_i^j}\right) \\
\Rightarrow\ & \forall j \in \{1, \ldots, k_i\} \left(\frac{\alpha_i^j}{\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \leq 1\right) \\
\Rightarrow\ & \sum_{j=1}^{k_i} \frac{\alpha_i^j}{\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \leq \sum_{j=1}^{k_i} 1 \\
\Rightarrow\ & \frac{1}{\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \sum_{j=1}^{k_i} \alpha_i^j \leq |\Sigma_o^{\min_i}| \\
\Rightarrow\ & \frac{1}{\mathcal{P}\left(\sigma_i, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \leq |\Sigma_o^{\min_i}|.
\end{aligned}$$

■

Lemma 10 can now be used to show the following result on the closeness of the cost of the approximation to the minimum cost observability set.

Theorem 23 *For the observability set Σ_o returned by Algorithm 9 and the minimum sensor selection $\Sigma_o^{\min}$,*

$$\frac{|\Sigma_o|}{|\Sigma_o^{\min}|} \leq \sum_{i=1}^{|\Sigma_o|} \rho_i$$

where $\{\rho_1, \ldots \rho_k\}$ are the iterative probabilities stored during the operation of Algorithm 9.

Proof: It has already been shown in Lemma 10 that:

$$\begin{aligned}
\frac{1}{\mathcal{P}\left(\sigma, \tilde{\mathcal{M}}_{\Sigma_o^i}\right)} \leq |\Sigma_o^{\min_i}| \quad &\Rightarrow \quad 1 \leq |\Sigma_o^{\min_i}| \mathcal{P}\left(\sigma, \tilde{\mathcal{M}}_{\Sigma_o^i}\right) \\
&\Rightarrow \quad 1 \leq |\Sigma_o^{\min_i}| \rho_i \\
&\Rightarrow \quad |\Sigma_o| \leq \sum_{i=1}^{|\Sigma_o|} |\Sigma_o^{\min_i}| \rho_i \\
&\Rightarrow \quad \frac{|\Sigma_o|}{|\Sigma_o^{\min}|} \leq \sum_{i=1}^{|\Sigma_o|} \rho_i
\end{aligned}$$

■

Because of Theorem 23, a bound on the closeness of the approximation returned by Algorithm 9 can be calculated. Unfortunately $\sum_{i=1}^{k} \rho_i$ can be on the order of $n - \epsilon$ in the worst case where n is the number of system events and ϵ is some constant greater than 0. A lower bound on the closeness of the bound on the approximation ratio shown in Theorem 23 is now shown.

Theorem 24 *From a set $\{\rho_1, \ldots, \rho_k\}$ calculated from a running of Algorithm 9,*

$$\sum_{i=1}^{k} \rho_i \geq H_k$$

where H_k is the sum of the harmonic series $k^{-1}, (k-1)^{-1}, \ldots, 1$.

Proof: Suppose the events chosen to be observable by Algorithm 9, the events in Σ_o are chosen in the order $\sigma_1, \ldots, \sigma_k$. In the automaton $\tilde{\mathcal{M}}_{\Sigma_o^i}$, the transition labelled by σ_i has probability ρ_i of occurring on a random path from $x_o^{\tilde{\mathcal{M}}_{\Sigma_o^i}}$ to d. Before the event σ_i is chosen, there are $k-i+1$ events left to be chosen in the set $\{\sigma_i, \ldots, \sigma_k\}$. All unique paths from $x_o^{\tilde{\mathcal{M}}_{\Sigma_o^i}}$ to d in $\tilde{\mathcal{M}}_{\Sigma_o^i}$ must contain at least one state transition with a label in $\{\sigma_i, \ldots, \sigma_k\}$.

Therefore,

$$\sum_{j=i}^{k} \mathcal{P}(\sigma_j, \tilde{\mathcal{M}}_{\Sigma_o^i}) \geq 1.$$

Because σ_i has the highest probability of occurring on a path from $x_o^{\tilde{\mathcal{M}}_{\Sigma_o^i}}$ to d in $\tilde{\mathcal{M}}_{\Sigma_o^i}$,

$$\begin{aligned} \rho_i \geq \frac{\sum_{j=i}^{k} \mathcal{P}(\sigma_j, \tilde{\mathcal{M}}_{\Sigma_o^i})}{k-i+1} &\Rightarrow \rho_i \geq \frac{1}{k-i+1} \\ &\Rightarrow \sum_{i=k}^{1} \rho_i \geq \sum_{i=k}^{1} \frac{1}{k-i+1} \\ &\Rightarrow \sum_{i=1}^{k} \rho_i \geq H_k. \end{aligned}$$

■

Although Theorem 24 puts a lower bound on the guarantee of the approximation ratio show in Theorem 23, it is not implied that Algorithm 9 cannot have an approximation ratio better than H_k.

6.6.1 A Randomized Greedy Algorithm

A randomized greedy minimal sensor selection algorithm is now given that combines elements of Algorithms 7 and 9. As with Algorithm 7, this new algorithm randomly enables events to be made observable, but uses the utility function $\mathcal{P}(\sigma, \tilde{\mathcal{M}}_{\Sigma_o})$

to weight the probability distribution of a sensor being selected. Therefore, an event with a relatively high probability of occurring over the set of all paths to d in $\tilde{\mathcal{M}}_{\Sigma_o}$ will have a higher probability of being added to the observable events when sensors are being selected.

Algorithm 10 Randomized Weighted Observability Set Search Algorithm (RanWObs):

Input: $G = (X^G, \Sigma, \delta^G, x_0^G)$, $H = (X^H, \Sigma, \delta^H, x_0^H)$, $\Sigma_c \subseteq \Sigma$

$\Sigma_o \leftarrow \emptyset$;

Construct $\tilde{\mathcal{M}}_{\Sigma_o}$ *from* G, H, Σ_c *and* Σ_o;

$i \leftarrow 1$;

$\tilde{\mathcal{M}}^T_{\Sigma_o} \leftarrow Trim(\tilde{\mathcal{M}}^T_{\Sigma_o})$;

While $\mathcal{L}_m(\tilde{\mathcal{M}}^T_{\Sigma_o}) \neq \emptyset$;

{

For all $\sigma \in \Sigma \setminus \Sigma_o$

{

$\Pr(\sigma) \leftarrow \frac{\mathcal{P}\left(\sigma, \tilde{\mathcal{M}}^T_{\Sigma_o}\right)}{\sum_{\gamma \in \Sigma \setminus \Sigma_o} \left(\mathcal{P}\left(\gamma, \tilde{\mathcal{M}}^T_{\Sigma_o}\right)\right)}$;

}

Randomly select $\sigma_i \in \Sigma \setminus \Sigma_o$ *according to probability distribution* $\Pr(\sigma)$;

$k \leftarrow i$;

Remove σ_i *labelled transitions in* $\tilde{\mathcal{M}}^T_{\Sigma_o}$;

$\Sigma_o \leftarrow \Sigma_o \cup \{\sigma_i\}$;

$i \leftarrow i + 1$;

$\tilde{\mathcal{M}}^T_{\Sigma_o} \leftarrow Trim(\tilde{\mathcal{M}}^T_{\Sigma_o})$;

}

Return Σ_o;

Algorithm 10 can be iterated multiple times using a method similar to that in Algorithm 8 to obtain multiple approximations to the minimum cost observability set. Unlike the deterministic approximation algorithm, Algorithm 9, Algorithm 10 may not always return the same approximation.

6.7 Integer and Linear Programming Methods

Another approach to approximating the minimal sensor selection is to use integer programming based methods. This section discusses how to convert the minimal cost sensor selection problem to an integer programming problem. First the integer programming problem is introduced.

Problem 3 The Integer Programming Problem: *Given a z element row vector C, a $y \times z$ matrix A and a y element column vector B, find a z element column vector $\vec{x} \in \{0,1\}^z$ that minimizes $C\vec{x}$ subject to $A\vec{x} \geq B$.*

The integer programming problem is known to be NP-complete, but there is a vast literature on efficiently calculating approximate solutions to this problem as outlined in [49, 77]. Once the sensor selection problem is in the form of an integer programming problem, these already developed methods can be used to find solutions to the sensor selection problem.

6.7.1 Problem Conversion

It is now shown how to convert the sensor selection problem to an integer programming problem. Suppose a system automaton G, a specification automaton H and a set of controllable events Σ_c are given. From this the automaton $\tilde{\mathcal{M}}_\emptyset$ and $\tilde{\mathcal{M}}_{\Sigma_o}$ can be constructed for some $\Sigma_o \subseteq \Sigma$. Note that for the sets of reachable states, $X^{\tilde{\mathcal{M}}_{\Sigma_o}} \subseteq X^{\tilde{\mathcal{M}}_\emptyset}$. That is, some reachable states in $\tilde{\mathcal{M}}_\emptyset$ may not be reachable in $\tilde{\mathcal{M}}_{\Sigma_o}$.

To convert a sensor selection problem to an integer programming problem, the events in Σ and states in $X^{\tilde{\mathcal{M}}_\emptyset}$ are be treated as binary variables. The events in the set $\Sigma_o \subseteq \Sigma$ are all assigned 1, 0 is assigned to all events in $\Sigma \setminus \Sigma_o$, and for all states $x \in X^{\tilde{\mathcal{M}}_\emptyset}$ such that x is reachable from $x_o^{\tilde{\mathcal{M}}_\emptyset}$ according to the transition rules of $\tilde{\mathcal{M}}_{\Sigma_o}$, then $x = 1$. Note that $x_o^{\tilde{\mathcal{M}}_\emptyset} = x_o^{\tilde{\mathcal{M}}_{\Sigma_o}}$.

To express the validity of variable assignments as a set of inequalities, suppose that in $\tilde{\mathcal{M}}_\emptyset$ there is an event $\sigma_{i_1} \in \Sigma$ and two states $x_{i_2}, x_{i_3} \in X^{\tilde{\mathcal{M}}_\emptyset}$ such that $x_{i_2} \overset{\sigma_{i_1}}{\mapsto}_{\tilde{\mathcal{M}}_\emptyset} x_{i_3}$. Therefore, using the variable assignments described above, this transition can be written as an inequality $x_{i_3} \geq x_{i_2} - \sigma_{i_1}$. This represents the property for the automaton $\tilde{\mathcal{M}}_{\Sigma_o}$, if x_{i_2} is reachable in $\tilde{\mathcal{M}}_{\Sigma_o}$ and there is a transition caused by an unobserved event σ_{i_1} that leads to x_{i_3}, then x_{i_3} should also be reachable. This inequality can be manipulated so that $x_{i_3} - x_{i_2} + \sigma_{i_1} \geq 0$.

Therefore, if all of the state transitions in $\tilde{\mathcal{M}}_\emptyset$ are expressed as integer inequalities and the initial state $x_o^{\tilde{\mathcal{M}}_\emptyset}$ is constrained to be reachable (that is, assigned to be 1), then the problem is to find the minimal cardinality set of events assigned to be observable (that is, assigned to be 1) such that the d state does not have to be assigned 1. This set of conditions can now be converted into an integer programming problem. Let a vector $\vec{x}$ be defined such that if $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ and $X^{\tilde{\mathcal{M}}_\emptyset} =$

$\{x_0, x_1, \ldots, x_{n-1}, d\}$:

$$\vec{x} = \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_k \\ x_0 \\ x_1 \\ \vdots \\ x_{n-1} \\ d \end{bmatrix}.$$

Therefore, $z = k + n$.

The row vector C can be a z element constant vector such that for all $i \in \{1, \ldots, k\}$, the ith entry of C is 1 and all other entries of C are 0.

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

Therefore, minimizing $C\vec{x}$ is equivalent to minimizing the cardinality of a sensor selection.

For the constraint matrices, suppose there are e transitions in $\tilde{\mathcal{M}}_\emptyset$. Let $y = e + 2$. Therefore, let B be a column vector of $(e + 1)$ 0's and a 1 at the row corresponding

to $x_0^{\tilde{\mathcal{M}}_\emptyset}$.

$$B = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

With the constraint matrix A below, the element assigned 1 in the B vector ensures that the initial state is forced to be reachable and the last 0 element ensures that the dump state is constrained to not be reachable.

The constraint matrix A is a $y \times z$ matrix and encodes the state reachability conditions depending on the observability of events. Let the transitions be ordered from 1 to e, such that transition l is from the l_2th state to the l_3th state on the occurrence of the l_1th event. Then, entry (l, l_1) is assigned to be 1, entry $(l, (k + l_3))$ is assigned to be 1 and entry $(l, (k + l_2))$ is assigned to be -1. All entries of A not corresponding to a transition are assigned 0. This ensures the constraint equation $A\vec{x} \geq B$ satisfies that for all transitions, $x_{l_3} - x_{l_2} + \sigma_{l_1} \geq 0$. The bottom two rows of A is all zeros, except for the (y, z) element which is assigned to be -1, and $(y-1, l_{x_0^{\tilde{\mathcal{M}}_\emptyset}})$ is assigned to be 1, if $l_{x_0^{\tilde{\mathcal{M}}_\emptyset}}$ represents the row of $\vec{x}$ corresponding to the $x_0^{\tilde{\mathcal{M}}_\emptyset}$ state. This constrains the d state to not be reachable and the initial state to be reachable. That is, the $x_0^{\tilde{\mathcal{M}}_\emptyset} \geq 1$ and $-d \geq 0$, so that $d \leq 0$.

Using this formulation of A, B, C and $\vec{x}$, the sensor selection problem is now in the form of an integer programming problem such that $C\vec{x}$ is minimized as long as $A\vec{x} \geq B$.

As a simple example of how to convert a graph cutting problem into an integer programming problem, consider the $\tilde{\mathcal{M}}_\emptyset$ automaton seen in Figure 6.9.

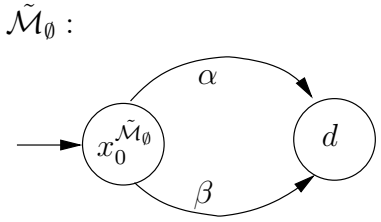

Figure 6.9: An example of a $\tilde{\mathcal{M}}_\emptyset$ automaton.

It is assumed that the system events have uniform cost of being observed. Therefore,

$$\vec{x} = \begin{bmatrix} \alpha \\ \beta \\ x_o^{\tilde{\mathcal{M}}_\emptyset} \\ d \end{bmatrix}$$

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

$$B = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

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

6.8 Applications to Minimal Communication Decentralized Control

An approximation problem for communicating decentralized control systems is now investigated where several controllers make local observations of a system and enforce local control actions that are combined globally using an intersection operation. That is, if an event is disabled by any one controller, then it is disabled by all controllers. It is assumed that the controllers communicate such that the occurrence of a subset of the system events observed by the controllers are communicated to the other controllers, and the problem is to find the minimal cardinality set of events that need to be communicated in order to solve a control task. This is a special case of the communicating controller open problem discussed in [75]. The computational methods developed for the centralized control problem discussed in the sections above can be intuitively applied to the communicating controller problem discussed here. This sections investigates two controller systems, but the results presented can be generalized to n controller systems.

For this problem a system G and a specification H are given with sets of events Σ_{c1}, Σ_{c2} locally controlled by the controllers and events Σ_{o1}, Σ_{o2} locally observed by the controllers. Suppose that $\mathcal{L}(H)$ is not co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2} and Σ_{c1}, Σ_{c2}. Therefore there does not exist a pair of controllers S_1, S_2 such that $\mathcal{L}(S_1 \wedge S_2/G) = \mathcal{L}(H)$.

However, it may be possible that controller 1 might be able to communicate its observations of a subset of the locally observable events $\Sigma_{o12} \subseteq \Sigma_{o1}$ to controller 2 and controller 2 might be able to communicate its observations of a subset of the locally observable events $\Sigma_{o21} \subseteq \Sigma_{o2}$ to controller 1 such that $\mathcal{L}(H)$ would then be co-observable with respect to $\mathcal{L}(G)$, $(\Sigma_{o1} \cup \Sigma_{o21}), (\Sigma_{o2} \cup \Sigma_{o12})$ and Σ_{c1}, Σ_{c2}. Then, if the

occurrence of *all* Σ_{o21} events are communicated to controller 1 and the occurrence of *all* Σ_{o12} events are communicated to controller 2, there would exists communicating controllers S_1, S_2 such that $\mathcal{L}(S_1 \wedge S_2/G) = \mathcal{L}(H)$.

Using this intuition, the sets of communicated events $\Sigma_{o12}, \Sigma_{o21}$ are called sufficient if $\mathcal{L}(H)$ is co-observable with respect to $\mathcal{L}(G)$, $(\Sigma_{o1} \cup \Sigma_{o21}), (\Sigma_{o2} \cup \Sigma_{o12})$ and Σ_{c1}, Σ_{c2}. As above in this chapter, it is generally assumed that $\mathcal{L}(H)$ is controllable. This prompts the following communication minimization problem definition.

Problem 4 Minimal Cardinality Communication Selection: *Given a system G, a specification H and controllable events $\Sigma_{c1}, \Sigma_{c2} \subseteq \Sigma$, find a sufficient communication selection $\Sigma_{o12}^{min}, \Sigma_{o21}^{min}$ such that for any other sufficient communication selection $\Sigma_{o12}, \Sigma_{o21}$,*

$$|\Sigma_{o12}^{min}| + |\Sigma_{o21}^{min}| \leq |\Sigma_{o12}| + |\Sigma_{o21}|.$$

It is assumed that $\mathcal{L}(H)$ is always co-observable with respect to the system $\mathcal{L}(G)$, the observed events with full communication $(\Sigma_{o1} \cup \Sigma_{o2}), (\Sigma_{o1} \cup \Sigma_{o2})$ and the controllable events Σ_{c1}, Σ_{c2}. If this does not hold, then there are no solutions to the communicating controller problem. It is now shown how to convert Problem 4 into a type of graph cutting problem.

6.8.1 Graph Cutting for Communication Selection

Problem 4 is a modification of Problem 1 and there exists a similar $\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}$ construction for converting the communication selection problem to a type of graph cutting problem. Consider the following construction for the two controller case that can be extended for the n controller case.

The $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ automaton is effectively a nondeterministic simulation of a pair of observers predicting unobservable system behavior with respect to a specification

and the communicated events $\Sigma_{o12}, \Sigma_{o21}$. These estimates are based on imperfect predictions of occurrences of locally unobservable and uncommunicated events ($\Sigma \setminus (\Sigma_{o1} \cup \Sigma_{o21})$ for observer 1) in the system.

Let Σ_1, Σ_2 and Σ' be copies of the event set Σ where for every event $\sigma \in \Sigma$, there is are corresponding events $\sigma_1 \in \Sigma_1$, $\sigma_2 \in \Sigma_2$ and $\sigma' \in \Sigma'$. The following can then be defined.

$$\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}} =$$
$$(X^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}, x_0^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}, (\Sigma \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma'), \delta^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}, X_m^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}})$$

where

$$\begin{aligned} X^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} &:= X^H \times X^H \times X^H \times G^G \cup \{d\}, \\ x_0^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} &:= (x_0^H, x_0^H, x_0^H, x_0^G), \\ X_m^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} &:= \{d\}. \end{aligned}$$

Suppose a string of events s has been simulated to occur in the system G by $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ and the simulation is at state $(x_1, x_2, x_3, x_4) \in X^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$. State x_4 represents the true state of the system G and x_3 represents the corresponding state of the specification H after s has occurred. States x_3 and x_4 always update simultaneously. However, as was stated above, the local observers attempt to predict the occurrence of locally unobserved and uncommunicated system events. States x_1 and x_2 represents the estimates of the possible state of the specification based on imperfect predictions of the simulated system behavior s for observer 1 and 2 respectively.

Furthermore, at state (x_1, x_2, x_3, x_4) of the simulation, if an event σ is correctly predicted by both observers in the simulation, there is a transition from (x_1, x_2, x_3, x_4)

labelled by σ where all of the component states of (x_1, x_2, x_3, x_4) update on the occurrence of σ according to the transition rules of H, H and G respectively. A correct prediction may occur for either observable or unobservable events.

However, if an event σ occurs in the system that is not predicted correctly by either of the observers in the simulation, there is transition from (x_1, x_2, x_3, x_4) labelled by σ' where in the resultant state the x_3, x_4 component states of (x_1, x_2, x_3, x_4) update on the occurrence of σ according to the transition rules of H and G respectively. This can only occur if σ is in $\Sigma \setminus (\Sigma_{o1} \cup \Sigma_{o2})$. Similarly, if an event $\sigma \in \Sigma \setminus (\Sigma_{o1} \cup \Sigma_{o2})$ does not occur in the system but is incorrectly predicted to occur by an observer in the simulation, there is a transition from (x_1, x_2, x_3, x_4) labelled by σ' where the x_1 component state of (x_1, x_2, x_3, x_4) updates on the occurrence of σ according to the transition rules of H. Also, there is a transition from (x_1, x_2, x_3, x_4) labelled by σ' where the x_2 component state of (x_1, x_2, x_3, x_4) updates on the occurrence of σ according to the transition rules of H if observer 2 predicts incorrectly. In the $\mathcal{M}_{\Sigma_o 12}, \Sigma_{o21}$ construction the Σ' transitions would not be removed by having the controller communicate more events.

Similar to the Σ' transitions, the Σ_1 and Σ_2 transitions are those transitions that correspond to events are respectively observed by controller 2 and 1, but not observed by both and would be removed if those events were communicated between the controllers.

Therefore, if an event σ occurs in the system that is not predicted correctly by controller 1, but is observed by controller 2, there is a σ_1 labelled transition from (x_1, x_2, x_3, x_4) where in the resultant state the x_2, x_3, x_4 component states update on the occurrence of σ according to the transition rules of H and G respectively. This can only occur if σ is in $\Sigma_{o2} \setminus (\Sigma_{o1} \cup \Sigma_{o21})$. Similarly, if an event $\sigma \in \Sigma_{o2} \setminus (\Sigma_{o1} \cup \Sigma_{o21})$

does not occur in the system but is incorrectly predicted to occur by observer 1 in the simulation, there is a transition from (x_1, x_2, x_3, x_4) labelled by σ_1 where the x_1 component state of (x_1, x_2, x_3, x_4) updates on the occurrence of σ according to the transition rules of H.

If an event σ occurs in the system that is not predicted correctly by controller 2, but is observed by controller 1, there is transition from (x_1, x_2, x_3, x_4) labelled by σ_2 where in the resultant state the x_1, x_3, x_4 component states of (x_1, x_2, x_3, x_4) update on the occurrence of σ according to the transition rules of H and G respectively. This can only occur if σ is in $\Sigma_{o1} \setminus (\Sigma_{o2} \cup \Sigma_{o12})$. Similarly, if an event $\sigma \in \Sigma_{o1} \setminus (\Sigma_{o2} \cup \Sigma_{o12})$ does not occur in the system but is incorrectly predicted to occur by observer 2 in the simulation, there is a transition from (x_1, x_2, x_3, x_4) labelled by σ_2 where the x_2 component state of (x_1, x_2, x_3, x_4) updates on the occurrence of σ according to the transition rules of H.

If the $\mathcal{M}_{\Sigma_o 12}, \Sigma_{o21}$ simulation ever reaches a composed state where both observers believe the occurrence of a controllable event $\sigma \in \Sigma_c$ is allowed by the specification due to the properties of states x_1 and x_2, but in reality it is not due to x_3, yet still possible due to x_4, then there is a control conflict. This possibility is captured by the $(*)$ condition such that if the simulation could reach a state (x_1, x_2, x_3, x_4) where $(*)$ holds, then illegal controllable behavior could occur in the system without either observer being able to resolve the control conflict.

$$\left.\begin{array}{ll} \delta^H(x_1, \sigma) & \text{is defined if } \sigma \in \Sigma_{c1} \\ \delta^H(x_2, \sigma) & \text{is defined if } \sigma \in \Sigma_{c2} \\ \delta^H(x_3, \sigma) & \text{is not defined} \\ \delta^G(x_4, \sigma) & \text{is defined} \end{array}\right\}. \qquad (*)$$

The transition relation $\delta^{\mathcal{M}_{\Sigma_{o12}, \Sigma_{o21}}}$ is defined as follows.

For $\sigma \in \Sigma$,

$$\delta^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}((x_1,x_2,x_3,x_4),\sigma) = \left\{ \begin{array}{l} (\delta^H(x_1,\sigma),\delta^H(x_2,\sigma),\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ d \quad \text{if } (*) \end{array} \right\}.$$

For $\sigma \in \Sigma \setminus (\Sigma_{o1} \cup \Sigma_{o2})$,

$$\delta^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}((x_1,x_2,x_3,x_4),\sigma') = \left\{ \begin{array}{l} (\delta^H(x_1,\sigma),x_2,x_3,x_4) \\ (x_1,\delta^H(x_2,\sigma),x_3,x_4) \\ (x_1,x_2,\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ d \quad \text{if } (*) \end{array} \right\}.$$

For $\sigma \in \Sigma_{o2} \setminus (\Sigma_{o1} \cup \Sigma_{o21})$ and the corresponding $\sigma_1 \in \Sigma_1$,

$$\delta^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}((x_1,x_2,x_3,x_4),\sigma_1) = \left\{ \begin{array}{l} (\delta^H(x_1,\sigma),x_2,x_3,x_4) \\ (x_1,\delta^H(x_2,\sigma),\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \end{array} \right\}.$$

For $\sigma \in \Sigma_{o1} \setminus (\Sigma_{o2} \cup \Sigma_{o12})$ and the corresponding $\sigma_2 \in \Sigma_2$,

$$\delta^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}((x_1,x_2,x_3,x_4),\sigma_2) = \left\{ \begin{array}{l} (x_1,\delta^H(x_2,\sigma),x_3,x_4) \\ (\delta^H(x_1,\sigma),x_2,\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \end{array} \right\}.$$

For $\sigma \in \Sigma$, $\delta^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}(d,\sigma)$ is undefined.

The construction for $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ is different from the one in [67] in that σ_1 and σ_2 transitions correspond to state estimation updates that could be removed if σ observances would be communicated between the controllers. The $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ construction prompts the following corollary to the main result of [67].

Corollary 9 *The state d is reachable from the initial state in* $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ *if and only if* $\mathcal{L}(H)$ *is co-observable with respect to* $\mathcal{L}(G)$, $(\Sigma_{o1} \cup \Sigma_{o21})$, $(\Sigma_{o2} \cup \Sigma_{o12})$, Σ_{c1} *and* Σ_{c2}.

Note that the $\mathcal{M}_{(\Sigma_{o12}\cup\{\sigma\}),\Sigma_{o21}}$ automaton can be constructed from the $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ automaton by cutting all transitions labelled by σ_1 and the $\mathcal{M}_{\Sigma_{o12},(\Sigma_{o21}\cup\{\sigma\})}$ automaton can be constructed from the $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ automaton by cutting all transitions labelled by σ_2. Therefore, the act of controller i communicating all occurrences of event σ to controller j corresponds to trimming all σ_j labelled transitions in $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$.

For the set of events $\Sigma_{oij} \subseteq \Sigma$, let $\Sigma_j^{oij} \subseteq \Sigma_j$ represent the corresponding set of events such that $\sigma \in \Sigma_{oij}$ if and only if $\sigma_j \in \Sigma_j^{oij}$. A set of events $\Sigma_1^{o21} \cup \Sigma_2^{o12}$ is a $x_0^{\mathcal{M}_{\emptyset,\emptyset}} d$-cut in $\mathcal{M}_{\emptyset,\emptyset}$ if and only if $\mathcal{L}_m(\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}) = \emptyset$ and consequently $\mathcal{L}(H)$ is co-observable with respect to $\mathcal{L}(G)$, $(\Sigma_{o1} \cup \Sigma_{o21})$, $(\Sigma_{o2} \cup \Sigma_{o12})$, Σ_{c1} and Σ_{c2}. Therefore, the pair of sets $(\Sigma_{o12}^{min}, \Sigma_{o21}^{min})$ is the smallest cardinality communication selection if and only if the corresponding events $\Sigma_1^{o21min} \cup \Sigma_2^{o12min} \subseteq \Sigma_1 \cup \Sigma_2$ is the smallest cardinality $x_0^{\mathcal{M}_{\emptyset,\emptyset}} d$-cut in $\mathcal{M}_{\emptyset,\emptyset}$ when restricted to cutting transitions labelled with events in $\Sigma_1 \cup \Sigma_2$.

As with the $\mathcal{M}_{\Sigma_o}$ construction given above this realization converts the communicating controller selection problem into a type of graph cutting problem as long as only events in Σ_1 and Σ_2 are cut. There is a $\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}$ construction that can be used to convert this graph cutting problem into a true edge-colored directed graph st-cut problem.

Define:

$$X_x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} = \left\{ y^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} | \exists t \in (\Sigma \cup \Sigma')^*, x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} \overset{t}{\mapsto}_{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} y^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} \right\}.$$

$X_x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$ represents all states that could be reached from $x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$ in $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$

if only Σ transitions were allowed. The states in $X_x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$ would be reachable from $x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$ in $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ no matter what events are communicated between the controllers because only transitions labelled by events in $\Sigma_1 \cup \Sigma_2$ can be cut in $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$ through the communication of events. With this in mind, the following nondeterministic automaton $\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}$ is constructed from $\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}$. It is assumed that $d \notin X_{x_0}^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$. Let $\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}} = (X^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}}, x_0^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}}, \Sigma^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}}, \delta^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}})$, where $X^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}} := X^H \times X^H \times X^H \times X^G \cup \{d\}, x_0^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}} := (x_0^H, x_0^H, x_0^H, x_0^G)$ and $\Sigma^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}} := \Sigma_1 \cup \Sigma_2$.

The transition relation $\delta^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}}$ is defined as follows. Suppose there exists three states $x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}, y^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}, z^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} \in X^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$ and $\sigma \in \Sigma$ such that $z^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} \in X_x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$, $z^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} \overset{\sigma_i}{\mapsto}_{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} y^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}$ where $\sigma_i \in \Sigma_1 \cup \Sigma_2$. Then,

$$\delta^{\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}}\left(x^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}}, \sigma_i\right) = \left\{ \begin{array}{ll} y^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} & \text{if} \quad d \notin X_y^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} \\ d & \text{if} \quad d \in X_y^{\mathcal{M}_{\Sigma_{o12},\Sigma_{o21}}} \end{array} \right\}$$

The $\tilde{\mathcal{M}}_{\Sigma_{o12},\Sigma_{o21}}$ automaton is really a colored directed graph where states are vertices, transitions are directed edges and the transition labels are the colors. This prompts another of the main contributions of this chapter.

Theorem 25 *Given an $\tilde{\mathcal{M}}_{\emptyset,\emptyset}$ automaton constructed from H, G, Σ_{o1}, Σ_{o2}, Σ_{c1}, Σ_{c2} and $\emptyset, \emptyset$ as the sets of communicated events, $\mathcal{L}(H)$ is co-observable with respect to $\mathcal{L}(G)$, $(\Sigma_{o1} \cup \Sigma_{o21}), (\Sigma_{o2} \cup \Sigma_{o12})$ and Σ_{c1}, Σ_{c2} if and only if $\Sigma_1^{o21} \cup \Sigma_2^{o12}$ is a colored $x_0^{\tilde{\mathcal{M}}_{\emptyset,\emptyset}} d$-cut in the colored directed graph $\tilde{\mathcal{M}}_{\emptyset,\emptyset}$.*

Proof: $\mathcal{L}(H)$ is co-observable with respect to $\mathcal{L}(G)$, $(\Sigma_{o1} \cup \Sigma_{o21}), (\Sigma_{o2} \cup \Sigma_{o12})$ and Σ_{c1}, Σ_{c2} if and only if $\Sigma_1^{o21} \cup \Sigma_2^{o12}$ is a colored $x_0^{\mathcal{M}_{\emptyset,\emptyset}} d$-cut in the colored directed graph $\mathcal{M}_{\emptyset,\emptyset}$. Therefore it is sufficient to show that $\Sigma_1^{o21} \cup \Sigma_2^{o12}$ is a colored $x_0^{\tilde{\mathcal{M}}_{\emptyset,\emptyset}} d$-cut in

the colored directed graph $\tilde{\mathcal{M}}_{\emptyset,\emptyset}$ if and only if $\Sigma_1^{o21} \cup \Sigma_2^{o12}$ is a colored $x_0^{\mathcal{M}_{\emptyset,\emptyset}} d$-cut in the colored directed graph $\mathcal{M}_{\emptyset,\emptyset}$

Define the natural projection $P_{12} : \Sigma \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma' \rightarrow \Sigma_1 \cup \Sigma_2$. Suppose that $\Sigma_1^{o21} \cup \Sigma_2^{o12}$ is not a colored $x_0^{\mathcal{M}_{\emptyset,\emptyset}} d$-cut in the colored directed graph $\mathcal{M}_{\emptyset,\emptyset}$. Then there exists a string of transitions labelled by $s \in (\Sigma \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma')^*$ such that $x_0^{\mathcal{M}_{\emptyset,\emptyset}} \overset{s}{\mapsto}_{\mathcal{M}_{\emptyset,\emptyset}} d$. Due to the construction of $\tilde{\mathcal{M}}_{\emptyset,\emptyset}$, $x_0^{\tilde{\mathcal{M}}_{\emptyset,\emptyset}} \overset{P_{12}(s)}{\mapsto}_{\tilde{\mathcal{M}}_{\emptyset,\emptyset}} d$.

Now suppose that $\Sigma_1^{o21} \cup \Sigma_2^{o12}$ is not a colored $x_0^{\tilde{\mathcal{M}}_{\emptyset,\emptyset}} d$-cut in the colored directed graph $\tilde{\mathcal{M}}_{\emptyset,\emptyset}$. Then, there exists a string of transitions labelled by $s \in (\Sigma_1^{o21} \cup \Sigma_2^{o12})^*$ such that $x_0^{\tilde{\mathcal{M}}_{\emptyset,\emptyset}} \overset{s}{\mapsto}_{\tilde{\mathcal{M}}_{\emptyset,\emptyset}} d$. Due to the construction of $\tilde{\mathcal{M}}_{\emptyset,\emptyset}$, there exists some string of transitions labelled by $t \in \tilde{P}_{12}^{-1}(s)$ such that $x_0^{\mathcal{M}_{\emptyset,\emptyset}} \overset{t}{\mapsto}_{\mathcal{M}_{\emptyset,\emptyset}} d$. ■

With the shown conversion between the graph cutting problem and the sensor selection problem the methods outlined above to approximate minimal solutions to the graph cutting problem can be used to approximate solutions to the communication selection problem. The $\tilde{\mathcal{M}}_{\emptyset,\emptyset}$ automaton shown above can be constructed in polynomial time, but this most likely does not hold if the number of controllers is unbounded. This is due to the result in [66] that the problem of deciding co-observability for systems with an unbounded number of controllers is PSPACE-complete.

6.9 Discussion

This chapter has shown results related to the approximation of minimal sensor selections for centralized supervisory control synthesis. It was shown that minimal sensor selections cannot be approximated within a constant factor in polynomial time unless P=NP. It was shown how to convert the sensor selection problem into an edge colored directed graph *st*-cut problem. Several deterministic and randomized heuristic approximation methods for this directed graph problem were shown and a

conversion of the directed graph problem to an integer programming problem was given. An open communicating controller problem was also discussed and it was shown how to convert this minimal communication decentralized control problem into an edge colored directed graph st-cut problem. Therefore the methods discussed in this chapter for dealing with the edge colored directed graph st-cut problem can be used to solve the minimal communication problem.

CHAPTER VII

SYMMETRIC DISTRIBUTED DISCRETE-EVENT SYSTEMS

7.1 Chapter Overview

This chapter discussed issues related to the verification of distributed discrete-event systems composed of isomorphic modules. A finite state automaton system model is assumed where a set of atomic propositions are defined on the states. A type of symmetry is defined for these modular systems and a restriction of the μ-calculus designed for these modular systems is introduced. A procedure is shown to reduce the time and space complexity of testing if states of these symmetric modular systems satisfy propositions in this μ-calculus. An example of a symmetric modular UAV platoon leader system is then discussed.

7.2 Chapter Assumptions

As many distributed systems problems have already been found to be generally computationally difficult as shown in Chapter IV, this chapter investigates an important special case where the various system modules are exact copies of one another except for the renaming of events. Systems modelled this way could be thought of intuitively in the context of the object orientation paradigm in computer science.

Each system module is an instantiation of a generic system object. The scope of some of the behavior of the modules is private and does not affect the other modules, while each of the modules might have some public behavior with global scope that is relevant to other modules. The system events may or may not be renamed in the various instantiations of the modules in order to reflect if behavior is coordinated among several modules on the occurrence of that event. These models can be used to represent several important systems such as swarms of Unmanned Aerial Vehicles (henceforth called UAV's), computer networks and resource sharing manufacturing systems.

The models used in this chapter are generalized from the standard model introduced in [54] to permit more atomic propositions on the module states than state marking. This model extension is inspired by the work in the formal methods community in computer science. See [14, 28] for an introduction to formal methods.

A method is given for reducing the inherent complexity of testing the specialized μ-calculus propositions for permutation symmetric systems by constructing quotient structures that are equivalent with respect to μ-calculus propositions for symmetric systems. Previously, group-theoretic methods have been used in [17] to define classes of states and transitions in a system that are equivalent under defined permutation operations. Finding classes of equivalent states as in [17] is computationally rather difficult and is at least as difficult as the graph isomorphism problem [26]. This has induced several authors to attempt the design distributed systems with special architectures so that special types of symmetry are guaranteed to occur *a priori*, therefore avoiding intensive symmetry verification procedures [17, 62]. This chapter expands on this approach by discussing classes of distributed systems that are assumed to contain a type of symmetry called "permutation symmetry". The quotient automa-

ton presented here for verifying properties on permutation symmetric systems was also discussed in [62], but for a more restricted class of systems.

7.3 Specialized Modelling and Notational Definitions

This chapter uses a distributed system model called an "isomorphic module system" that is a modification of the standard supervisory control discrete-event system model used above. The distributed systems are composed of sets of interacting automata $\{G_1, \ldots, G_n\}$, and each automaton $G_i = (X, x_0, AP, L, \Sigma_i, \delta_i)$ models the behavior of a single module in the system. The set X is a set of system states and x_0 is the initial state. AP is a set of atomic propositions and $L : X \to 2^{AP}$ maps a state to the set of propositions that hold at that state. Σ_i is the set of events relevant to the behavior of module G_i and $\delta_i : X \times \Sigma_i \to X$ is the state transition function. The isomorphic module systems can now be formally defined.

Definition 16 *Suppose a set of automata* $\{G_1, \ldots, G_n\}$ *are given such that* $\forall i \in \{1, \ldots, n\}, G_i = (X, x_0, AP, L, \Sigma_i, \delta_i)$. *The automata* $\{G_1, \ldots, G_n\}$ *form an* isomorphic module system *if the automata are isomorphic to one another when the state transition labellings are disregarded.*

As stated, the system modules, $\{G_1, \ldots, G_n\}$, are isomorphic to one another, but that isomorphism does not extend to transition labelling. That is, the module G_j is a copy of G_i except that the local transition labels Σ_i are replaced with the respective events from Σ_j according to a predefined translation mapping $\Psi_{ij} : \Sigma_i \to \Sigma_j$. The function $\Psi_{ij}(\cdot)$ translates a string of events relevant to module i to a string of events relevant to module j. To formalize, for $x \in X, \gamma \in \Sigma_i$, the transition function is defined as $\delta_i(x, \gamma) = \delta_j(x, \Psi_{ij}(\gamma))$. The function $\Psi_{ij}(\cdot)$, which is assumed to be one-to-one, is also extended in the usual manner for strings and languages.

The inverse translation function is defined as $\Psi_{ij}^{-1}(\cdot) = \Psi_{ji}(\cdot)$. Note that it is possible that $\sigma \in \Sigma_i \cap \Sigma_j$ but $\Psi_{ij}(\sigma) \neq \sigma$. For an event $\sigma_i \in \Sigma_i$, the notation σ_j is used to represent $\Psi_{ij}(\sigma_i)$ when it can be done without ambiguity.

Note that the set of system states, X, is not indexed nor are the atomic propositions AP or the state labelling function L. Therefore, when two modules are at the same local system state, the same atomic propositions hold at both states. If the system state labels are restricted to a binary state marking (i.e., to $AP = \{m\}$), then this model is equivalent to the commonly used model in supervisory control theory [54].

An extended parallel composition operation, denoted by $\|$, is used to model the interaction between modules. For a set of isomorphic modules $\{G_1, \ldots, G_n\}$, the interaction of these modules is the system

$$\begin{aligned} G^{\|} &= G_1 \| \cdots \| G_n \\ &= (X^{\|}, x_0^{\|}, AP, L^{\|}, \Sigma, \delta^{\|}). \end{aligned}$$

The $X^{\|}$, $x_0^{\|}$ and $\delta^{\|}$ components are defined in the usual manner for the parallel composition operation. Let $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_n$. The states of the composed system $G_1 \| \cdots \| G_n$ (i.e., $x^{\|} \in X^{\|}$) are called *composed states*. The individual states of a module (i.e., $x \in X$) are called *module states*. The composed states are n-tuples of the module states.

For a composed state n-tuple $x^{\|}$, the ith module state is represented as ${x^{\|}}^i$. A transposition operator $\phi_{ij} : X^{\|} \to X^{\|}$ is defined where $\phi_{ij}(x^{\|})$ is $x^{\|}$ with the ith and jth module states swapped. The composed state labelling function $L^{\|} : X^{\|} \to 2^{AP}$ is not predefined except to require that for

$$\phi_{ij}(x_a^{\|}) = x_b^{\|} \Rightarrow L^{\|}(x_a^{\|}) = L^{\|}(x_b^{\|}). \tag{7.1}$$

A fundamental result of group theory [24] is that any permutation operator can be constructed from the composition of transposition operators. Therefore, a state permutation operator is constructed as follows from a set of transposition operators for a given input $x^{\|} \in X^{\|}$:

$$\Phi_{[(i_1 j_1)(i_2 j_2)\cdots(i_m j_m)]}(x^{\|}) = \phi_{i_1 j_1}\left(\phi_{i_2 j_2}\left(\cdots \phi_{i_m j_m}\left(x^{\|}\right)\right)\right). \tag{7.2}$$

Given a state $x^{\|}$, the set of all possible permutation operators $\Phi_{[\ldots]}(\cdot)$ can be used to define the set of composed states that are permutations of the components in the n-tuple $x^{\|}$. These states are called the *permutation equivalent* states of $x^{\|}$. Given a state space $X = \{x_1, \ldots, x_k\}$ for an isomorphic module system $G_1, \ldots, G_n$, an arbitrary ordering can be placed on the states in X such that $x_1 < x_2 < \cdots < x_k$. Therefore, for any set of permutation equivalent states from $X^{\|}$, there is always one state such that the module states of the composed states have the correct relative order with respect to the ordering $x^{\|1} \leq x^{\|2} \leq \cdots \leq x^{\|n}$. This state is called the *standard permutation* of all states in its equivalence class. A function $SP : X^{\|} \rightarrow X^{\|}$ is defined such that when given an n-tuple composed state $z^{\|}$, $SP(z^{\|}) = y^{\|}$ is another composed state with the same module states as $z^{\|}$ in the correct order, i.e. $\{z^{\|1}, z^{\|2}, \cdots, z^{\|n}\} = \{y^{\|1}, y^{\|2}, \cdots, y^{\|n}\}$ and $y^{\|1} \leq y^{\|2} \leq \cdots \leq y^{\|n}$. Hence, $SP(\cdot)$ is called the standard permutation operator.

Let $\vec{X} = \{x^{\|} \in X^{\|} | x^{\|} = SP(x^{\|})\}$ be the set of standard permutations. The inverse function $SP^{-1} : \vec{X} \rightarrow 2^{X^{\|}}$ returns the set of states that has the input as its standard permutation. Note that the initial state $x_0^{\|}$ is its own standard permutation because $x_0^{\|} = (x_0, \ldots, x_0)$. Also note that using the notation just defined, the above mentioned requirement on the composed state labelling function that for $x_a^{\|}, x_b^{\|} \in X^{\|}$, $i, j \in \{1, \ldots, n\}$ if $\phi_{ij}(x_a^{\|}) = x_b^{\|}$, then $L^{\|}(x_a^{\|}) = L^{\|}(x_b^{\|})$[Equation 7.1] can be rephrased

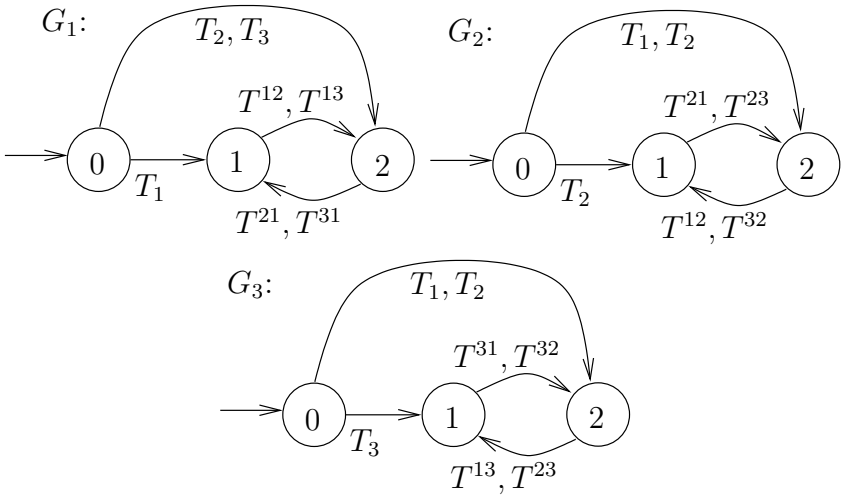

Figure 7.1: Modules G_1, G_2, G_3 for Example 17.

as follows for all $\vec{x} \in \vec{X}$ and $x^{\|} \in X^{\|}$

$$x^{\|} \in SP^{-1}(\vec{x}) \Rightarrow L^{\|}(x^{\|}) = L^{\|}(\vec{x}). \tag{7.3}$$

For all $x^{\|} \in X^{\|}$ there is a (non-unique) string of index pairs $\left[(i_1^{x^{\|}} j_1^{x^{\|}})(i_2^{x^{\|}} j_2^{x^{\|}}) \cdots (i_m^{x^{\|}} j_m^{x^{\|}})\right]$ such that $\Phi_{\left[(i_1^{x^{\|}} j_1^{x^{\|}})(i_2^{x^{\|}} j_2^{x^{\|}}) \cdots (i_m^{x^{\|}} j_m^{x^{\|}})\right]}(x^{\|}) = SP(x^{\|})$. This string of index pairs defines a permutation operator $\Phi_{x^{\|}}(\cdot) : X^{\|} \rightarrow X^{\|}$ such that $\Phi_{x^{\|}}(\cdot) = \Phi_{\left[(i_1^{x^{\|}} j_1^{x^{\|}})(i_2^{x^{\|}} j_2^{x^{\|}}) \cdots (i_m^{x^{\|}} j_m^{x^{\|}})\right]}(\cdot)$. Therefore, $\Phi_{x^{\|}}(x^{\|}) = SP(x^{\|})$.

7.4 Modular Symmetry

A special symmetry property for isomorphic module systems is now defined. Consider the following simple example.

Example 17 *Consider the set of isomorphic token passing modules* $\{G_1, G_2, G_3\}$ *shown in Figure 7.1.*

This example can be thought of as a controller for resource sharing modules $\{G_1, G_2, G_3\}$ *where module* G_i *would have exclusive access to a resource if and only*

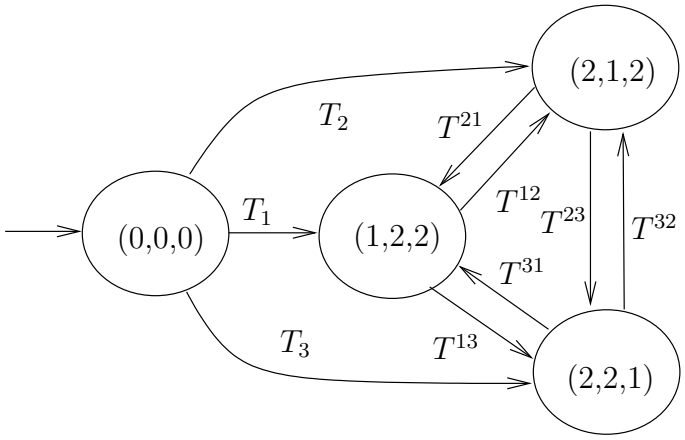

Figure 7.2: The composed $G_1 \| G_2 \| G_3$ for Example 17.

if it possesses the token. A module possesses a token if and only if it is in state 1. At initialization, all of the modules are at state 0 *and no modules possess tokens. On the occurrence of event* T_i*, a token is given to module* G_i *and module* G_i *enters state* 1*. The other modules enter state* 2*. The token is passed from module* G_i *to module* G_j *on the occurrence of event* T^{ij} *so that module* G_i *enters state* 2 *and* G_j *enters state* 1*. The composition* $G_1 \| G_2 \| G_3$ *can be seen in Figure 7.2.*

For $G_1 \| G_2 \| G_3$ *there are two classes of states that could be considered equivalent with respect to permutations of the component states. The two sets of equivalent state classes are* $\{(0,0,0)\}$ *and* $\{(1,2,2),(2,1,2),(2,2,1)\}$*. A valid state labelling for the* $G_1 \| G_2 \| G_3$ *automaton might be that states* $\{(1,2,2),(2,1,2),(2,2,1)\}$ *have proposition labelling* T *to signify there is exactly one token in the system and the state* $(0,0,0)$ *has proposition labelling* F *to signify that no modules possess tokens.*

Note that there is a transition $\delta^{\|}(x_a, \sigma) = x_b$ *if and only if for all state permutation operators* $\Phi(\cdot)$*, there exists an event* σ' *such that* $\delta^{\|}(\Phi(x_a), \sigma') = \Phi(x_b)$*. More specifically, consider the state transitions* $\delta^{\|}((0,0,0), T_1) = (1,2,2)$ *and* $\delta^{\|}((0,0,0), T_2) =$

$(2,1,2)$. *For this pair of global transitions the starting and ending states are permutation equivalent. Also, the events that drive these transitions are the same event with different indices. Now consider the state transitions* $\delta^{\|}((1,2,2),T^{12}) = (2,1,2)$ *and* $\delta^{\|}((2,2,1),T^{31}) = (1,2,2)$. *Note that the initial and final composed states in both of this pair of transitions are also permutations of each other and the events that drive these transitions are the same event with different indices.*

The intuition gained in Example 17 is that for some special isomorphic module systems, there is a type of symmetry such that there is a transition $\delta^{\|}(x_a,\sigma) = x_b$ if and only if for any state permutation operator, $\Phi(\cdot)$, there exists exactly one transition labelled by an event σ_Φ such that $\delta^{\|}(\Phi(x_a),\sigma_\Phi) = \Phi(x_b)$. Therefore, corresponding to $\Phi(\cdot)$ there should be an event translation operator $\Pi_\Phi : \Sigma \rightarrow \Sigma$ to calculate $\Pi_\Phi(\sigma)$ such that $\delta^{\|}(\Phi(x_a),\Pi_\Phi(\sigma)) = \Phi(x_b)$ for any other transition $\delta^{\|}(x_a,\sigma) = x_b$.

Consequently, for systems containing this kind of symmetry, if there is a transition between two states labelled by an event (that is $\delta^{\|}(x_a,\sigma) = x_b$), then for any permutation on those states ($\Phi(\cdot)$), another unique transition exists labelled by event ($\Pi_\Phi(\sigma)$) that can be computed by a translation operator ($\Pi_\Phi(\cdot)$) such that there is a transition between the permuted states driven by the computed event (that is, $\delta^{\|}(\Phi(x_a),\Pi_\Phi(\sigma)) = \Phi(x_b)$). The definition of the event translation operator $\Pi_\Phi(\cdot)$ should also be able to be extendable to strings of arbitrary length. This intuition is used to define a property below called *permutation symmetry* below that forces a large class of global properties to hold at states independent of the composed state orderings.

Let there be a set of permutation operators $\{\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)\}$ for the indices $i_1,j_1,\ldots,i_m,j_m \in \{1,\ldots,n\}$. Suppose there is a class of doubly indexed functions

$\{\pi_{ij}(\cdot) : \Sigma \to \Sigma | i, j \in \{1, \ldots, n\}\}$ and define the composition of these functions as was done above for the transposition operations. Namely:

$$\Pi_{[(i_1 j_1)(i_2 j_2)\cdots(i_m j_m)]}(\sigma) = \pi_{i_1 j_1}\left(\pi_{i_2 j_2}\left(\cdots \pi_{i_m j_m}\left(\sigma\right)\right)\right). \tag{7.4}$$

Definition 17 Modular State Permutation Symmetry*: Suppose there is a set of functions* $\{\pi_{ij}\}$ *such that for all strings of module index pairs* $(i_1 j_1)(i_2 j_2)\cdots(i_m j_m)$ *and* $(i'_1 j'_1)(i'_2 j'_2)\cdots(i'_{m'} j'_{m'})$ *such that*

$$\left(\Phi_{[(i_1 j_1)(i_2 j_2)\cdots(i_m j_m)]}(\cdot) = \Phi_{\left[(i'_1 j'_1)(i'_2 j'_2)\cdots(i'_{m'} j'_{m'})\right]}(\cdot)\right) \tag{7.5}$$

then

$$\left(\Pi_{[(i_1 j_1)(i_2 j_2)\cdots(i_m j_m)]}(\cdot) = \Pi_{\left[(i'_1 j'_1)(i'_2 j'_2)\cdots(i'_{m'} j'_{m'})\right]}(\cdot)\right). \tag{7.6}$$

A system composed of $\{G_1, \ldots, G_n\}$ *is said to have* modular state permutation symmetry with respect to $\{\pi_{ij}\}$ *(or* permutation symmetry *for short) if* $\forall i, j \in \{1, \ldots, n\}, x_a^{\|}, x_b^{\|} \in X^{\|}$,

$$\left(\delta^{\|}(x_a^{\|}, \sigma) = x_b^{\|}\right) \iff \left(\delta^{\|}(\phi_{ij}(x_a^{\|}), \pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})\right). \tag{7.7}$$

It is now discussed how the token passing system in Example 17 has permutation symmetry.

Example 18 *Consider the set of isomorphic token passing modules* $\{G_1, G_2, G_3\}$ *discussed in Example 17. According to Definition 17, it is required to define the event translation function in order to demonstrate permutation symmetry for this*

system. Suppose $i, j, k \in \{1, 2, 3\}$ *such that* i,j *and* k *are not equal:*

$$\pi_{ij}: \quad \left[\begin{array}{lcl} T_i & \rightarrow & T_j \\ T_j & \rightarrow & T_i \\ T_k & \rightarrow & T_k \\ T^{ij} & \rightarrow & T^{ji} \\ T^{ik} & \rightarrow & T^{jk} \\ T^{ji} & \rightarrow & T^{ij} \\ T^{jk} & \rightarrow & T^{ik} \\ T^{ki} & \rightarrow & T^{kj} \\ T^{kj} & \rightarrow & T^{ki} \end{array}\right].$$

With this definition of the $\{\pi_{ij}(\cdot)\}$ *mapping, it is straightforward to verify that the system* $G_1 \| G_2 \| G_3$ *is permutation symmetric with respect to* $\{\pi_{ij}(\cdot)\}$.

As a demonstration of the symmetry exhibited by the transition structure in this example, consider the $\phi_{13}(\cdot)$ *transposition operator. For the string of transitions labelled by* $T_1 T_{13} T_{32} T_{32} T_{21}$, *due to the* $\pi_{13}(\cdot)$ *mapping as defined above,*

$$\pi_{13}(T_1 T_{13} T_{32} T_{32} T_{21}) = T_3 T_{31} T_{12} T_{12} T_{23}.$$

Also,

$$\begin{aligned} \phi_{13}((0,0,0)) &= (0,0,0), \\ \phi_{13}((1,2,2)) &= (2,2,1). \end{aligned}$$

Because $G_1 \| G_2 \| G_3$ *is permutation symmetric with respect to* $\{\pi_{ij}\}$,

$$\begin{aligned} \delta^{G_1 \| G_2 \| G_3}((0,0,0), T_1 T_{13} T_{32} T_{32} T_{21}) &= (1,2,2), \\ \delta^{G_1 \| G_2 \| G_3}((0,0,0), T_3 T_{31} T_{12} T_{12} T_{23}) &= (2,2,1). \end{aligned}$$

The intuition behind the definition of permutation symmetry for $G^{\|}$ is that for any state transition in $G^{\|}$ between two states $x_a^{\|}, x_b^{\|}$ on the occurrence of an event σ, such that $\delta^{\|}(x_a^{\|}, \sigma) = x_b^{\|}$, then for any state transposition operator $\phi_{ij}(\cdot)$, there is an event translation function $\pi_{ij}(\cdot)$ such that $\delta^{\|}(\phi_{ij}(x_a^{\|}), \pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})$.

Also, because all permutation operators are compositions of transpositions operator, for any state permutation operator $\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$, corresponding event permutation operator $\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$, and permutation symmetric system $G^{\|}$, then $\forall x_a^{\|}, x_b^{\|} \in X^{\|}, \sigma \in \Sigma^{\|}$,

$$\left(\delta^{\|}(x_a^{\|}, \sigma) = x_b^{\|}\right) \iff \left(\delta^{\|}\left(\Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_a^{\|}), \Pi_{[(i_1j_1)\cdots(i_mj_m)]}(\sigma)\right) = \Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_b^{\|})\right). \tag{7.8}$$

Furthermore, due to Equations 7.5 and 7.6, the definition of permutation symmetry ensures that for any two strings of module index pairs $(i_1j_1)(i_2j_2)\cdots(i_mj_m)$ and $(i'_1j'_1)(i'_2j'_2)\cdots(i'_{m'}j'_{m'})$ that define the same permutation operators

$$\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot) = \Phi_{\left[(i'_1j'_1)(i'_2j'_2)\cdots(i'_{m'}j'_{m'})\right]}(\cdot)$$

then for the strings of pairs $(i_1j_1)(i_2j_2)\cdots(i_mj_m)$ and $(i'_1j'_1)(i'_2j'_2)\cdots(i'_{m'}j'_{m'})$ and an event σ there will be exactly one transition labelled by an event σ' such that

$$\sigma' = \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\sigma) = \Pi_{\left[(i'_1j'_1)(i'_2j'_2)\cdots(i'_{m'}j'_{m'})\right]}(\sigma).$$

Furthermore, the event σ' ensures that

$$\left(\delta^{\|}(x_a^{\|}, \sigma) = x_b^{\|}\right) \iff \left(\delta^{\|}\left(\Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_a^{\|}), \sigma'\right) = \Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_b^{\|})\right).$$

If in the definition of state permutation symmetry Equation 7.5 did not imply Equation 7.6, then it would be ambiguous if the unique transition corresponding to $\delta^{\|}\left(x_a^{\|}, \sigma\right) = x_b^{\|}$ from $\Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_a^{\|})$ to $\Phi_{[(i_1j_1)\cdots(i_mj_m)]}(x_b^{\|})$ is labelled by $\Pi_{[(i_1j_1)\cdots(i_mj_m)]}(\sigma)$ or $\Pi_{\left[(i'_1j'_1)\cdots(i'_mj'_m)\right]}(\sigma)$.

Note that for a permutation symmetric system with a set of event translation functions $\{\pi_{ij}(\cdot)\}$, the inverse functions $\Pi^{-1}_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$ Can be easily found because $\Phi^{-1}_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot) = \Phi_{[(i_mj_m)\cdots(i_2j_2)(i_1j_1)]}(\cdot)$. Therefore, due to the definition of permutation symmetry, $\Pi^{-1}_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot) = \Pi_{[(i_mj_m)\cdots(i_2j_2)(i_1j_1)]}(\cdot)$.

The subscripts $[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]$ on the functions $\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$ and $\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$ are sometimes dropped when it can be done so without ambiguity. If $\Phi_{x^\parallel}(\cdot) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$ then define $\Pi_{x^\parallel}(\cdot) = \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\cdot)$.

Using the notation of [17], a language $\mathcal{L}(G^\parallel)$ has a *symmetric group* of operators $S_\Sigma = \{\pi : \Sigma \to \Sigma\}$ if $\forall \pi \in S_\Sigma$, then $\mathcal{L}(G^\parallel) = \pi(\mathcal{L}(G^\parallel))$. This prompts the following lemma and theorem.

Lemma 11 *Suppose $\{G_1, \ldots, G_n\}$ has permutation symmetry with respect to $\{\pi_{ij}\}$. Then*

$$s \in \mathcal{L}(G^\parallel) \Rightarrow \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s) \in \mathcal{L}(G^\parallel). \tag{7.9}$$

Proof: From the definition of permutation symmetry,

$$\left(\delta^\parallel(x^\parallel_a, \sigma) = x^\parallel_b\right) \iff \left(\delta^\parallel(\phi_{ij}(x^\parallel_a), \pi_{ij}(\sigma)) = \phi_{ij}(x^\parallel_b)\right).$$

With the construction of Φ and Π and due to Equation 7.8,

$$\left(\delta^\parallel(x^\parallel_a, \sigma) = x^\parallel_b\right) \iff$$
$$\left(\delta^\parallel(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x^\parallel_a), \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\sigma)) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x^\parallel_b)\right).$$

The rest of this lemma is shown with proof by induction on the length of s that if $\delta^\parallel(x^\parallel_0, s) = x^\parallel_{|s|}$, then

$$\delta^\parallel(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x^\parallel_0), \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s)) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x^\parallel_{|s|})$$

and

$$s \in \mathcal{L}(G^{\|}) \Rightarrow \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s) \in \mathcal{L}(G^{\|}).$$

.

Base of induction: Suppose $s = \sigma$.

From above, if $\delta^{\|}(x_0^{\|}, \sigma) = x_1^{\|}$, then

$$\delta^{\|}(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_0^{\|}), \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\sigma)) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_1^{\|}).$$

Therefore,

$$\sigma \in \mathcal{L}(G^{\|}) \Rightarrow \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\sigma) \in \mathcal{L}(G^{\|}).$$

Induction hypothesis: Suppose $|s| = l$. If there exists a state $x_l^{\|}$ such that $\delta^{\|}(x_0^{\|}, s) = x_l^{\|}$ then

$$\delta^{\|}(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_0^{\|}), \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s)) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_l^{\|}).$$

and

$$s \in \mathcal{L}(G^{\|}) \Rightarrow \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s) \in \mathcal{L}(G^{\|}).$$

.

Induction step: Suppose $|s'| = l + 1$ and $s' = s\sigma$. Suppose there exists a state $x_{l+1}^{\|}$ such that $\delta^{\|}(x_0^{\|}, s') = x_{l+1}^{\|}$. Then there is also a state $x_l^{\|}$ such that $\delta^{\|}(x_0^{\|}, s) = x_l^{\|}$ and $\delta^{\|}(x_l^{\|}, \sigma) = x_{l+1}^{\|}$. From the induction hypothesis it is known that

$$\delta^{\|}(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_0^{\|}), \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s)) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_l^{\|})$$

and from Equation 7.8

$$\delta^{\|}(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_l^{\|}), \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\sigma)) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_{l+1}^{\|}).$$

Therefore, by combining these two results,

$$\delta^{\|}(\Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_0^{\|}), \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s')) = \Phi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(x_{l+1}^{\|}).$$

Hence,

$$s' \in \mathcal{L}(G^{\|}) \Rightarrow \Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s') \in \mathcal{L}(G^{\|}).$$

. ■

Theorem 26 *The set of operators* $\{\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}\}$ *constructed from* $\{\pi_{ij}\}$ *forms a symmetric group for the language* $\mathcal{L}(G^{\|})$ *if* $\{G_1, \ldots, G_n\}$ *has permutation symmetry with respect to* $\{\pi_{ij}\}$.

Proof: It is first shown that $\mathcal{L}(G^{\|}) \subseteq \Pi_{[(i_mj_m)\cdots(i_2j_2)(i_1j_1)]}(\mathcal{L}(G^{\|}))$. Suppose $s \in \mathcal{L}(G^{\|})$. This implies that $\Pi_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(s) \in \mathcal{L}(G^{\|})$ from Lemma 11. Therefore, $s \in \Pi^{-1}_{[(i_1j_1)(i_2j_2)\cdots(i_mj_m)]}(\mathcal{L}(G^{\|}))$. This implies that $\mathcal{L}(G^{\|}) \subseteq \Pi_{[(i_mj_m)\cdots(i_2j_2)(i_1j_1)]}(\mathcal{L}(G^{\|}))$. The reverse inclusion can be shown to hold by similar methods. ■

Theorem 26 shows that the language generated by an isomorphic module system with state permutation symmetry also contains a type of language symmetry with respect to the event translation operators $\{\pi_{ij}(\cdot)\}$. That is, for any i, j, $\pi_{ij}(\mathcal{L}(G^{\|})) = \mathcal{L}(G^{\|})$ even though the $\pi_{ij}(\cdot)$ function is not an identity map in general.

7.5 Permutation Symmetric μ-Calculus

In the most commonly accepted version of the μ-calculus as discussed in [14], a transition system $M = (S, T, AP, L)$ is given where S is a set of states, T is a set of transition classes $T \subseteq 2^{S\times S}$, AP is a set of atomic propositions and $L : S \to 2^{AP}$ is a state labelling function. A transition class $T_{\sigma_i} \in T, T_{\sigma_i} \subseteq S \times S$ can be thought of

as the set of all transitions of the same "type", similar to how transitions might be labelled in discrete-event systems. The μ-calculus system also uses a set of relational variables $VAR = \{Q_1, Q_2, \ldots\}$ where each relational variable $Q_i \in VAR$ can be assigned a subset of S. Alternatively, a relational variable can be thought of as a variable set of states $Q_i \subseteq S$. Following the notation used in [14], $e : VAR \to 2^S$ denotes an *environment* where states are assigned to the relational variables. Let Q be an arbitrary element of $\{Q_1, Q_2, \ldots\}$. The notation $e_{[Q \leftarrow W]}$ is used to denote a new environment that is the same as e except $e_{[Q \leftarrow W]}(Q) = W$. That is, with $e_{[Q \leftarrow W]}$, the states in W are assigned to Q.

The μ-calculus can be used to express a set of formulas and a μ-calculus formula f can be said to hold at some states in S, but not in others. The notation used is that $M, s \models f$ if the formula f holds at state s in M. The set $[\![f]\!]_M e$ denotes the states in M where f holds with environment e. The sets of formulas that can hold in the μ-calculus are now recursively defined simultaneously with some notational definitions.

- If $p \in AP$, then p is a formula. An atomic proposition holds at a state according to the state labelling function $L(\cdot)$.

$$[\![p]\!]_M e = \{s \in S | p \in L(s)\} \tag{7.10}$$

- A relational variable $Q_i \in VAR$ is a formula. A relational variable holds at a state if that state is assigned to the relational variable.

$$[\![Q]\!]_M e = e(Q) \tag{7.11}$$

- If f and g are formulas, then $\neg f$, $f \vee g$ and $f \wedge g$ are formulas. The formula $\neg f$ holds in the set of states where f does not hold, $f \vee g$ holds in the set of

states where f or g hold and $f \wedge g$ holds the set of states where f and g hold.

$$[\![\neg f]\!]_M e = S \setminus [\![f]\!]_M e \tag{7.12}$$

$$[\![f \wedge g]\!]_M e = [\![f]\!]_M e \cap [\![g]\!]_M e \tag{7.13}$$

$$[\![f \vee g]\!]_M e = [\![f]\!]_M e \cup [\![g]\!]_M e \tag{7.14}$$

- If f is a formula and $T_{\sigma_i} \in T$ then $[T_{\sigma_i}]f$ and $\langle T_{\sigma_i} \rangle$ are formulas. The formula $[T_{\sigma_i}]f$ holds at a state $s_1 \in S$ if for all $s_2 \in S$ such that $(s_1, s_2) \in T_{\sigma_i}$, f holds at s_2. Similarly, $\langle T_{\sigma_i} \rangle f$ holds at a state $s_1 \in S$ if there exists some $s_2 \in S$ such that $(s_1, s_2) \in T_{\sigma_i}$ and f holds at s_2.

$$[\![\langle T_{\sigma_i} \rangle f]\!]_M e = \{s_1 | \exists s_2 \left[((s_1, s_2) \in T_{\sigma_i}) \wedge (s_2 \in [\![f]\!]_M e) \right]\} \tag{7.15}$$

$$[\![[T_{\sigma_i}]\, f]\!]_M e = \{s_1 | \forall s_2 \left[((s_1, s_2) \in T_{\sigma_i}) \wedge (s_2 \in [\![f]\!]_M e) \right]\} \tag{7.16}$$

- If $Q \in VAR$ and f is a formula that is a function of Q, then $\mu Q.f$ and $\nu Q.f$ are formulas, provided that f is syntactically monotone with respect to Q. The least fixpoint of states $\mu Q.f$ is the set of states such that if Q holds in those states, then f also holds in those states. The greatest fixpoint $\nu Q.f$ is similarly defined. A formula f is said to be *syntactically monotone* with respect to a relational variable Q if all occurrences of Q fall under an even number of negations in f.

$$[\![\mu Q.f]\!]_M e \text{ is the least fixpoint of } \tau(W) = [\![f]\!]_M e_{[Q \leftarrow W]} \tag{7.17}$$

$$[\![\nu Q.f]\!]_M e \text{ is the greatest fixpoint of } \tau(W) = [\![f]\!]_M e_{[Q \leftarrow W]} \tag{7.18}$$

A function τ is monotonic if $S \subseteq S' \Rightarrow \tau(S) \subseteq \tau(S')$. Because of the monotonicity of all possible functions $\tau(W) = [\![f]\!]_M e_{[Q \leftarrow W]}$, there are simple methods for finding the least and greatest fixpoints. To find the least fixpoint, assign $W_0 := \emptyset, W_1 := \tau(W_0), \ldots$ and so on until $W_{i+1} = W_i$. Then W_i is the least fixpoint. For a k state system, the least fixed point will be found in less than k iterative compositions of the $\tau(\cdot)$ operation. Similarly, to find the greatest fixpoint, let $W_0 := S, W_1 := \tau(W_0), \ldots$ until the first i is found such that $W_{i+1} = W_i$. Then W_i is the greatest fixpoint and this fixpoint will be found in less than k steps in a k state system.

Several important properties of the transition system M can be easily expressed using the μ-calculus. For instance, for a transition system M, the expression $M, s \models \vee_i \langle T_{\sigma_i} \rangle True$ is used to denote that a state $s \in S$ does not deadlock. Also, $M, s \models \nu Q.\, (\vee_i \langle T_{\sigma_i} \rangle True) \wedge (\wedge_i [T_{\sigma_i}]\, Q)$ denotes that all states reachable from s are deadlock free. Furthermore, $M, s_0 \models \nu Q_1.\, (\mu Q_2.\, (m) \vee (\vee_i \langle T_{\sigma_i} \rangle Q_2)) \wedge (\wedge_i [T_{\sigma_i}]\, Q_1)$ can be used to express that all states reachable from s_0 can eventually lead to a state where an atomic proposition called marking (m) holds. In supervisory control theory, systems that satisfy this last property are said to be non-blocking.

Given a μ-calculus formula f, the *depth* of f (denoted by $depth(f)$) is defined recursively as follows. Let f_1 and f_2 be two μ-calculus formulas and let T_{σ_i} be a transition class.

- If $f \in AP$ or $f \in VAR$, then $depth(f) = 0$.
- If $f = \neg f_1$, $f = [T_{\sigma_i}] f_1$, $f = \langle T_{\sigma_i} \rangle f_1$, $f = \mu Q.f_1$ or $f = \nu Q.f_1$, then $depth(f) = depth(f_1) + 1$.
- If $f = f_1 \wedge f_2$ or $f = f_1 \vee f_2$, then $depth(f) = \max\{f_1, f_2\} + 1$.

Now that the standard μ-calculus and its properties have been introduced, it

is shown how the μ-calculus can be restricted to take advantage of permutation symmetry of systems such that two permutation equivalent states satisfy versions of the same μ-calculus formulas. This restriction of the μ-calculus for permutation symmetric systems and permutation symmetric properties is called the *permutation symmetric μ-calculus*.

As a small example how these permutation symmetric formulas could be useful, consider a system with permutation symmetry, such that module i always enters a failure state on the occurrence of local event σ_i when module i is in state x_1 and all other modules are in state x_2. In an n module system there are n permutations of the modular state $(x_1, x_2, \ldots, x_2)$ composed of one x_1 state and $(n-1)$ states labelled x_2. To verify that in this system that no σ_i event could occur in all permutations of the symmetric system structure, it would be sufficient to check n different permutations, but because of underlying system symmetry it is only needed to verify that $\neg\langle T_{\sigma_1}\rangle$ from state $(x_1, x_2, \ldots, x_2)$ and avoid verifying all of the other redundant symmetric propositions.

Motivated by the previous example, the transition classes and relational variables for $G^{\|}$ are defined so that μ-calculus formulas can be written such that for two states $x_a^{\|}, x_b^{\|} \in X^{\|}$ and μ-calculus formula f,
$\Big(\phi_{ij}(x_a^{\|}) = x_b^{\|}\Big) \Rightarrow \Big(G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f\Big)$.

The μ-calculus is restricted as follows.

- For all $x_a^{\|}, x_b^{\|} \in X^{\|}$ such that $SP(x_a^{\|}) = SP(x_b^{\|})$, $L(x_a^{\|}) = L(x_b^{\|})$.
- For all $x_a^{\|}, x_b^{\|} \in X^{\|}$ and $Q \in VAR$ such that $SP(x_a^{\|}) = SP(x_b^{\|})$, $x_a^{\|} \in Q \iff x_b^{\|} \in Q$.
- For $x_a^{\|}, x_b^{\|} \in X^{\|}$, $\sigma_1 \in \Sigma$, if $\delta^{\|}\Big(SP(x_a^{\|}), \sigma_1\Big) = \Phi_{x_a^{\|}}(x_b^{\|})$, then assign $(x_a^{\|}, x_b^{\|})$ to

T_{σ_1}.

For the first bullet the state labelling function was previously restricted to be permutation independent in Section 7.3. In the second bullet, the assignments to the relational variables must be permutation independent as with the state labelling function. The definition of the transition classes in the third bullet ensures that all permutation equivalent transitions (according to $\Phi(\cdot)$ and $\Pi(\cdot)$ mappings) must be in the same permutation class. Excepting these restrictions, formulas in the permutation symmetric μ-calculus can then be constructed in the usual manner. Permutation equivalent states in permutation symmetric systems satisfy the same permutation symmetric μ-calculus formulas.

Theorem 27 *Suppose a permutation symmetric system $G^{\|}$ is given with two states $x_a^{\|}, x_b^{\|} \in X^{\|}$ and a permutation symmetric μ-calculus formula f. Then*

$$\left(\phi_{ij}(x_a^{\|}) = x_b^{\|}\right) \Rightarrow \left(G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f\right). \tag{7.19}$$

Proof: This theorem is demonstrated using a proof by generalized induction on the depth of f.

Base of induction: Suppose $depth(f) = 0$. Then f is either an atomic proposition or a relational variable. Due to the restrictions on the permutation symmetric systems and permutation symmetric μ-calculus, the proposition labelling and relational variable assignments for $x_a^{\|}$ and $x_b^{\|}$ must be identical.

Induction hypothesis: For a formula f such that $depth(f) \leq n$, $\phi_{ij}(x_a^{\|}) = x_b^{\|} \Rightarrow \left(G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f\right)$.

Induction step: Suppose $depth(f) = n + 1$. The induction step is demonstrated in 7 cases. Let f_1 and f_2 be any formulas such that $depth(f_1) = n$ and $depth(f_2) \leq n$. From the induction hypothesis it is known that $\phi_{ij}(x_a^{\|}) = x_b^{\|} \Rightarrow$

$\Big(G^{\|}, x_a^{\|} \models f_1 \iff G^{\|}, x_b^{\|} \models f_1\Big) \wedge \Big(G^{\|}, x_a^{\|} \models f_2 \iff G^{\|}, x_b^{\|} \models f_2\Big)$ Suppose in general that $\phi_{ij}(x_a^{\|}) = x_b^{\|}$.

Case 1 *:* $f = \neg f_1$.

It is already known from the induction hypothesis that

$$G^{\|}, x_a^{\|} \models f_1 \iff G^{\|}, x_b^{\|} \models f_1$$

$\Rightarrow G^{\|}, x_a^{\|} \not\models f_1 \iff G^{\|}, x_b^{\|} \not\models f_1$

$\Rightarrow G^{\|}, x_a^{\|} \models \neg f_1 \iff G^{\|}, x_b^{\|} \models \neg f_1$

$\Rightarrow G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f.$

Case 2 *:* $f = f_1 \wedge f_2$.

It is already known from the induction hypothesis that

$$\Big(G^{\|}, x_a^{\|} \models f_1 \iff G^{\|}, x_b^{\|} \models f_1\Big) \wedge \Big(G^{\|}, x_a^{\|} \models f_2 \iff G^{\|}, x_b^{\|} \models f_2\Big)$$

$\Rightarrow \Big(G^{\|}, x_a^{\|} \models f_1 \wedge f_2 \iff G^{\|}, x_b^{\|} \models f_1 \wedge f_2\Big)$

$\Rightarrow G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f.$

Case 3 *:* $f = f_1 \vee f_2$.

It is already known from the induction hypothesis that

$$\Big(G^{\|}, x_a^{\|} \models f_1 \iff G^{\|}, x_b^{\|} \models f_1\Big) \wedge \Big(G^{\|}, x_a^{\|} \models f_2 \iff G^{\|}, x_b^{\|} \models f_2\Big)$$

$\Rightarrow \Big(G^{\|}, x_a^{\|} \models f_1 \vee f_2 \iff G^{\|}, x_b^{\|} \models f_1 \vee f_2\Big)$

$\Rightarrow G^{\|}, x_a^{\|} \models f \iff G^{\|}, x_b^{\|} \models f.$

Case 4 *:* $f = \langle T_{\sigma_i} \rangle f_1$.

Suppose that $\phi_{ij}({x'}^{\|}_a) = {x'}^{\|}_b$. Due to the definition of T_{σ_i} in the restricted μ-calculus, $(x^{\|}_a, {x'}^{\|}_a) \in T_{\sigma_i} \iff (x^{\|}_b, {x'}^{\|}_b) \in T_{\sigma_i}$. It is already known from the induction hypothesis that $\left(G^{\|}, {x'}^{\|}_a \models f_1\right) \iff \left(G^{\|}, {x'}^{\|}_b \models f_1\right)$

$$\Rightarrow \left(\begin{array}{l} \left(\left(\exists {x'}^{\|}_a | (x^{\|}_a, {x'}^{\|}_a) \in T_{\sigma_i}\right) \iff \left(\exists {x'}^{\|}_b | (x^{\|}_b, {x'}^{\|}_b) \in T_{\sigma_i}\right)\right) \wedge \\ \left(G^{\|}, {x'}^{\|}_a \models f_1 \iff G^{\|}, {x'}^{\|}_b \models f_1\right) \end{array} \right)$$

$$\Rightarrow G^{\|}, x^{\|}_a \models \langle T_{\sigma_i} \rangle f_1 \iff G^{\|}, x^{\|}_b \models \langle T_{\sigma_i} \rangle f_1$$

$$\Rightarrow G^{\|}, x^{\|}_a \models f \iff G^{\|}, x^{\|}_b \models f.$$

Case 5 *:* $f = [T_{\sigma_i}] f_1$.

For all ${x'}^{\|}_a, {x'}^{\|}_b$ such that $\phi_{ij}({x'}^{\|}_a) = {x'}^{\|}_b$, $(x^{\|}_a, {x'}^{\|}_a) \in T_{\sigma_i} \iff (x^{\|}_b, {x'}^{\|}_b) \in T_{\sigma_i}$ due to the definition of T_{σ_i} in the restricted μ-calculus. It is already known from the induction hypothesis that $G^{\|}, {x'}^{\|}_a \models f_1 \iff G^{\|}, {x'}^{\|}_b \models f_1$

$$\Rightarrow \left(\begin{array}{l} \left(\left(\forall {x'}^{\|}_a | (x^{\|}_a, {x'}^{\|}_a) \in T_{\sigma_i}\right) \iff \left(\forall {x'}^{\|}_b | (x^{\|}_b, {x'}^{\|}_b) \in T_{\sigma_i}\right)\right) \wedge \\ \left(G^{\|}, {x'}^{\|}_a \models f_1 \iff G^{\|}, {x'}^{\|}_b \models f_1\right) \end{array} \right)$$

$$\Rightarrow G^{\|}, x^{\|}_a \models [T_{\sigma_i}] f_1 \iff G^{\|}, x^{\|}_b \models [T_{\sigma_i}] f_1$$

$$\Rightarrow G^{\|}, x^{\|}_a \models f \iff G^{\|}, x^{\|}_b \models f.$$

Case 6 *:* $f = \mu Q.f_1$.

Let $\tau(W) = [\![f_1]\!]_{G^{\|}} e_{[Q \leftarrow W]}$. It is already known from the induction hypothesis that $G^{\|}, x^{\|}_a \models f_1 \iff G^{\|}, x^{\|}_b \models f_1$, so that for any valid permutation symmetric evaluation $e_{[Q \leftarrow W]}$ in the restricted μ-calculus, it is known from the induction hypothesis that $x^{\|}_a \in \tau(W) \iff x^{\|}_b \in \tau(W)$. Let $[\![\mu Q.f]\!]_{G^{\|}} e$ be the least fixpoint of $\tau(W)$. Therefore, $x^{\|}_a \in [\![\mu Q.f]\!]_{G^{\|}} e \iff x^{\|}_b \in [\![\mu Q.f]\!]_{G^{\|}} e$.

Case 7 *:* $f = \nu Q.f_1$.

This case can be demonstrated using an argument similar to that for Case 6. This completes the proof by induction. ■

7.6 Permutation Symmetry Quotient Automata

Given a set of automata $\{G_1, \ldots, G_n\}$ such that the size of their respective state spaces is bounded by k, the composed automaton $G_1 \| \cdots \| G_n$ has k^n reachable states in the worst case. As n grows, this state space can become unbearably large and procedures that require an enumeration over all of the reachable states of $G_1 \| \cdots \| G_n$ would take a lot of time. Therefore procedures for solving many control and verification problems on the composed system $G_1 \| \cdots \| G_n$ are time intensive using the currently known methods.

However, for many types of systems a quotient automaton construction based on state permutation equivalences can be used to greatly decrease computation time when testing if properties hold at various system states [14]. In general, defining a quotient automaton with the smallest state space for verifying system properties for a given system is generally computationally difficult. In this section a nondeterministic quotient automaton $\vec{G}$ that avoids these computational difficulties is constructed from the modules $\{G_1, \ldots, G_n\}$. A version of this quotient automaton for the standard supervisory control model was shown in [17, 62]. In [17] it was not shown how this quotient automaton could be used to more efficiently verify μ-calculus properties of systems.

The automaton $\vec{G}$ has a predefined quotient structure that does not need to be computed. Although this quotient structure may not have the smallest state space, it generally leads to a significant decrease in the computational difficulty of testing symmetric μ-calculus formulas on permutation symmetric systems.

The automaton $\vec{G}$ is a 6-tuple $\vec{G} = \left(\vec{X}, \vec{x_0}, AP, L^{\|}, \Sigma^{\|}, \vec{\delta}\right)$ such that $\vec{G}$ uses the set of the standard permutations $\vec{X}$ as defined above as its state space. A state $\vec{x}$ in

$\vec{G}$ is also in $G^{\|}$, so the same set of atomic propositions and state labelling function are used for $\vec{G}$ and $G^{\|}$. Let $\vec{x_o}$ be the initial state of $G_1 \| \cdots \| G_n$. Assume that the transition structures of $\{G_1, \cdots, G_n\}$ are generalized such that if $\sigma \notin \Sigma_i$, then $\forall x \in X, \delta_i(x, \sigma) = x$. The nondeterministic state transition function $\vec{\delta} : \vec{X} \rightarrow \vec{X}$ is defined as follows:

$$\vec{\delta}(\vec{x}, \sigma) = \left\{ \begin{array}{ll} SP\left((\delta_1(\vec{x}^1, \sigma), \ldots, \delta_n(\vec{x}^n, \sigma))\right) & \text{if } \delta_1(\vec{x}^1, \sigma)! \wedge \cdots \wedge \delta_n(\vec{x}^n, \sigma)! \\ \text{undefined otherwise.} & \end{array} \right\}$$

The definition of $\vec{\delta}(\cdot, \cdot)$ can be extended to allows strings of arbitrary length using the usual methods. The isomorphic module system introduced in Example 17 is now used to demonstrate the construction of a reduced state space composed automaton $\vec{G}$.

Example 19 *Consider the isomorphic module system G_1, G_2, G_3 introduced in Example 17 above. The set of standard permutations of the sets of equivalent states is $\{(0,0,0),(1,2,2)\}$. The reduced state space composed automaton $\vec{G}$ can be seen in Figure 7.3.*

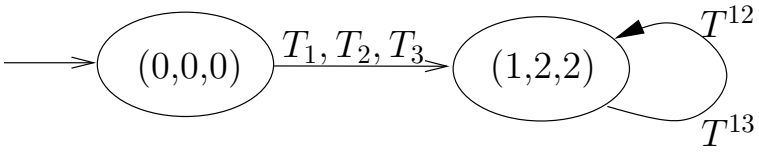

Figure 7.3: The automaton $\vec{G}$ constructed from G_1, G_2, G_3.

7.7 Verifying Symmetric μ-Calculus Formulas

It is now shown that the $\vec{G}$ automaton constructed from the set of modules $\{G_1, \ldots, G_n\}$ can be used to test permutation symmetric μ-calculus propositions in

the original $G^{\|}$ system. Suppose there is a formula f in the restricted μ-calculus and it is desired to test if for a state $x^{\|}$ of $G^{\|}$ whether or not $G^{\|}, x^{\|} \models f$. Suppose also that $\{T_{\sigma_1}, \ldots\}$ is the set of transition classes of $G^{\|}$ for the restricted μ-calculus defined above. First, the automaton $\vec{G}$ is constructed from $\{G_1, \ldots, G_n\}$. The set of transition classes $\{\vec{T}_{\sigma_1}, \ldots\}$ is then constructed such that for all $(\vec{x}_1, x_2^{\|}) \in T_{\sigma_i}$, $(\vec{x}_1, SP(x_2^{\|}))$ is assigned to $\vec{T}_{\sigma_i}$.

Finally construct a μ-calculus formula $\vec{f}$ from f, $\{T_{\sigma_1}, \ldots\}$ and $\{\vec{T}_{\sigma_1}, \ldots\}$ such that $\vec{f}$ is a copy of f with any occurrence of T_{σ_i} in f replaced with the corresponding $\vec{T}_{\sigma_i}$. This formula construction can be used to test if $G^{\|}, \vec{x} \models f$ as indicated in Theorem 28.

Theorem 28 *Let $\vec{x}$ be a state of a permutation symmetric system $G^{\|}$ and let f be a permutation symmetric μ-calculus formula. From this system construct $\vec{G}$, $\{\vec{T}_{\sigma_1}, \ldots\}$ and $\vec{f}$ as described above. Then,*

$$\left(G^{\|}\vec{x} \models f\right) \iff \left(\vec{G}\vec{x} \models \vec{f}\right). \tag{7.20}$$

Proof: This theorem is demonstrated using a proof by generalized induction on the depth of f.

Base of induction:

Suppose $depth(f) = 0$. Then f is either an atomic proposition or a relational variable. Due to the construction of $G^{\|}$ and $\vec{G}$, the proposition labelling and relational variable assignments for $\vec{x}$ are identical in both systems.

Induction hypothesis:

For a formula f such that $depth(f) \leq n$, $\left(G^{\|}\vec{x} \models f\right) \iff \left(\vec{G}\vec{x} \models \vec{f}\right)$

Induction step:

Suppose $depth(f) = n + 1$. The induction step is demonstrated in 7 cases. Let f_1

and f_2 be any formulas such that $depth(f_1) = n$ and $depth(f_2) \leq n$. It is known from the induction hypothesis that $\left(G^{\|}\vec{x} \models f_1\right) \iff \left(\vec{G}\vec{x} \models \vec{f_1}\right)$ and $\left(G^{\|}\vec{x} \models f_1\right) \iff \left(\vec{G}\vec{x} \models \vec{f_1}\right)$.

Case 1 *:* $f = \neg f_1$.

$$\begin{aligned}
&\left(G^{\|}\vec{x} \models f_1\right) \iff \left(\vec{G}\vec{x} \models \vec{f_1}\right) \\
&\Rightarrow \left(G^{\|}\vec{x} \not\models f_1\right) \iff \left(\vec{G}\vec{x} \not\models \vec{f_1}\right) \\
&\Rightarrow \left(G^{\|}\vec{x} \models \neg f_1\right) \iff \left(\vec{G}\vec{x} \models \neg\vec{f_1}\right).
\end{aligned}$$

Case 2 *:* $f = f_1 \wedge f_2$.

$$\begin{aligned}
&\left(\left(G^{\|}\vec{x} \models f_1\right) \iff \left(\vec{G}\vec{x} \models \vec{f_1}\right)\right) \wedge \left(\left(G^{\|}\vec{x} \models f_2\right) \iff \left(\vec{G}\vec{x} \models \vec{f_2}\right)\right) \\
&\Rightarrow \left(\left(G^{\|}\vec{x} \models f_1\right) \wedge \left(G^{\|}\vec{x} \models f_2\right)\right) \iff \left(\left(\vec{G}\vec{x} \models \vec{f_1}\right) \wedge \left(\vec{G}\vec{x} \models \vec{f_2}\right)\right) \\
&\Rightarrow \left(\left(G^{\|}\vec{x} \models f_1 \wedge f_2\right) \iff \left(\vec{G}\vec{x} \models \vec{f_1} \wedge \vec{f_2}\right)\right)
\end{aligned}$$

Case 3 *:* $f = f_1 \vee f_2$.

$$\begin{aligned}
&\left(\left(G^{\|}\vec{x} \models f_1\right) \iff \left(\vec{G}\vec{x} \models \vec{f_1}\right)\right) \wedge \left(\left(G^{\|}\vec{x} \models f_2\right) \iff \left(\vec{G}\vec{x} \models \vec{f_2}\right)\right) \\
&\Rightarrow \left(\left(G^{\|}\vec{x} \models f_1\right) \vee \left(G^{\|}\vec{x} \models f_2\right)\right) \iff \left(\left(\vec{G}\vec{x} \models \vec{f_1}\right) \vee \left(\vec{G}\vec{x} \models \vec{f_2}\right)\right) \\
&\Rightarrow \left(\left(G^{\|}\vec{x} \models f_1 \vee f_2\right) \iff \left(\vec{G}\vec{x} \models \vec{f_1} \vee \vec{f_2}\right)\right)
\end{aligned}$$

Case 4 *:* $f = \langle T_{\sigma_i}\rangle f_1$.

Suppose that $G^{\|}\vec{x} \models \langle T_{\sigma_i}\rangle f_1$. Then there is a state $x_1^{\|}$ such that $G^{\|}x_1^{\|} \models f_1$ and $(\vec{x}, x_1^{\|}) \in T_{\sigma_i}$. Due to the definition of $\vec{T}_{\sigma_i}$, $(\vec{x}, SP(x_1^{\|})) \in \vec{T}_{\sigma_i}$. From Theorem 27, $G^{\|}SP(x_1^{\|}) \models f_1$ and because of the induction hypothesis, $\vec{G}SP(x^{\|}) \models \vec{f_1}$. Therefore, $\vec{G}\vec{x} \models \langle\vec{T}_{\sigma_i}\rangle\vec{f_1}$.

Now, suppose that $\vec{G}\vec{x} \models \langle\vec{T}_{\sigma_i}\rangle\vec{f_1}$. Therefore, there is a state $\vec{x}'$ such that $(\vec{x}, \vec{x}') \in \vec{T}_{\sigma_i}$ and $\vec{G}\vec{x}' \models \vec{f_1}$. By the induction hypothesis, $G^{\|}\vec{x}' \models f_1$. By the definition of

$\vec{T}_{\sigma_i}$, there is a state $x^{\|}$ such that $SP(x^{\|}) = \vec{x}'$ and $(\vec{x}, x^{\|}) \in T_{\sigma_i}$. Furthermore, by Theorem 27, $G^{\|}x^{\|} \models f_1$. Therefore, $G^{\|}\vec{x} \models \langle T_{\sigma_i}\rangle f_1$.

Overall, $\left(G^{\|}\vec{x} \models \langle T_{\sigma_i}\rangle f_1\right) \iff \left(\vec{G}\vec{x} \models \langle \vec{T}_{\sigma_i}\rangle \vec{f_1}\right)$.

Case 5 *:* $f = [T_{\sigma_i}]f_1$.

Suppose that $G^{\|}\vec{x} \not\models [T_{\sigma_i}]f_1$. Then there is a state $x_1^{\|}$ such that $G^{\|}x_1^{\|} \not\models f_1$ and $(\vec{x}, x_1^{\|}) \in T_{\sigma_i}$. Due to the definition of $\vec{T}_{\sigma_i}$, $(\vec{x}, SP(x_1^{\|})) \in \vec{T}_{\sigma_i}$. From Theorem 27, $G^{\|}SP(x_1^{\|}) \not\models f_1$ and because of the induction hypothesis, $\vec{G}SP(x_1^{\|}) \not\models \vec{f_1}$. Therefore, $\vec{G}\vec{x} \not\models \langle \vec{T}_{\sigma_i}\rangle \vec{f_1}$.

Now, suppose that $\vec{G}\vec{x} \not\models [\vec{T}_{\sigma_i}]\vec{f_1}$. Therefore, there is a state $\vec{x}'$ such that $(\vec{x}, \vec{x}') \in \vec{T}_{\sigma_i}$ and $\vec{G}\vec{x}' \not\models \vec{f_1}$. By the induction hypothesis, $G^{\|}\vec{x}' \not\models f_1$. By the definition of $\vec{T}_{\sigma_i}$, there is a state $x_2^{\|}$ such that $SP(x_2^{\|}) = \vec{x}'$ and $(\vec{x}, x_2^{\|}) \in T_{\sigma_i}$. Furthermore, by Theorem 27, $G^{\|}x_2^{\|} \not\models f_1$. Therefore, $G^{\|}\vec{x} \not\models [T_{\sigma_i}]f_1$.

Overall, $\left(G^{\|}\vec{x} \models [T_{\sigma_i}]f_1\right) \iff \left(\vec{G}\vec{x} \models [\vec{T}_{\sigma_i}]\vec{f_1}\right)$.

Case 6 *:* $f = \mu Q.f_1$.

Let $W^{\|}$ be a permutation symmetric relational variable assignment for $G^{\|}$ and let $\vec{W} = SP(W^{\|})$. Define $\tau^{\|}(W^{\|}) = [\![f_1]\!]_{G^{\|}} e_{[Q \leftarrow W^{\|}]}$ and $\vec{\tau}(\vec{W}) = [\![f_1]\!]_{\vec{G}} e_{[Q \leftarrow \vec{W}]}$. It is known from the induction hypothesis and Theorem 27 that if $SP(W^{\|}) = \vec{W}$ then $SP(\tau^{\|}(W^{\|})) = \vec{\tau}(\vec{W})$ and $\tau^{\|}(W^{\|})$ must be permutation symmetric. Therefore define the following series:

$$W_i^{\|} = \begin{cases} \emptyset & \text{if } i = 0 \\ \tau^{\|}(W_{i-1}^{\|}) & \text{if } i > 0 \end{cases} \tag{7.21}$$

$$\vec{W}_i = \begin{cases} \emptyset & \text{if } i = 0 \\ \vec{\tau}(\vec{W}_{i-1}) & \text{if } i > 0 \end{cases} \tag{7.22}$$

$SP(W_0^{\|}) = \vec{W}_0$

$\Rightarrow SP(W_1^{\|}) = \vec{W}_1$

$\Rightarrow SP(W_2^{\|}) = \vec{W}_2$

$\ldots$

$\Rightarrow SP(W_i^{\|}) = \vec{W}_i.$

Also, because of the inherent permutation symmetry of $W_i^{\|}$,

$$\left(\emptyset = W_i^{\|} \setminus W_{i-1}^{\|}\right) \iff \left(\emptyset = SP(W_i^{\|}) \setminus SP(W_{i-1}^{\|})\right).$$

Therefore, $\left(W_i^{\|} = W_{i-1}^{\|}\right) \iff \left(\vec{W}_i = \vec{W}_{i-1}\right)$ Hence, the least fixpoint of $\tau^{\|}(W^{\|})$ corresponds to the least fixpoint of $\vec{\tau}(\vec{W})$. So, if $[\![\mu Q.f]\!]_{G^{\|}} e$ is the least fixpoint of $\tau^{\|}(W^{\|})$ and $[\![\mu Q.f]\!]_{\vec{G}} e$ is the least fixpoint of $\vec{\tau}(\vec{W})$, then $x^{\|} \in [\![\mu Q.f_1]\!]_{G^{\|}} e \iff \vec{x} \in [\![\mu Q.\vec{f_1}]\!]_{\vec{G}} e$.

Case 7 *:* $f = \nu Q.f_1$.

This case follows from the same reasoning for Case 6. Redefine the following series:

$$W_i^{\|} = \begin{cases} X^{\|} & \text{if } i = 0 \\ \tau^{\|}(W_{i-1}^{\|}) & \text{if } i > 0 \end{cases} \tag{7.23}$$

$$\vec{W}_i = \begin{cases} SP(X^{\|}) & \text{if } i = 0 \\ \vec{\tau}(\vec{W}_{i-1}) & \text{if } i > 0 \end{cases} \tag{7.24}$$

$SP(W_0^{\|}) = \vec{W}_0$

$\Rightarrow SP(W_1^{\|}) = \vec{W}_1$

$\Rightarrow SP(W_2^{\|}) = \vec{W}_2$

$\ldots$

$\Rightarrow SP(W_i^{\|}) = \vec{W}_i.$

Also, because of the inherent permutation symmetry of $W_i^{\|}$,

$$\left(\emptyset = W_{i-1}^{\|} \setminus W_i^{\|}\right) \iff \left(\emptyset = SP(W_{i-1}^{\|}) \setminus SP(W_i^{\|})\right).$$

Therefore, $\left(W_i^{\|} = W_{i-1}^{\|}\right) \iff \left(\vec{W}_i = \vec{W}_{i-1}\right)$ Therefore, the greatest fixpoint of $\tau^{\|}(W^{\|})$ corresponds to the greatest fixpoint of $\vec{\tau}(\vec{W})$. So, if $[\![\nu Q.f]\!]_{G^{\|}} e$ is the greatest fixpoint of $\tau^{\|}(W^{\|})$ and $[\![\nu Q.f]\!]_{\vec{G}} e$ is the greatest fixpoint of $\vec{\tau}(\vec{W})$, then $x^{\|} \in [\![\nu Q.f_1]\!]_{G^{\|}} e \iff \vec{x} \in [\![\nu Q.\vec{f_1}]\!]_{\vec{G}} e$. This completes the proof by induction. ■

Due to the permutation symmetry of $G^{\|}$ and f with Theorem 27 and Theorem 28, if f holds at a state $\vec{x}$, then f holds at all states in the same permutation equivalence class.

Corollary 10 *Let $x^{\|}$ be a state of a permutation symmetric system $G^{\|}$ and let f be a permutation symmetric μ-calculus formula. From this system construct $\vec{G}$, $\{\vec{T}_{\sigma_1}, \ldots\}$ and $\vec{f}$ as described above and let $\vec{x}$ be $SP(x^{\|})$. Then,*

$$\left(G^{\|} x^{\|} \models f\right) \iff \left(\vec{G}\vec{x} \models \vec{f}\right). \tag{7.25}$$

This corollary is important because the symmetric μ-calculus is very powerful and is more general than many other common logics such as CTL*, CTL and LTL. The $\vec{G}$ automaton also has a much smaller state space than the $G^{\|}$ automaton, so the state explosion problem inherent to many modular systems is not as problematic; this is demonstrated below in an example in Section 7.8.

Given a set of modules $\{G_1, \ldots, G_n\}$ such that $G^{\|}$ is permutation symmetric, what is being saved through the use of the quotient automaton $\vec{G}$? What is an upper limit on the size of the state space of $\vec{G}$? This question is equivalent to the question from bag theory [50] where given a bag that can contain n objects and a set of k elements, how many ways are there to fill the bag? This problem can be reduced

to a bin and ball problem from combinatorics where given $(k-1)$ balls and $(n+1)$ bins, how many unique ways are there to fill the bins? This problem is discussed in [70] where it is shown that there are

$$\left(\begin{array}{c} k+n-1 \\ \\ n \end{array} \right) = \left(\frac{(k+n-1)!}{(k-1)!n!} \right) \tag{7.26}$$

ways to fill the bins and therefore there are the same number of classes of states that need to be verified to check a global property.

Although Equation 7.26 indicates that there are still a large number of classes of states that need to be verified, it is rather smaller than k^n and generally there is a reduction in state space on the order of $k!$. The reduction in computational difficulty for verification is demonstrated with an example in the next section.

7.8 UAV Leader Selection Protocol Example

Another example of a permutation symmetric system is now presented. Consider a platoon of UAV's that are assigned to scout over some unknown territory. The UAV's are all identical to one another and have three modes of operation: regular, leader and failed. Every platoon should have at most one leader that coordinates the behavior of the other members of the platoon and in turn communicates with the platoon's home base. Because the platoon members are identical, all UAV's have the capability to become leaders. However, it is assumed that during platoon operation, a failure event may occur in a leader causing it to enter a failure state. Once a UAV is in a failure state it may be able to reset into the regular mode of operation.

A permutation symmetric leader selection protocol for an UAV swarm is discussed and can be seen in Figure 7.4. A UAV is in regular mode if it is in an R^i state, leader mode if it is in an L^i state and in failure mode if it is in state F.

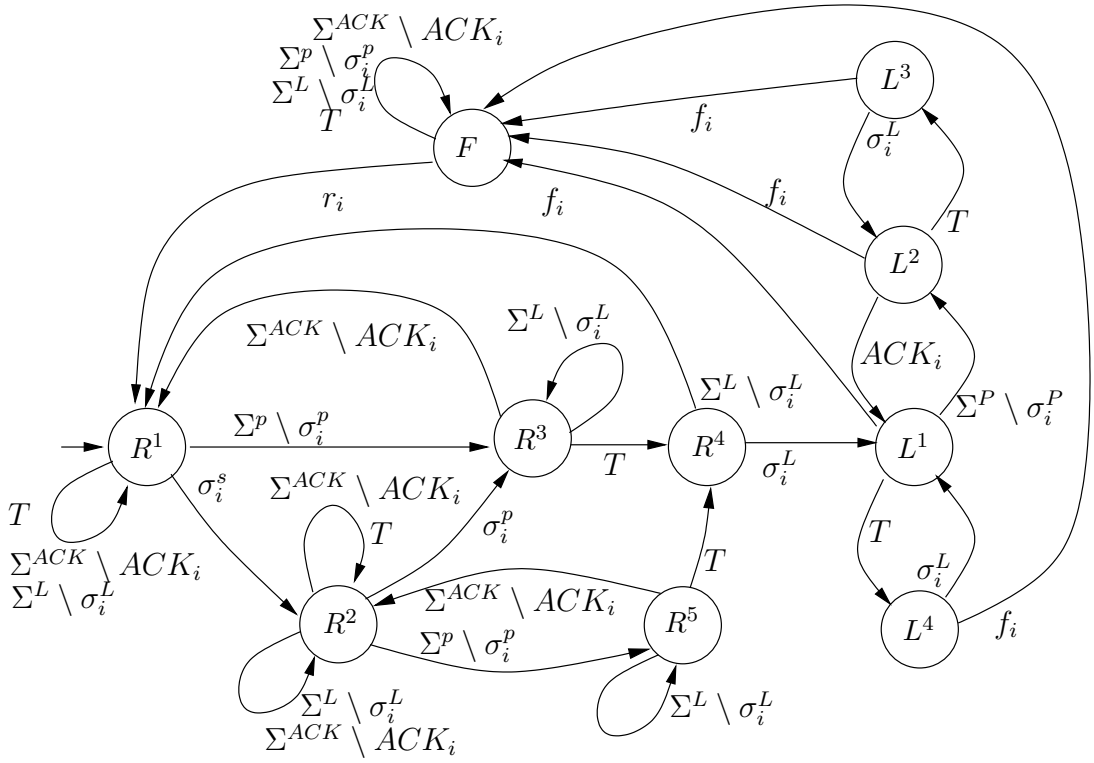

Figure 7.4: High Level UAV Swarm Leader Protocol for Platoon Member i.

Starting from the initial state in the regular mode of operation, when a private event σ_i^S occurs in a UAV, the UAV is prompted to broadcast σ_i^p to the other members of the platoon. It is assumed that the platoon always remains in radio contact with one another and that transmissions are never lost. When a leader j observes the broadcast event σ_i^p, it should be acknowledged with the ACK_j event. If a σ_i^p event is not acknowledged within a certain time, the timer event global broadcast T occurs to signal to the platoon members that a new leader may need to be selected as the previous leader j may have entered a failure state due to the occurrence of a private failure f_j. If the previous leader j has not entered the failure state, it broadcasts σ_j^L to signify that it is still the leader. On the absence of an event in $\Sigma^L \setminus \sigma_i^L$, platoon member i then broadcasts σ_i^L to signify that it is declaring itself the new platoon leader, and on the observation of this event, all other regular platoon members return to the initial state. If UAV i enters a failure state, it is reset on the occurrence of r_i to the initial state. Note that at initialization no platoon members are declared leader.

The events Σ^{ACK} are used to denote the set of acknowledgment events that could be sent by the various modules when they are in leader mode. The set Σ^p denotes the events the modules broadcast when in regular mode to communicate with a leader and Σ^L is the set of events the modules broadcast to declare that they are in leader mode. Note that Σ^{ACK}, Σ^p and Σ^L all denote broadcast-type events that occur in all of the modules, but on different transitions in each module. For instance, the occurrence of σ_i^L in state R^4 of module i means that module i transitions to leader mode and state L^1. However, the occurrence of σ_i^L in state R^4 of module j means that module j remains in regular mode and returns to state R^1. Using the $\Psi_{ij}(\cdot)$ notation introduced above, $\Psi_{ij}(\sigma_i^L) = \sigma_j^L$ and $\Psi_{ij}(\sigma_j^L) = \sigma_i^L$. This relationship with

the translation mapping also holds for events in Σ^{ACK} and Σ^{p}.

The set of failure events and reset events are denoted by Σ^{f} and Σ^{r} respectively. The set Σ^{s} denotes the set of private events for each module to internally signal it would like to broadcast a Σ^{p} event to the platoon. The events in Σ^{f}, Σ^{r} and Σ^{s} are private to their respective modules and occur in exactly one module each. Using the $\Psi_{ij}(\cdot)$ notation introduced above, $\Psi_{ij}(f_i) = f_j$ and $\Psi_{ij}(f_j)$ is not defined. This relationship with the translation mapping also holds for events in Σ^{r} and Σ^{s}.

The timer alarm event T is a global unindexed event that occurs in all modules simultaneously and for identical state transitions. Therefore, $\Psi_{ij}(T) = T$.

The set of system events for platoon member i is defined to be $\Sigma_i = \{T, f_i, r_i\} \cup \Sigma^{L} \cup \Sigma^{ACK} \cup \Sigma^{P}$.

The system described in this example is not in the class of systems discussed in [62] as there are events that are broadcast between the modules of the system. For example, the Σ^{ACK} events are broadcast between modules, but they are not "global" events in the sense of [62].

For this example an event translation function can be defined as follows for $\sigma \in \Sigma$:

$$\pi_{ij}(\sigma) = \begin{cases} \Psi_{ij}(\sigma) & \text{if} \quad \sigma \in \Sigma_i \\ \Psi_{ji}(\sigma) & \text{if} \quad \sigma \in \Sigma_j \\ \sigma & \text{if} \quad \text{otherwise} \end{cases}$$

Using the π_{ij} definition above, the UAV leader selection protocol in Figure 7.4 is permutation symmetric as defined in Definition 17. Therefore, a quotient automaton can be constructed for testing system properties such as "It is possible for two modules to enter leader mode at the same time". For even relatively small systems like the one in this example where the modules have 10 local states there is a significant

reduction in the computational difficulty of testing system properties due to use of the quotient automaton. For a small platoon of 4 UAV's, the normal $G^{\|}$ automaton constructed from the composition of the automaton in Figure 7.4 has 2707 states while the reduced $\vec{G}$ automaton constructed for this example has only 246 states. (These results were found using the UMDES toolbox.) This is over a factor of 10 reduction in the automaton's complexity and this reduction increases as more automata are added. Testing a relatively simple property like state reachability is linear in the size of the state space, and there are most likely no polynomial time methods for testing more complicated μ-calculus formulas. Therefore, the reductions is state space size with the $\vec{G}$ are potentially very important for "large scale" systems.

7.9 Discussion

This chapter introduced a class of isomorphic module systems and a state permutation symmetry definition for these distributed systems. A permutation symmetric μ-calculus was introduced that was tailored for these permutation symmetric systems. It was shown how a predefined quotient structure can be used to more efficiently test if permutation symmetric μ-calculus propositions hold at states in permutation symmetric systems. The next chapter builds on this work by discussing a special class of permutation symmetric systems where the behavior of each module is partitioned to be either global or private in behavior.

CHAPTER VIII

ISOMORPHIC DISTRIBUTED DISCRETE-EVENT SYSTEMS

8.1 Chapter Overview

This chapter explores the properties of a special class of distributed systems where each module's behavior is represented as a language over a set of events partitioned into private and global events. A private event represents behavior that forces a state update only in the module where it occurs. In contrast, all modules must coordinate their respective behaviors on the occurrence of global events. This model is a special case of the permutation symmetric systems in Chapter VII and is presented in the standard supervisory control framework discussed in Chapter III. This class of distributed systems is referred to as *similar module systems* hereafter. A time efficient decomposition operation is given for constructing the subsystems of a composed similar module system that is a large improvement on current known methods.

Control properties of these distributed systems are also discussed. It is assumed that when the modules are controlled, they are controlled locally with exactly one controller per module such that the controllers make local observations of the behavior of a module and enforce local control actions. Also, the controllers enforce the

same control policy at each of their respective subsystems. There is no communication between controllers besides the implicit communication due to the occurrence of observable global events. Necessary and sufficient conditions for achieving local and global specifications in this setting are identified. Methods are given for verifying the local and global behavior of these systems with respect to regular language specifications.

8.2 Similar Module Systems

This chapter discusses the properties of similar module systems $\{G_1, \ldots, G_n\}$, using the standard supervisory control model $G_i = (X, x_0, \Sigma_i, \delta_i, X_m)$ outlined in Chapter III. These systems are a special case of the permutation symmetric systems of Chapter VII where state markings are the only proposition labellings allowed on the states and the module event sets Σ_i are partitioned into the distinct subsets Σ_g and Σ_{pi}, the (unique) global event set and the private event set for module i, respectively. The private event sets $\Sigma_{p1}, \ldots, \Sigma_{pn}$ are copies of one another for $G_1, \ldots, G_n$, respectively such that for all $i, j, i \neq j$, $\Sigma_{pi} \cap \Sigma_{pj} = \emptyset$. It is assumed that $\Sigma_g \cap \Sigma_{pi} = \emptyset$ and Σ denotes $\Sigma_1 \cup \cdots \cup \Sigma_n$. For the set of modules $\{G_1, \ldots, G_n\}$ an automaton $G^{\|} = G_1 \| \cdots \| G_n$ can be constructed as with the systems in Chapter VII such that $G^{\|} = (X^{\|}, x_0^{\|}, \Sigma, \delta^{\|}, X_m^{\|})$ and the only atomic proposition labelling on the states is a binary state marking as specified by $X_m^{\|}$.

For any $\sigma_i \in \Sigma_{pi}$ there are corresponding events $\Psi_{i1}(\sigma_i) \in \Sigma_{p1}, \ldots, \Psi_{in}(\sigma_i) \in \Sigma_{pn}$ in the other private event sets. The one-to-one mapping $\Psi_{ij}(\cdot)$ is extended for notational simplicity in this setting to $\Psi_{ij} : \Sigma \to \Sigma$ so that the private event set of module i is mapped to the jth private event set, the private event set of module j is mapped to the ith private event set and all other events (namely, global events) are

mapped to themselves. The function $\Psi_{ij}(\cdot)$ is extended in the usual manner to map strings of events and languages. Note that $\Psi_{ii}(\cdot)$ is the identity function and $\Psi_{ii}(\cdot) = \Psi_{ij}(\Psi_{ij}(\cdot))$. As an example of the operation of the $\Psi_{ij}(\cdot)$ function, suppose there are system event sets $\Sigma_i = \{a_i, b_i, d, g\}$ and $\Sigma_j = \{a_j, b_j, d, g\}$. For these alphabets $\Sigma_g = \{d, g\}$, $\Sigma_{p1} = \{a_i, b_i\}$ and $\Sigma_{p2} = \{a_j, b_j\}$. Then, $\Psi_{ij}(dga_ia_jb_jgb_i) = dga_ja_ib_igb_j$. Example 20 shows a simple example of the systems discussed in this chapter.

Example 20 *Consider the 2 module similar module system composed G_1 and G_2 seen in Figure 8.1. The relevant event sets are: $\Sigma_g = \{\gamma, \lambda\}$, $\Sigma_{p1} = \{\alpha_1, \beta_1\}$ and $\Sigma_{p2} = \{\alpha_2, \beta_2\}$. G_1 and G_2 are both locally non-blocking and deadlock free.*

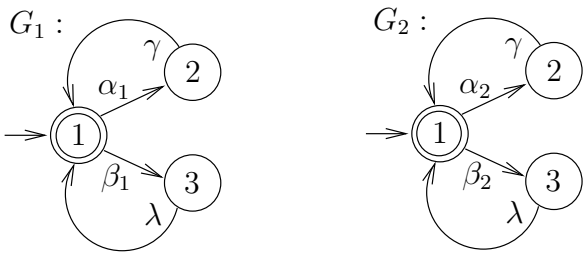

Figure 8.1: The automata G_1 and G_2.

$P_i : \Sigma^* \to \Sigma_i^*$ is the natural projection such that $P_i(s)$ is the string s with events in $\Sigma \setminus \Sigma_i$ erased. Similarly, let $P_{pi} : \Sigma^* \to \Sigma_{pi}^*$ be the natural projection that erases events in $\Sigma \setminus \Sigma_{pi}$ and let $P_g : \Sigma^* \to \Sigma_g^*$ be the natural projection that erases events in $\Sigma \setminus \Sigma_g$. All modules $G_i \in \{G_1, \ldots, G_n\}$ have the same transition structure except that transitions labelled with private events are relabelled according to the $\Psi_{ij}(\cdot)$ function. To formalize, let $G_i = (X, x_0, \Sigma_i, \delta_i, X_m)$. For $x \in X, \gamma \in \Sigma_i$, $\delta_i(x, \gamma) = \delta_j(x, \Psi_{ij}(\gamma))$ if it is defined.

The automata $\{G_1, \ldots, G_n\}$ model the private behaviors of the modules of the

similar module systems, but $G_1 \| \cdots \| G_n$ is used to model the global networked behavior of the systems where the private behaviors of the modules $\{G_1, \ldots, G_n\}$ are coordinated on the occurrence of common global events in Σ_g, according to the parallel composition operation. Example 21 shows the automaton $G_1 \| G_2$ constructed from the automata in Example 20.

Example 21 *Consider the composed system $G_1 \| G_2$ as seen in Figure 8.2.*

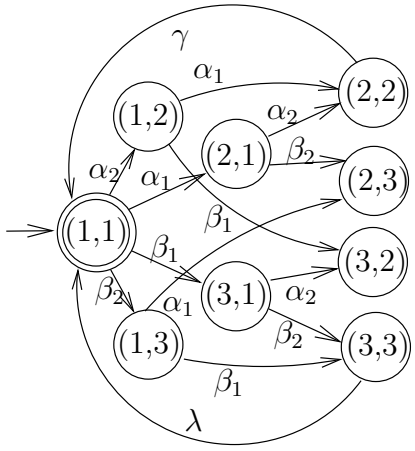

Figure 8.2: The automaton $G_1 \| G_2$.

The automaton $G_1 \| G_2$ has a large degree of symmetry due to the similarity of the component automata G_1 and G_2. In fact, the modular system has permutation symmetry. Consider the states $(1,2)$ and $(2,1)$. These states are reached by the occurrence of α_2 and α_1 events respectively. Consider also the $(2,3)$ and $(3,2)$ states. If the subscript labels of the events in the strings leading to state $(2,3)$ are swapped then the resulting string will lead to state $(3,2)$. Any property of state $(2,3)$ is also held by state $(3,2)$ because the swapping is being done on the order of parallel composition of the component automata when state locations are swapped. This operation is valid because the parallel composition operation is commutative and therefore the

order of parallel composition is arbitrary. The various similar modules are identical with respect to a renaming of private events.

For $G_1 \| G_2$ there are six classes of states that could be considered equivalent with respect to a reordering of the component states. The six sets of equivalent state classes are $\{(1,1)\}$,$\{(1,2),(2,1)\}$,$\{(1,3),(3,1)\}$,$\{(2,2)\}$, $\{(2,3),(3,2)\}$ *and* $\{(3,3)\}$.

These similar module systems are a special case of the permutation symmetric systems in Chapter VII.

Theorem 29 *If a similar module system $\{G_1, \ldots, G_n\}$ is given as outlined above, then $G^{\|}$ is permutation symmetric with respect to the following transposition translation operator.*

$$\pi_{ij}(\sigma) = \begin{cases} \Psi_{ij}(\sigma) & \text{if} \quad \sigma \in \Sigma_{pi} \\ \Psi_{ji}(\sigma) & \text{if} \quad \sigma \in \Sigma_{pj} \\ \sigma & \text{if} \quad \text{otherwise} \end{cases} \tag{8.1}$$

Proof: This proof uses notation from Chapter VII. Suppose there are two states $x_a^{\|}$ and $x_b^{\|}$ in $G^{\|}$ such that $\delta^{\|}(x_a^{\|}, \sigma) = x_b^{\|}$. It is sufficient to demonstrate that for any $i, j \in \{1, \ldots, n\}$, $\delta^{\|}(\phi_{ij}(x_a^{\|}), \pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})$. This is shown in four parts.

Case 1 *: $\sigma \in \Sigma_g$.*

In this case, $\forall k \in \{1, \ldots, n\} \delta^k(x_a^{\|k}, \sigma) = x_b^{\|k}$. Therefore, $\delta^{\|}(\phi_{ij}(x_a^{\|}), \sigma) = \phi_{ij}(x_b^{\|})$ and consequently $\delta^{\|}(\phi_{ij}(x_a^{\|}), \pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})$.

Case 2 *: $\sigma \in (\cup_{k=1}^{n} \Sigma_{pi}) \setminus (\Sigma_{pi} \cup \Sigma_{pj})$.*

Therefore, $\exists k \in \{1, \ldots, n\}, k \neq i, j$ such that $\forall l \neq k \; x_a^{\|l} = x_b^{\|l}$ and $\delta^k(x_a^{\|k}, \sigma) = x_b^{\|k}$. This implies that only the kth module state of $x_a^{\|}$ is updated on the occurrence of

σ and there is no effect on the ith and jth module states. Hence, $\delta^{\|}(\phi_{ij}(x_a^{\|}), \sigma) = \phi_{ij}(x_b^{\|})$ and consequently $\delta^{\|}(\phi_{ij}(x_a^{\|}), \pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})$.

Case 3 *:* $\sigma \in \Sigma_{pi}$.

Therefore, only the ith module state of $x_a^{\|}$ is updated on the occurrence of σ and there is no effect on other module states. Hence, $\delta^{\|}(\phi_{ij}(x_a^{\|}), \Psi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})$ and consequently $\delta^{\|}(\phi_{ij}(x_a^{\|}), \pi_{ij}(\sigma)) = \phi_{ij}(x_b^{\|})$.

Case 4 *:* $\sigma \in \Sigma_{pj}$.

Same as Case 3 with the i and j indices swapped. ■

However, not every permutation symmetric system can be considered a similar module system.

Example 22 *Consider the system from Example 17 in Chapter VII. Note that none of the system events in* $\{G_1, G_2, G_3\}$ *could possibly belong to a set of global events* Σ_g *because no transitions in* $\{G_1, G_2, G_3\}$ *are labelled by exactly the same events in all of these systems. Also, no events in* $\{G_1, G_2, G_3\}$ *could belong to a set of private events* Σ_{pi} *because there are no events that occur in exactly one module.*

It is now shown that $P_i\left(\mathcal{L}_m(G_1\| \cdots \|G_n)\right)$, the local behavior of the interacting modules, is language equivalent to the local behavior when the modules operate in isolation (i.e., $\mathcal{L}_m(G_i)$).

Theorem 30 *For a similar module system* $\{G_1, \ldots, G_n\}$ *as introduced above with respective local projection operations* $\{P_1, \ldots, P_n\}$ *and for* $i \in \{1, \ldots, n\}$,

$$P_i\left(\mathcal{L}_m(G_1\| \cdots \|G_n)\right) = \mathcal{L}_m(G_i).$$

Proof: It is known that $\mathcal{L}_m(G_1 \| \cdots \| G_n) = \left[P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))\right]$, so it is sufficient to show that $P_i\left[P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))\right] = \mathcal{L}_m(G_i)$.

This proof is composed of two parts.

It is known that $P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n)) \subseteq P_i^{-1}(\mathcal{L}_m(G_i))$.

$\Rightarrow P_i\left[P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))\right]$

$\subseteq P_i\left[P_i^{-1}(\mathcal{L}_m(G_i))\right]$.

$P_i\left[P_i^{-1}(\mathcal{L}_m(G_i))\right] = \mathcal{L}_m(G_i))$, so

$P_i\left[P_1^{-1}(\mathcal{L}_m(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(G_n))\right] \subseteq \mathcal{L}_m(G_i)$.

The other direction is proved by showing

$(t_i \in \mathcal{L}_m(G_i)) \Rightarrow \left(t_i \in P_i\left[\cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j))\right]\right)$.

Let $t_i \in \mathcal{L}_m(G_i)$. This implies that $t_i \in P_i(P_i^{-1}(\mathcal{L}_m(G_i)))$. Let us now construct a string $t_{\{1,\ldots,n\}}$ from the string t_i. Let $t_{\{1,\ldots,n\}}$ be a copy of t_i except replace all private events $\sigma_{pi} \in \Sigma_{pi}$ with the string $\sigma_{p1}\sigma_{p2}, \ldots, \sigma_{pn}$. Due to the construction of $\{G_1, \ldots, G_n\}$, $t_{\{1,\ldots,n\}} \in P_j^{-1}(\mathcal{L}_m(G_j))$ for all $j \in \{1, \ldots, n\}$. Thus,

$\forall j \in \{1, \ldots, n\} t_{\{1,\ldots,n\}} \in P_j^{-1}(\mathcal{L}_m(G_j))$

$\Rightarrow t_{\{1,\ldots,n\}} \in \cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j))$.

$\Rightarrow t_i \in P_i\left[\cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j))\right]$.

Therefore, $\mathcal{L}_m(G_i) \subseteq P_i\left[\cap_{j=1}^n P_j^{-1}(\mathcal{L}_m(G_j))\right]$ which implies

$$\mathcal{L}_m(G_i) \subseteq P_i\left(\mathcal{L}_m(G_1 \| \cdots \| G_n)\right).$$

■

It is a simple matter to extend the result in Theorem 30 to the case of languages generated instead of languages marked, i.e.,

$$P_i\left[P_1^{-1}(\mathcal{L}(G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}(G_n))\right] = \mathcal{L}(G_i).$$

The following is also a simple corollary to Theorem 30.

Corollary 11 *Suppose a similar module system* $\{G_1, \ldots, G_n\}$ *as introduced above with respective local projections* $\{P_1, \ldots, P_n\}$ *and local languages* $\{K_1, \ldots, K_n\}$ *are given. Then, for all* $i \in \{1, \ldots, n\}$,

$$(K_i = P_i\left(\mathcal{L}_m(G_1 \| \cdots \| G_n)\right)) \iff (K_i = \mathcal{L}_m(G_i))\,.$$

8.3 Decomposition and Similar Module Systems

This section shows some of the fundamental properties of similar module systems. Some preliminary definitions and lemmas are first shown and then used to demonstrate the properties of a decomposition operation of similar module systems that runs in polynomial time. This decomposition operation is one of the main contributions of this chapter as the are few time-efficient methods for performing decomposition operations in discrete-event systems theory.

Definition 18 *[68] A language* K *is* decomposable *with respect to the projections* $\{P_1, \ldots, P_n\}$ *if* $K = P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))$.

A language is decomposable if given the local knowledge of K at all sites, i.e., $P_1(K), \ldots, P_n(K)$, the language K can be recovered exactly. Note that this definition is slightly altered from the definition given in [68] in that external system behavior is disregarded. It has been assumed that $K \subseteq \Sigma^*$ and the set $\{P_1, \ldots, P_n\}$ has been used for projection operations.

Definition 19 *[81] A set of languages* $\{L_1, \ldots, L_n\}$ *is said to be* non-conflicting *if* $\overline{L_1 \cap \cdots \cap L_n} = \overline{L_1} \cap \cdots \cap \overline{L_n}$.

It is known that the parallel composition of a set of non-blocking automata need not be non-blocking unless the respective languages marked by the automata are non-conflicting.

Definition 20 *A language $K \subseteq \Sigma^*$ is said to be* symmetric *with respect to the private event sets $\Sigma_{p1}, \ldots, \Sigma_{pn}$ and the corresponding mappings $\Psi_{11}, \ldots, \Psi_{1n}$ if $\forall i \in \{1, \ldots, n\}$ $\Psi_{1i}(K) = K$.*

This symmetry definition is used to convey the intuition that K is identical with respect to a relabelling of private events for the similar module system $\{G_1, .., G_n\}$.

This prompts the following lemmas whose proofs are straightforward and therefore omitted.

Lemma 12 *Given set $\Sigma_i \subseteq \Sigma$, a language $K \subseteq \Sigma^*$ and natural projection operation $P_i : \Sigma^* \to \Sigma_i^*$ with corresponding inverse projection $P_i^{-1} : \Sigma_i^* \to \Sigma^*$,*

1. *$P_i(\overline{K}) = \overline{P_i(K)}$.*
2. *$P_i^{-1}(\overline{K}) = \overline{P_i^{-1}(K)}$.*

Lemma 13 *Given a language $K \subseteq \Sigma^*$, the previously defined projection operations $P_1(\cdot), \ldots, P_n(\cdot)$ and the mappings $\Psi_{11}(\cdot), \ldots, \Psi_{1n}(\cdot)$,*

$$\forall i \in \{1, \ldots, n\} \left(K = \Psi_{1i}(K)\right) \Rightarrow \left(\Psi_{1i}(P_1(K)) = P_i(K)\right).$$

It can also be shown that if a language K is decomposable, then $\overline{K}$ is also decomposable

Lemma 14 *Given a language K and a set of projection operators $\{P_1, \ldots, P_n\}$, if K is decomposable with respect to $\{P_1, \ldots, P_n\}$ and $\{P_1^{-1}(P_1(K)), \ldots, P_n^{-1}(P_n(K))\}$ are non-conflicting, then $\overline{K}$ is decomposable with respect to $\{P_1, \ldots, P_n\}$.*

Proof: Using Lemma 12, it is known that

$$\forall i \in \{1, \ldots, n\}, P_i^{-1}(P_i(\overline{K})) = \overline{P_i^{-1}(P_i(K))}.$$

Consequently,

$$\cap_{i=1}^{n} \overline{P_i^{-1}(P_i(K))} = \cap_{i=1}^{n} P_i^{-1}(P_i(\overline{K})).$$

Due to the assumption that the languages $\{P_1^{-1}(P_1(K)), \ldots, P_n^{-1}(P_n(K))\}$ are non-conflicting,

$$\overline{\cap_{i=1}^{n} P_i^{-1}(P_i(K))} = \cap_{i=1}^{n} P_i^{-1}(P_i(\overline{K})).$$

Therefore, due to the decomposability of K,

$$K = \cap_{i=1}^{n} P_i^{-1}(P_i(K))$$

and

$$\overline{K} = \cap_{i=1}^{n} P_i^{-1}(P_i(\overline{K})).$$

■

The next result relates the $\Psi_{ij}(\cdot)$ operator with the prefix-closure of languages.

Lemma 15 *Given a language K and the $\Psi_{ij}(\cdot)$ operator introduced above,*

$$\Psi_{ij}(\overline{K}) = \overline{\Psi_{ij}(K)}.$$

Proof: Suppose $s \in \overline{\Psi_{ij}(K)}$. Then there exists a string t such that $st \in \Psi_{ij}(K)$.

$\Rightarrow \Psi_{ij}(st) \in K$

$\Rightarrow \Psi_{ij}(s)\Psi_{ij}(t) \in K$

$\Rightarrow \Psi_{ij}(s) \in \overline{K}$

$\Rightarrow s \in \Psi_{ij}(\overline{K})$

The reverse direction of this proof follows from performing the steps above in the opposite order. ■

Lemma 15 can be used to show the following theorem about the symmetry of prefix-closed languages.

Theorem 31 *Let K be symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$. Then, $\overline{K}$ is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$.*

Proof: Because K is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$, then for all i, j, $\Psi_{ij}(K) = K$. Hence, by Lemma 15, $\forall i, j \in \{1, \ldots, n\} \Psi_{ij}(\overline{K}) = \overline{K}$ and therefore $\overline{K}$ is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$. ■

It is now demonstrated that symmetric and decomposable languages can be modelled using the similar module system framework.

Theorem 32 *Suppose a trim automaton $H = (X, x_0, \Sigma, \delta, X_m)$ is given such that $\mathcal{L}_m(H)$ is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$ and decomposable with respect to $\{P_1, \ldots, P_n\}$. Then there exists a set of similar module systems $\{H_1, \ldots, H_n\}$ such that $\mathcal{L}_m(H_1 \| \cdots \| H_n) = \mathcal{L}_m(H)$ and $\forall i \in \{1, \ldots, n\}$ $P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i)$.*

Proof: Using standard methods, given Σ_i and H, a automaton H_i can be constructed such that $\mathcal{L}_m(H_i) = P_i(\mathcal{L}_m(H))$ for all i such that $i \in \{1, \ldots, n\}$. Because $\mathcal{L}_m(H)$ is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$,

$\forall i, j \in \{1, \ldots, n\}$ $\Psi_{ij}(\mathcal{L}_m(H)) = \mathcal{L}_m(H)$

$\Rightarrow \Psi_{ij}(P_i(\mathcal{L}_m(H))) = P_j(\mathcal{L}_m(H))$.

$\Rightarrow \Psi_{ij}(\mathcal{L}_m(H_i)) = \mathcal{L}_m(H_j)$.

Therefore, the set of automata $\{H_1, \ldots, H_n\}$ can be constructed such that they are isomorphic to one another with respect to a renaming of their private events.

Because $\mathcal{L}_m(H)$ is decomposable with respect to $\{P_1, \ldots, P_n\}$,

$P_1^{-1}(P_1(\mathcal{L}_m(H))) \cap \cdots \cap P_n^{-1}(P_n(\mathcal{L}_m(H))) = \mathcal{L}_m(H)$

$\Rightarrow P_1^{-1}(\mathcal{L}_m(H_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(H_n)) = \mathcal{L}_m(H)$

$\Rightarrow \mathcal{L}_m(H_1 \| \cdots \| H_n) = \mathcal{L}_m(H)$ due to properties of the parallel composition operation.

$\Rightarrow \forall i \in \{1, \ldots, n\} P_i(\mathcal{L}_m(H_1 \| \cdots \| H_n)) = P_i(\mathcal{L}_m(H))$

$\Rightarrow \forall i \in \{1, \ldots, n\} \mathcal{L}_m(H_i) = P_i(\mathcal{L}_m(H))$.

This last step is by the result of Theorem 30 shown above. ■

In the proof of Theorem 32, any method can be used to construct the similar module systems $\{H_1, \ldots, H_n\}$ from H such that $\mathcal{L}_m(H_1 \| \cdots \| H_n) = \mathcal{L}_m(H)$. The standard method consists of converting all transitions labelled by events in $\Sigma \setminus \Sigma_1$ in H to non-deterministic ϵ-transitions. Then, this non-deterministic automaton is converted into a deterministic one. Unfortunately, the determinization algorithm takes exponential time and space exponential in the size of H in the worst case.

There has been little discussion in the discrete-event systems literature about ways of performing modular decompositions besides ad-hoc methods developed on a case-by-case basis. Developing formal methods for performing this operation is very important for many real-world problems where a large complicated system model would need to be converted into simpler, modular model blocks. An efficient method has been developed for performing this decomposition on similar module systems as discussed below.

Before this method is presented an intuitive introduction to the algorithm's operation is given. Let $\{H_1, \ldots, H_n\}$ be the desired modules that compose the given symmetric and decomposable H, i.e., $H = H_1 \| \cdots \| H_n$. Suppose distributed simulations of the behavior of $\{H_1, \ldots, H_n\}$ are run such that the only way for private events to occur would be in strings such as $\sigma_{p1}\sigma_{p2} \cdots \sigma_{pn}$ with no interleavings of other events and the events $\{\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}\}$ are private events identical to one another with respect to $\Psi_{ij}(\cdot)$ mappings. On the occurrence of these $\sigma_{p1}\sigma_{p2} \cdots \sigma_{pn}$ strings the local states $\{H_1, \ldots, H_n\}$ are forced to update with identical state transitions. Because $H = H_1 \| \ldots \| H_n$, this lockstep simulation of private behaviors in

the global system model can be used to calculate the local system behaviors. Given H, this automaton can be trimmed by only allowing global events and strings of private events $\{\sigma_{p1}, \sigma_{p2}, \ldots, \sigma_{pn}\}$ as outlined above to occur. The behavior of this specially trimmed automaton matches the behavior of the lockstep simulation of the $\{H_1, \ldots, H_n\}$ automata. Then, if the event σ_{p1} is substituted for every occurrence of a string $\sigma_{p1}\sigma_{p2}\cdots\sigma_{pn}$, the behavior of the resulting automaton will match the behavior of the H_1 module.

In the following algorithm, a slightly modified set notation for the state transition function is used, i.e., if $\delta(x, s) = y$ for state transition function $\delta(\cdot, \cdot)$, states x, y and string s, then the notation that $(x, s, y) \in \delta$ is used. For simplicity it is assumed that the imputed automaton H is deterministic and trim.

Algorithm 11 *Decomposition Construction Algorithm.*

Input:

$H = \left(X^H, x_0^H, \Sigma, \delta^H, X_m^H\right)$

$\Sigma_1, \ldots, \Sigma_n$

(Note that $\delta^H \subseteq (X^H \times \Sigma \times X^H)$*).*

Output:

$H_1 = (X^1, x_0^1, \Sigma_1, \delta^1, X_m^1)$.

(Note that $\delta^1 \subseteq (X^1 \times \Sigma_1 \times X^1)$*).*

Assumptions:

H is trim.

$\mathcal{L}_m(H)$ is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$ and decomposable with respect to $\{P_1, \ldots, P_n\}$.

S is a stack with the normal push and pop operations.

Initialize:

$X^1 := \{x_0^H\}$;

$x_0^1 := x_0^H$;

$\delta^1 := \emptyset$;

$S := [x_0^1]$;

Repeat:

{

$x_s = pop(S)$

Do the following for all $x_b \in X^H$*:*

Do the following for all $\sigma_g \in \Sigma_g$ *with* $(x_s, \sigma_g, x_b) \in \delta^H$*:*

{

$\delta^1 := \delta^1 \cup \{(x_s, \sigma_g, x_b)\}$;

If $x_b \notin X^1$

Then

{

$X^1 := X^1 \cup \{x_b\}$;

$push(S, x_b)$;

}

}

Do the following for all $\sigma_{p1} \in \Sigma_{p1}, \ldots, \sigma_{pn} \in \Sigma_{pn}$

such that $\sigma_{pi} = \Psi_{1i}(\sigma_{p1})$, $(x_s, \sigma_{p1}\sigma_{p2}\cdots\sigma_{pn}, x_b) \in \delta^H$*:*

{

$\delta^1 := \delta^1 \cup \{(x_s, \sigma_{p1}, x_b)\}$;

If $x_b \notin X^1$

Then

{

$X^1 := X^1 \cup \{x_b\}$;

$push(S, x_b)$;

}

}

}

Until S is empty;

$X_m^1 = X_m^H \cap X^1$;

Construct $\{H_2, \ldots, H_n\}$ from H_1 using $\Psi_{1i}(\cdot)$ operations;

Return $\{H_1, \ldots, H_n\}$.

The correctness of the decomposition algorithm is now demonstrated.

Theorem 33 *Suppose there is a trim automaton H such that $\mathcal{L}_m(H)$ is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$ and decomposable with respect to $\{P_1, \ldots, P_n\}$. Then, if Algorithm 11 is run with H as input, a set of automata $\{H_1, \ldots, H_n\}$ is returned such that $\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \| \cdots \| H_n)$ and $\forall i \in \{1, \ldots, n\}$ $P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i)$.*

Proof: It is known from Theorem 32 that there exists similar modules $\{H_1', \ldots, H_n'\}$ such that $\mathcal{L}_m(H) = \mathcal{L}_m(H_1' \| \cdots \| H_n')$. From H, a specially trimmed version of H called H^{lock} can be constructed by trimming all private event transitions in H except those that correspond to occurrence of chains of private events

$$\sigma_{p1} \Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1})$$

for $\sigma_{p1} \in \Sigma_{p1}$ and no other events are allowed to be interleaved in these strings.

Therefore,

$$\mathcal{L}_m(H^{lock}) = \mathcal{L}_m(H) \cap \left(\{\sigma_{p1} \Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1}) | \sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g\right)^*.$$

Now consider the composition of the $\{H'_1, \ldots, H'_n\}$ automata where global events are allowed to occur freely and private events are restricted to occur in strings $\sigma_{p1}\Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1})$ for $\sigma_{p1} \in \Sigma_{p1}$ with no other events interleaved as above. This automaton marks the same language as H^{lock} shown above because $\mathcal{L}_m(H) = \mathcal{L}_m(H'_1 \| \cdots \| H'_n)$. Furthermore, due the construction of H^{lock}, it should be apparent that

$$P_i\left(\mathcal{L}_m(H^{lock})\right) = \mathcal{L}_m(H'_i)$$

because the marking behavior of H_i can be can be reclaimed from H^{lock} by replacing the $\sigma_{p1}\Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1})$ transitions with a single σ_{p1} transition. Therefore,

$$P_i\left[\mathcal{L}_m(H) \cap (\{\sigma_{p1}\Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1}) | \sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g)^*\right] = \mathcal{L}_m(H'_i).$$

Algorithm 11 constructs H_1 by restricting the behavior of private events in H to strings such as $\sigma_{p1}\Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1})$ and then converts these transition strings to σ_{p1} events. Therefore,

$$\mathcal{L}_m(H_1) = P_i\left[\mathcal{L}_m(H) \cap (\{\sigma_{p1}\Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1}) | \sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g)^*\right].$$

Consequently, $\mathcal{L}_m(H'_1) = \mathcal{L}_m(H_1)$. For all $i \in \{1, \ldots, n\}$, the automata H'_i and H_i are copies of the H'_1 and H_1 automata with respect to a renaming of transition labels by the $\Psi_{1i}(\cdot)$ functions. This implies that

$$\bigcap_{i \in \{1,\ldots,n\}} \mathcal{L}_m(H'_i) = \bigcap_{i \in \{1,\ldots,n\}} \mathcal{L}_m(H_i)$$

Therefore,

$$\mathcal{L}_m(H'_1 \| \cdots \| H'_n) = \mathcal{L}_m(H_1 \| \cdots \| H_n)$$

Hence, $\mathcal{L}_m(H) = \mathcal{L}_m(H_1 \| \cdots \| H_n)$.

By construction $\{H_1, \ldots, H_n\}$ are similar module systems. By Theorem 30, $P_i\left(\mathcal{L}_m(H_1 \| \cdots \| H_n)\right) = \mathcal{L}_m(H_i)$. Therefore, by substitution,

$$\forall i \in \{1, \ldots, n\}\ P_i(\mathcal{L}_m(H)) = \mathcal{L}_m(H_i).$$

■

The main $Repeat - Until$ loop in Algorithm 11 iterates $|X^1|$ times, which is the size of the state space of H_1. The size of the state space of H_1 is always at least as small as $|X^H|$, the size of the state space of H, and generally much more so. Inside the main iterative loop, there are two types of tests for transition existence, one for individual global events and one for chains of private events. The tests for transition existence are generally efficient and negligible depending on the encoding of H, especially if each state contains a list of its outgoing transitions. In this case, only the existence of $|\Sigma_g|$ global event transitions and $|\Sigma_{p1}|$ private event string transitions at each state need to be tested. If there is a $\sigma_{p1} \in \Sigma_{p1}$ transition at the current state, it needs to be tested if there is a chain of transitions $\sigma_{p1}\sigma_{p2}\ldots\sigma_{pn}$ from the current state. Therefore, for every private event transition detected, at most $n-1$ other transitions need to be tested. Therefore, Algorithm 11 is in $O\left(|X^1| * (|\Sigma_g| + n * |\Sigma_{p1}|)\right)$.

An example of a run of Algorithm 11 is now given using a trimmed version of the automaton $G_1 \| G_2$ from Example 20.

Example 23 *Consider the automaton G that is the trimmed version of $G_1 \| G_2$ from Example 20 with the states renamed for convenience. This automaton can be seen in Figure 8.3.*

Note that $\mathcal{L}_m(G) = \left((\alpha_1\alpha_2 + \alpha_2\alpha_1)\,\gamma + (\beta_1\beta_2 + \beta_2\beta_1)\,\lambda\right)^*$. By inspection, this language is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$ and decomposable with respect to $\{P_1, \ldots, P_n\}$.

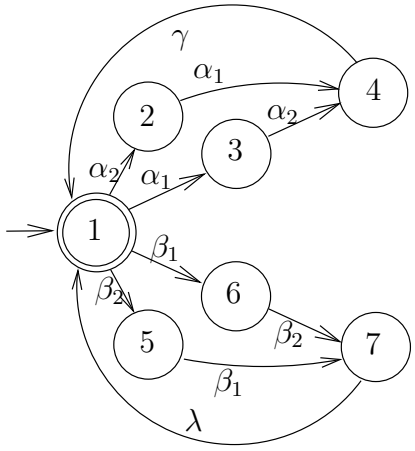

Figure 8.3: The automaton $G = Trim(G_1 \| G_2)$.

Algorithm 11 is now run with G *as input for* $\Sigma_1 = \{\alpha_1, \beta_1, \gamma, \lambda\}$ *and* $\Sigma_2 = \{\alpha_2, \beta_2, \gamma, \lambda\}$.

To initialize: $X^1 := \{1\}$, $x_0^1 := 1$, $\delta^1 := \emptyset$, $S := [1]$.

The first state is popped off of S. $x_s := 1$.

There are no Σ_g *transitions from state* 1, *but there are two private event chains,* $\alpha_1\alpha_2$ *and* $\beta_1\beta_2$ *starting from state* 1 *and respectively leading to states* 4 *and* 7. *Therefore, states* $4, 7$ *are added to* X^1, $(1, \alpha_1, 4)$ *and* $(1, \beta_1, 7)$ *are added to* δ^1 *and* 4 *and* 7 *are pushed onto* S.

Now, $X^1 = \{1, 4, 7\}$ $\delta^1 = \{(1, \alpha_1, 4), (1, \beta_1, 7)\}$, $S = [4, 7]$.

Next, 4 *is popped off* S *and* $x_s := 4$. *There is a* Σ_g *transition from* 4 $(4, \gamma, 1)$, *but no private event chains. Therefore,* $(4, \gamma, 1)$ *is added to* δ^1 *and no other changes are made.*

Next, 7 *is popped off* S *and* $x_s := 7$. *There is a* Σ_g *transition from* 7 $(7, \lambda, 1)$, *but no private event chains. Therefore,* $(7, \lambda, 1)$ *is added to* δ^1 *and no other changes are made. The stack* S *is empty, so the* $Repeat - Until$ *loop has been completed.*

The marked state list is now assigned $X_m^1 = \{1\}$ *and this completes the construction of* G_1. G_1 *is then copied as* G_2 *by replacing* α_1 *with* α_2 *and* β_1 *with* β_2.

This results in the automata seen in Figure 8.4.

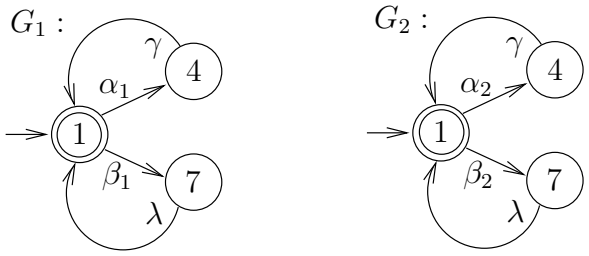

Figure 8.4: The automata G_1 and G_2 decomposed from G in Figure 8.3.

It is now shown that the $\{H_1, \ldots, H_n\}$ automata returned by Algorithm 11 are guaranteed to be trim.

Theorem 34 *Suppose a trim automaton* H *is given such that* $\mathcal{L}_m(H)$ *is symmetric with respect to* $\{\Sigma_1, \ldots, \Sigma_n\}$ *and decomposable with respect to* $\{P_1, \ldots, P_n\}$. *Let* $\{H_1, \ldots, H_n\}$ *be the automata constructed from* H *using Algorithm 11. Then, the automata* $\{H_1, \ldots, H_n\}$ *are all trim.*

Proof: It suffices to show that H_1 is trim.

It is known that all states X^1 in H_1 are reachable from the initial state, so to show that H_1 is trim, it must be demonstrated that for every unmarked state in H_1, there must be a string to a marked state.

Suppose $x \in X^1 \setminus X_m^1$. Because H is trim, there must be two strings $s, t \in \Sigma$ such that s is in

$$\left(\{\sigma_{p1}\Psi_{12}(\sigma_{p1}) \cdots \Psi_{1n}(\sigma_{p1}) | \sigma_{p1} \in \Sigma_{p1}\} \cup \Sigma_g\right)^*,$$

$\delta^H(x_0^H, s) = x$ and $\delta^H(x, t) \in X_m^H$. Define s_1, t_1 such that $s_1 = P_1(s)$ and $t_1 = P_1(t)$. Because of the construction in Algorithm 11, it must be true that $\delta^1(x_0^1, s_1) = x$. Furthermore, because $\delta^H(x, t) \in X_m^H$ and t_1 is the string of events that is relevant to H_1 when t occurs, then it must be true that $\delta^1(x^1, t_1) \in X_m^1$ because $\mathcal{L}_m(H_1 \| \cdots \| H_n) = \mathcal{L}_m(H)$. ■

Even though a language K may be symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$ and decomposable with respect to $\{P_1, \ldots, P_n\}$, it is possible that $\{P_1(K), \ldots, P_n(K)\}$ are conflicting. Consider the G automaton shown in Figure 8.3 from Example 23. It has already been shown how G is symmetric and decomposable. Now consider the G_1 and G_2 automata that are output from running the decomposition algorithm as in Example 23. The parallel composition of these automata is blocking, so $\{P_1(\mathcal{L}_m(G_1)), \ldots, P_n(\mathcal{L}_m(G_2))\}$ are conflicting, as seen in Figure 8.2 in Example 21.

Finally, note that even though a language K may be symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$ and $\{P_1(K), \ldots, P_n(K)\}$ are non-conflicting, it is possible that K is not decomposable with respect to $\{P_1, \ldots, P_n\}$. Consider the language $K = \{\epsilon, \alpha_1, \alpha_2\}$. K is symmetric with respect to $\{\{\alpha_1\}, \{\alpha_2\}\}$, and $\{\{\epsilon, \alpha_1\}, \{\epsilon, \alpha_2\}\}$ are non-conflicting. However, $K = \{\epsilon, \alpha_1, \alpha_2\}$ is not decomposable with respect to $\{\{\alpha_1\}, \{\alpha_2\}\}$.

8.4 Quotient Automata and Global Systems Properties

In Chapter VII a quotient automaton $\vec{G}$ is shown that can be used to verify properties of a permutation symmetric composed system $G^{\|}$. Because similar module systems are permutation symmetric the $\vec{G}$ construction could also be used to test properties of interacting similar module systems. However, due to the extra symmetric structure of these systems, a quotient automaton with an even smaller

state space can be used to test some system properties in $G^{\|}$. This section shows another quotient automaton, denoted by $\tilde{G}$, for testing important properties such as state reachability, non-blockingness and deadlockfreeness in similar module systems. The $\tilde{G}$ automaton uses the sets of states with the same sets of component states as the state equivalence class used in the quotient automaton construction. Therefore the number of various component states is disregarding when establishing composed state equivalences. A simple tutorial example is shown that demonstrates how the states of the modular automata used in the model can be partitioned into sets of "equivalent" states.

Example 24 *Reconsider the system first introduced in Example 20 with a third component G_3 similar to G_1 and G_2 in Figure 8.1. The automaton $G_1\|G_2\|G_3$ can be seen in Figure 8.5.*

State $(1,1,2)$ in $G_1\|G_2\|G_3$ is equivalent to $(2,1,1)$, $(1,2,2)$, $(2,1,2)$, etc... with respect to the sets of module states these composed states contain. By inspection, these states are also equivalent with respect to some important system properties such as deadlock, blocking and reachability. However, not all of these states are reachable from the initial state in the same number of transitions.

The intuition behind Example 24 is that state orderings and duplicated component states do not matter when testing several important global properties. A quotient automaton $\tilde{G}$ for testing these properties in similar module systems is now formally introduced.

A state $x^{\|}$ from the composed automata $G^{\|} = G_1\| \cdots \|G_n$ is an n-tuple of states in X. Let $\tilde{x}$ represent the set of module states that compose the n-tuple $x^{\|}$. For example, the set $\{1,2,6\}$ is the set of module states that compose the states

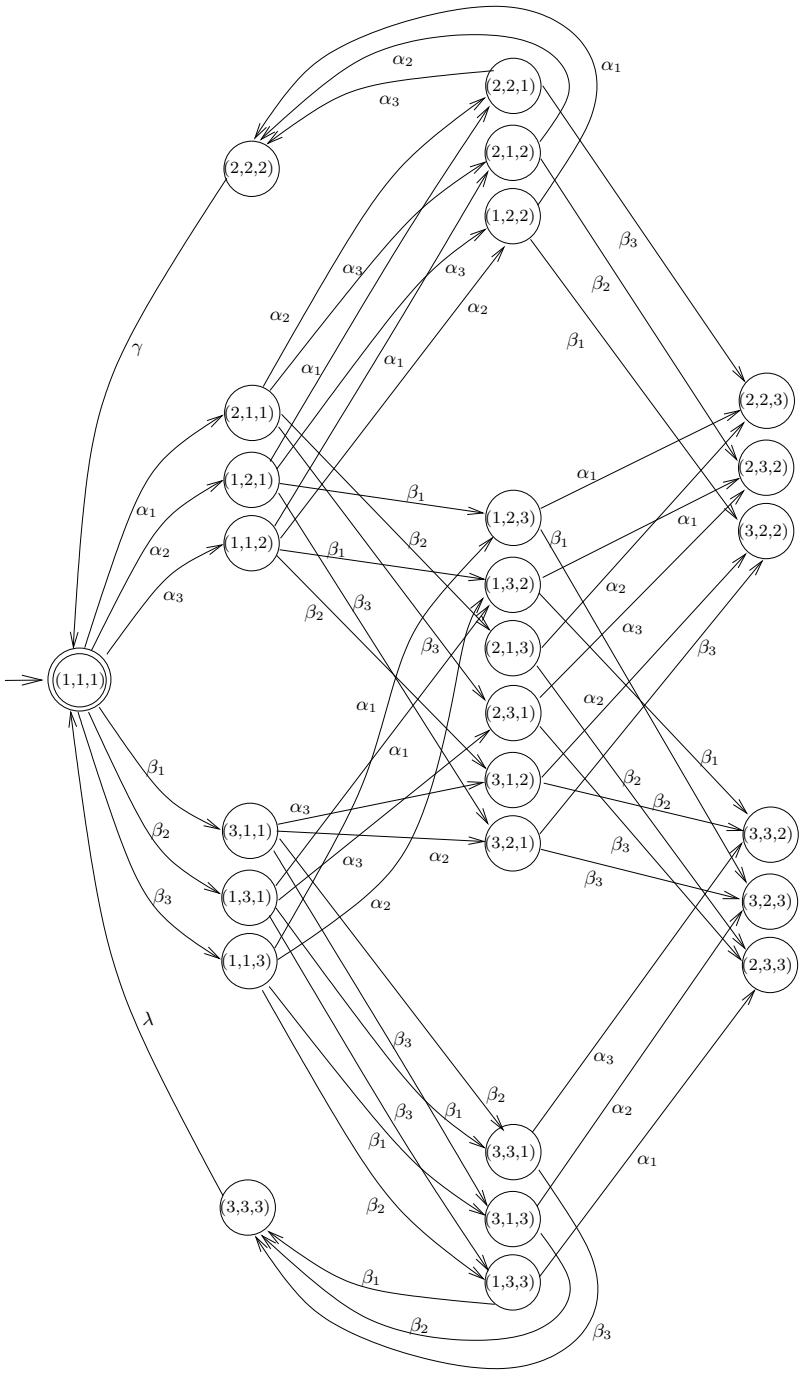

Figure 8.5: The automaton $G_1 \| G_2 \| G_3$.

$(1, 2, 6, 2)$ and $(1, 6, 6, 2)$ in $X^{\|}$. The function $Comp : X^{\|} \to 2^X$ is defined such that $Comp(x^{\|}) = \tilde{x}$ and the function $Comp^{-1}(\tilde{x})$ returns the set of states that has exactly

the states $\tilde{x}$ in its n-tuple. As an example of the operation of the $Comp(\cdot)$ function, $Comp((2,1,2)) = \{1,2\}$, $Comp((2,1,3)) = \{1,2,3\}$, $Comp((3,1,2,1,2)) = \{1,2,3\}$.

The set of state equivalence classes defined by the $Comp(\cdot)$ function is used as the state space of $\tilde{G}$. Let $\tilde{G} = (\tilde{X}, \{x_0\}, \Sigma_1, \tilde{\delta}, \tilde{X}_m)$ where $\tilde{X} = 2^X$, the power set of the component automaton G_i states, and let $\tilde{X}_m$, the set of marked states, be the states of $\tilde{G}$ such that at least one component state in the set is marked, i.e., $\tilde{X}_m = 2^X \setminus 2^{X \setminus X_m}$. The nondeterministic transition operator $\tilde{\delta}(\cdot,\cdot)$ is defined such that if there exists two states $x^{\|} \in Comp^{-1}(\tilde{x})$ and $y^{\|} \in Comp^{-1}(\tilde{y})$ and an event $\sigma_i \in \Sigma_i$ such that $\delta^{\|}(x^{\|}, \sigma_i) = y^{\|}$, then there is a corresponding transition in $\tilde{G}$ from $\tilde{x}$ to $\tilde{y}$ on the occurrence of $\Psi_{i1}(\sigma_i)$.

Note that even if $G^{\|}$ is a deterministic automaton, the corresponding $\tilde{G}$ automaton maybe nondeterministic; at a state $\tilde{x} \in \tilde{X}$ there may be several outgoing transitions labelled by an event σ. Therefore, similar to as was done with the $\vec{G}$ construction, the state transition function is defined such that $\tilde{\delta} : \tilde{X} \times \Sigma_1 \to 2^{\tilde{X}}$ and a state is in $\tilde{\delta}(\tilde{x}, \sigma) \subseteq \tilde{X}$ if there exists a σ labelled transitions from $\tilde{x}$ to that state in $\tilde{G}$.

$$\tilde{\delta}(\tilde{x}, \sigma) = \tag{8.2}$$
$$\left\{ \begin{array}{ll} \{\delta_1(x,\sigma) | x \in \tilde{x}\} & \text{if } (\sigma \in \Sigma_g) \wedge (\forall x \in \tilde{x}, \delta_1(x,\sigma)!) \\ \tilde{x} \cup \{\delta_1(x,\sigma)\} & \text{if } (\sigma \in \Sigma_{p1}) \wedge (|\tilde{x}| < n) \wedge (\delta_1(x,\sigma)!) \wedge (x \in \tilde{x}) \\ [\tilde{x} \setminus \{x\}] \cup \{\delta_1(x,\sigma)\} & \text{if } (\sigma \in \Sigma_{p1}) \wedge (|\tilde{x}| > 1) \wedge (\delta_1(x,\sigma)!) \wedge (x \in \tilde{x}) \end{array} \right\}$$

The intuition behind the definition of $\tilde{\delta}(\cdot,\cdot)$ is shown in three parts for the three cases in which this transition operator is defined.

The first case of the $\tilde{\delta}(\cdot,\cdot)$ definition corresponds to the occurrence of a global event such that all elements of $\tilde{x}$ are updated simultaneously on the occurrence of

global event $\sigma \in \Sigma_g$. All elements of $\tilde{x}$ need to be updated on the occurrence of σ because all states in $Comp^{-1}(\tilde{x})$ from $G^{\|}$ can be updated on the occurrence of σ according to the transition rules of $\delta^{\|}(\cdot,\cdot)$. Therefore $\delta_1(x,\sigma)$ is defined for all $x \in \tilde{x}$. This implies that $Comp\left(\delta^{\|}(x^{\|},\sigma)\right) \in \tilde{\delta}(\tilde{x},\sigma)$ and in this case $\{\delta_1(x,\sigma)|x \in \tilde{x}\} \in \tilde{\delta}(\tilde{x},\sigma)$ if $\sigma \in \Sigma_g$ and $\delta_1(x,\sigma)$ is defined for all $x \in \tilde{x}$.

The second case of the definition corresponds to the situation where there exists some state $x^{\|} \in Comp^{-1}(\tilde{x})$ such that there exists $i,j \in \{1,\ldots,n\}$ such that $i \neq j$ and $x^{\|^i} = x^{\|^j}$. In other words, $|\tilde{x}| < n$ as required for this case in the definition of $\tilde{\delta}(\cdot,\cdot)$. Therefore, at $x^{\|}$, if an event $\sigma_{pi} \in \Sigma_{pi}$ were to occur such that $\delta_i(x^{\|^i},\sigma_{pi})!$, then the resulting state $\delta(x^{\|},\sigma_{pi})$ exists and

$$Comp(\delta(x^{\|},\sigma_{pi})) = Comp(x^{\|}) \cup \left\{\delta_i(x^{\|^i},\sigma_{pi})\right\}.$$

Therefore,

$$Comp(x^{\|}) \cup \left\{\delta_i(x^{\|^i},\sigma_{pi})\right\} \in \tilde{\delta}(x^{\|},\Psi_{i1}(\sigma_{pi})),$$

and for this case, there is some $x \in \tilde{x}$, $\sigma \in \Sigma_{p1}$ such that $\tilde{x} \cup \{\delta_1(x,\sigma)\} \in \tilde{\delta}(\tilde{x},\sigma)$ as specified in the definition of $\tilde{\delta}(\cdot,\cdot)$.

The third case of the definition corresponds to the situation where there exists some state $x^{\|} \in Comp^{-1}(\tilde{x})$ such that there exists $i \in \{1,\ldots,n\}$ and for any $j \in \{1,\ldots,n\} \setminus \{i\}$, $x^{\|^i} \neq x^{\|^j}$. Thus, $|\tilde{x}| > 1$, as required for this case of the definition of $\tilde{\delta}(\cdot,\cdot)$. Therefore, at $x^{\|}$, if an event $\sigma_{pi} \in \Sigma_{pi}$ were to occur such that $\delta_i(x^{\|^i},\sigma_{pi})!$, then the resulting state $\delta(x^{\|},\sigma_{pi})$ would be in

$$\left[Comp(x^{\|}) \setminus \left\{x^{\|^i}\right\}\right] \cup \left\{\delta_i(x^{\|^i},\sigma_{pi})\right\} \in \tilde{\delta}(Comp(\tilde{x}),\Psi_{i1}(\sigma_{pi})),$$

and for this case there is some $x \in \tilde{x}$ and $\sigma_{pi} \in \Sigma_{pi}$ such that $[\tilde{x} \setminus x] \cup \{\delta_1(x,\sigma)\} \in \tilde{\delta}(\tilde{x},\sigma)$ as specified in the definition of $\tilde{\delta}(\cdot,\cdot)$.

The definition of $\tilde{\delta}(\cdot,\cdot)$ can be extended to allows strings of arbitrary length using the usual methods. The similar module system introduced in Example 24 is used to demonstrate the construction of a simple reduced state space composed automaton $\tilde{G}$.

Example 25 *Consider the similar module system G_1, G_2, G_3 discussed in Example 24 above. The set of state equivalence classes in $G_1 \| G_2 \| G_3$ with respect to the Comp$(\cdot)$ operation is $\{\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$. The quotient automaton $\tilde{G}$ constructed from G_1, G_2, G_3 can be seen in Figure 8.6.*

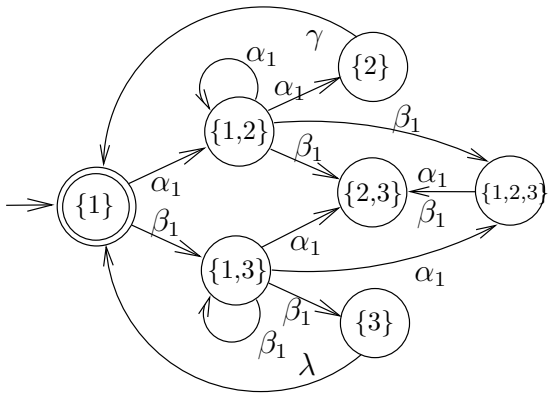

Figure 8.6: The automaton $\tilde{G}$ constructed from G_1, G_2, G_3.

The three different transition class from the definition of $\tilde{\delta}(\cdot,\cdot)$ are all exhibited in this example of $\tilde{G}$.

Consider the $\{2\} \overset{\gamma}{\mapsto}_{\tilde{G}} \{1\}$ transition in $\tilde{G}$. This transition is in the first case of the $\tilde{\delta}(\cdot,\cdot)$ definition and in the original $G_1 \| G_2 \| G_3$ automaton, it corresponds to the $(2,2,2) \overset{\gamma}{\mapsto}_{G_1 \| G_2 \| G_3} (1,1,1)$ transition. That is, a global event γ occurs and all states of $(2,2,2)$ are updated simultaneously.

Now consider the $\{1,2\} \overset{\alpha_1}{\mapsto}_{\tilde{G}} \{1,2\}$ transition in $\tilde{G}$. This transition corresponds to several transitions in the $G_1 \| G_2 \| G_3$ automaton, including $(1,1,2) \overset{\alpha_2}{\mapsto}_{G_1 \| G_2 \| G_3} (1,2,2)$.

For this transition, module state 1 *is replicated and the second module state of* $(1,2,2)$ *is updated on the occurrence of* α_2*, but the first module state remains* 1*. The* $\{1,2\}\overset{\alpha_1}{\mapsto}_{\tilde{G}}\{1,2\}$ *transition is in the second class of transitions in the definition of* $\tilde{\delta}(\cdot,\cdot)$*.*

The $\{1,2\}\overset{\alpha_1}{\mapsto}_{\tilde{G}}\{2\}$ *transition corresponds to the third class of transitions in the* $\tilde{\delta}(\cdot,\cdot)$ *definition. In the original* $G_1\|G_2\|G_3$ *automaton, this transition corresponds to the* $(2,2,1)\overset{\alpha_3}{\mapsto}_{G_1\|G_2\|G_3}(2,2,2)$ *transition among others. In this case, there is only one module in state* 1 *and on the occurrence of* α_3 *this module updates to state* 2 *and the other modules remain the same.*

Also notice that in $\tilde{G}$ *there is a transition* $\{1\}\overset{\alpha_1}{\mapsto}_{\tilde{G}}\{1,2\}$*. It is true that* $\{2,2,1\}\in Comp^{-1}(\{1,2\})$ *but in* $G_1\|G_2\|G_3$ *there is no event* σ *such that there is a transition* $(1,1,1)\overset{\sigma}{\mapsto}_{G_1\|G_2\|G_3}(2,2,1)$*. Therefore, state reachability in* $\tilde{G}$ *does not exactly imply state reachability in* $G_1\|G_2\|G_3$*. However, there is a string* $\alpha_1\alpha_2$ *such that* $(1,1,1)\overset{\alpha_1\alpha_2}{\mapsto}_{G_1\|G_2\|G_3}(2,2,1)$*. It is shown below that state reachability in* $\tilde{G}$ *is shown to imply a form of reachability in* $G_1\|\cdots\|G_n$ *and vice-versa.*

8.4.1 Verifying System Properties Using $\tilde{G}$

Two fundamental reachability results related to the $\tilde{G}$ and $G^{\|}$ constructions for similar module systems are now shown.

Lemma 16 *Suppose a similar module system* $\{G_1,\ldots,G_n\}$ *is given that is used to construct* $G^{\|}$ *and* $\tilde{G}$*. For two states of* $G^{\|}$*,* $x_1^{\|},x_2^{\|}\in X^{\|}$*, two states of* $\tilde{x}_1,\tilde{x}_2\in 2^X$ *and a string of transitions labelled by* $t^{\|}\in\Sigma^*$ *such that* $x_1^{\|}\overset{t^{\|}}{\mapsto}_{G^{\|}}=x_2^{\|}$*,* $Comp(x_1^{\|})=\tilde{x}_1$ *and* $Comp(x_2^{\|})=\tilde{x}_2$*, there exists a string of transitions labelled by* $\tilde{t}\in\Sigma_1^*$ *such that according to the transition rules of* $\tilde{G}$*,* $\tilde{x}_1\overset{\tilde{t}}{\mapsto}_{\tilde{G}}\tilde{x}_2$*.*

Proof: This lemma is shown using a proof by generalized induction on the length of $t^{\|}$.

For the base of induction, suppose $|t^{\|}| = 1$. This induction base is shown for two cases. In the first case, suppose that $t^{\|} = \sigma_g \in \Sigma_g$. Because all component states in $x_1^{\|}$ can update on the occurrence of σ_g, the resultant states are components in $x_2^{\|}$. Therefore, by the definition of the transition structure of $\tilde{G}$, $\tilde{x}_1 \overset{\sigma_g}{\mapsto}_{\tilde{G}} \tilde{x}_2$.

For the second case of the base of induction, suppose that $t^{\|} = \sigma_{pi} \in \Sigma_{pi}$. Therefore, when module G_i is in state ${x_1^{\|}}^i$, the module can update on the occurrence of σ_{pi} to state ${x_2^{\|}}^i$ and all other modules in $G^{\|}$ do not update, i.e., $\forall j \neq i,\ {x_1^{\|}}^j = {x_2^{\|}}^j$. So, when module G_i is in state ${x_1^{\|}}^i$, the module can update on the occurrence of σ_{pi} to state ${x_2^{\|}}^i$. Therefore, $\tilde{x}_1 \overset{\sigma_{pi}}{\mapsto}_{G^{\|}} \tilde{x}_2$. This completes the base of induction.

For the induction hypothesis, suppose that there is a string of transitions labelled by $t^{\|} \in \Sigma^n$ such that $x_1^{\|} \overset{t^{\|}}{\mapsto}_{G^{\|}} x_2^{\|}$, $Comp(x_1^{\|}) = \tilde{x}_1$ and $Comp(x_2^{\|}) = \tilde{x}_2$. Then, there exists a string of transitions labelled by $\tilde{t} \in \Sigma_1^*$ such that $\tilde{x}_1 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}_2$.

For the induction step, suppose there is a string of transitions labelled by $t^{\|} \in \Sigma^n$ and a transition labelled by $\sigma^{\|} \in \Sigma$ such that $x_1^{\|} \overset{t^{\|}\sigma^{\|}}{\mapsto}_{G^{\|}} = x_3^{\|}$, $Comp(x_1^{\|}) = \tilde{x}_1$ and $Comp(x_3^{\|}) = \tilde{x}_3$. Because $x_1^{\|} \overset{t^{\|}\sigma^{\|}}{\mapsto}_{G^{\|}} = x_3^{\|}$, there must be a state $x_2^{\|}$ such that $x_1^{\|} \overset{t^{\|}}{\mapsto}_{G^{\|}} x_2^{\|}$ and $x_2^{\|} \overset{\sigma^{\|}}{\mapsto}_{G^{\|}} x_3^{\|}$ where $Comp(x_2^{\|}) = \tilde{x}_2$. Because of the induction hypothesis there is a string of transitions labelled by $\tilde{t} \in \Sigma_1^*$ such that $\tilde{x}_1 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}_2$. Because of the base of induction, there is a transition labelled by $\tilde{\sigma} \in \Sigma_1$ such that $\tilde{x}_2 \overset{\tilde{\sigma}}{\mapsto}_{\tilde{G}} \tilde{x}_3$. Therefore, there is a string of transitions labelled by $\tilde{t}\tilde{\sigma} \in \Sigma_1^*$ such that $\tilde{x}_1 \overset{\tilde{t}\tilde{\sigma}}{\mapsto}_{\tilde{G}} \tilde{x}_3$. ■

Lemma 17 *Suppose a similar module system* $\{G_1, \ldots, G_n\}$ *is given that is used to construct* $G^{\|}$ *and* $\tilde{G}$ *with two states of the quotient automaton* $\tilde{x}_1, \tilde{x}_2 \in 2^X$*, a state* $x_1^{\|} \in X^{\|}$ *and a string of transitions labelled by* $\tilde{t} \in \Sigma_1^*$ *such that* $\tilde{x}_1 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}_2$ *and*

$Comp(x_1^{\|}) = \tilde{x}_1$. Then there exists a state of $G^{\|}$, $x_2^{\|} \in X^{\|}$ and a string of transitions labelled by $t^{\|} \in \Sigma_1^*$ such that $Comp(x_2^{\|}) = \tilde{x}_2$ and $x_1^{\|} \overset{t^{\|}}{\mapsto}_{G^{\|}} x_2^{\|}$.

Proof: The $\tilde{G}$ automaton can be thought of as a simulation of the corresponding $G^{\|}$ automaton. For $\tilde{x}_1 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}_2$, suppose that $\tilde{t}' < \tilde{t}$ and $\tilde{x}_1 \overset{\tilde{t}'}{\mapsto}_{\tilde{G}} \tilde{x}$. If $\sigma_g \in \Sigma_g$ and $\tilde{t}'\sigma_g < \tilde{t}$, then for a $x^{\|} \in Comp^{-1}(\tilde{x})$, the transition at $\tilde{x}$ labelled by σ_g corresponds in $G^{\|}$ to all component states in $x^{\|}$ updating simultaneously. If $\sigma_p \in \Sigma_{p1}$ and $\tilde{t}'\sigma_p < \tilde{t}$, then for a $x^{\|} \in Comp^{-1}(\tilde{x})$, the transition at $\tilde{x}$ labelled by σ_p corresponds in $G^{\|}$ to exactly one component state ${x^{\|}}^i$ in $x^{\|}$ for some $i \in \{1, \ldots, n\}$ updating on the occurrence of an event $\sigma_{pi} = \Psi_{1i}(\sigma_p)$.

Therefore, for every component state $x_1 \in \tilde{x}_1$, there must be a string of transitions labelled by events in Σ_1^* that leads to a component state $x_2 \in \tilde{x}_2$ according to the transition rules of G_1 and for every component state $x_2 \in \tilde{x}_2$, there must be a string of transitions labelled by events in Σ_1^* from a component state in $x_1 \in \tilde{x}_1$ to the component state x_2 according to the transition rules of G_1. All of these strings are able to occur synchronously with respect to the occurrence of global events and are copies of the string $\tilde{t}$ with some events from Σ_{p1} removed.

Furthermore, for the string of events $\tilde{t} \in \Sigma_1^*$ such that $\tilde{x}_1 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}_2$, the string $\tilde{t}$ can be split into a set of strings $T = \{t_{x_1^1 x_2^1}, \ldots, t_{x_1^m x_2^m}\} \subseteq \Sigma_1^*$ such that:

- For all $i \in \{1, \ldots m\}$, $t_{x_1^i x_2^i}$ is a copy of t with some Σ_{p1} events removed.
- $P_g(t_{x_1^1 x_2^1}) = \cdots = P_g(t_{x_1^m x_2^m})$
- For every component state $x_1^i \in \tilde{x}_1$, there is at least one component state $x_2^i \in \tilde{x}_2$ and a string of events $t_{x_1^i x_2^i} \in T$ such that $\delta^1(x_1^i, t_{x_1^i x_2^i}) = x_2^i$.
- For every component state $x_2^j \in \tilde{x}_2$, there is at least one component state $x_1^j \in \tilde{x}_1$ and a string of events $t_{x_1^j x_2^j} \in T$ such that $\delta^1(x_1^i, t_{x_1^i x_2^i}) = x_2^i$.

- $m = \max\{|\tilde{x}_1|, |\tilde{x}_2|\} \leq n$

Consider the ordered list of n strings

$$L_t = [t_{x_1^1 x_2^1}, \ldots, t_{x_1^m x_2^m}, t_{x_1^m x_2^m}, \ldots, t_{x_1^m x_2^m}].$$

Corresponding to L_t, there are two ordered lists of pairs of states in X

$$\begin{aligned} L_x^1 &= [x_1^1, \ldots, x_1^m, x_1^m, \ldots, x_1^m] \\ L_x^2 &= [x_2^1, \ldots, x_2^m, x_2^m, \ldots, x_2^m] \end{aligned}$$

such that if

$$\begin{aligned} x_{1\Psi}^{\|} &= (x_1^1, \ldots, x_1^m, x_1^m, \ldots, x_1^m) \\ x_{2\Psi}^{\|} &= (x_2^1, \ldots, x_2^m, x_2^m, \ldots, x_2^m), \end{aligned}$$

then

$$\begin{aligned} Comp(x_{1\Psi}^{\|}) &= \tilde{x}_1 \\ Comp(x_{2\Psi}^{\|}) &= \tilde{x}_2. \end{aligned}$$

Moreover, according to the transition rules of G_1, $\delta^1(x_{k,1}, t_k) = x_{k,2}$ where $L_t(k) = t_k$ is the kth string in L_t and $L_x^1(k) = x_{k,1}$ and $L_x^2(k) = x_{k,2}$ are the kth states in L_x^1 and L_x^2 respectively. It is discussed below how the assumption that the last $(n-m)$ pairings of states from the L_1 and L_2 lists are replicated can be done without loss of generality.

The list of strings L_t are now converted using the $\Psi_{1i}(\cdot)$ operator to the list

$$L_t^{\Psi} = [\Psi_{11}(t_{x_1 x_2}^1), \ldots, \Psi_{1m}(t_{x_1 x_2}^m), \Psi_{1(m+1)}(t_{x_1 x_2}^m), \ldots, \Psi_{1n}(t_{x_1 x_2}^m)].$$

Because all of the strings in this converted list can still be synchronized on the occurrence of events in Σ_g, the intersection

$$\left(\begin{array}{c} P_1^{-1}(\Psi_{11}(t^1_{x_1x_2})) \cap \ldots \cap P_m^{-1}(\Psi_{1m}(t^m_{x_1x_2})) \\ \cap\, P_{m+1}^{-1}(\Psi_{1(m+1)}(t^m_{x_1x_2}) \cap \ldots \cap P_n^{-1}(\Psi_{1n}(t^m_{x_1x_2}) \end{array} \right)$$

is nonempty. Let $t^{\|}_{\Psi}$ be some string in this intersection. From the construction above for L^1_x and L^2_x, if $x^{\|}_{1\Psi}$ is the n-tuple corresponding to the list of states in L^1_x and $x^{\|}_{2\Psi}$ is the n-tuple corresponding to the list of states in L^2_x, then $\delta^{\|}(x^{\|}_{1\Psi}, t^{\|}_{\Psi}) = x^{\|}_{2\Psi}$. Therefore, $x^{\|}_1 \overset{t^{\|}}{\mapsto}_{G^{\|}} x^{\|}_2$.

Notice that the lists L_t, L^1_x and L^2_x could be constructed to correspond to any state $x^{\|}_1$ such that $x^{\|}_1 \in Comp^{-1}(\tilde{x}_1)$ and $x^{\|}_1$ is the n-tuple corresponding to the list of states in L^1_x. ■

An example is now given to aid in the visualization of the constructions used in the proof of Lemma 17.

Example 26 *It is now shown how to construct lists L_t, L^1_x, L^2_x and the states $x^{\|}_{1\Psi}$, $x^{\|}_{1\Psi}$ in Lemma 17 from a given $\tilde{x}_1$, $\tilde{x}_2$, $x^{\|}_1$ and $\tilde{t}$ using the $\tilde{G}$ automaton in Example 24. Suppose $\tilde{x}_1 = \{1,2\}$, $\tilde{x}_2 = \{2,3\}$, $\tilde{t} = \alpha_1\gamma\beta_1\beta_1\alpha_1$ and $x^{\|}_1 = (1,1,2)$.*

From $x^{\|}_1 = (1,1,2)$, L^1_x is set to be $[1,1,2]$. Let $L_t = [\alpha_1\gamma\alpha_1, \alpha_1\gamma\beta_1, \gamma\beta_1]$ and let $L^2_x = [2,3,3]$. Therefore, $x^{\|}_{1\Psi} = (1,1,2)$ and $x^{\|}_{1\Psi} = (2,3,3)$. Note that

$$\begin{aligned} \delta_1(1, \alpha_1\gamma\alpha_1) &= 2, \\ \delta_1(1, \alpha_1\gamma\beta_1) &= 3, \\ \delta_1(2, \gamma\beta_1) &= 3. \end{aligned}$$

Also note that $L^{\Psi}_t = [\alpha_1\gamma\alpha_1, \alpha_2\gamma\beta_2, \gamma\beta_3]$ and it is possible for the string $t^{\|}$ corresponding to $\tilde{t}$ to be $\alpha_1\alpha_2\gamma\alpha_1\beta_2\beta_3$. Therefore, $\delta^{\|}((1,1,2), \alpha_1\alpha_2\gamma\alpha_1\beta_2\beta_3) = (2,3,3)$.

The dual of Lemma 17 is now shown where $x_2^{\|}$ is specified instead of $x_1^{\|}$.

Lemma 18 *Suppose a similar module system* $\{G_1, \ldots, G_n\}$ *is given that is used to construct* $G^{\|}$ *and* $\tilde{G}$ *with two states of the quotient automaton* $\tilde{x}_1, \tilde{x}_2 \in 2^X$*, a state* $x_2^{\|} \in X^{\|}$ *and a string of transitions labelled by* $\tilde{t} \in \Sigma^*$ *such that* $\tilde{x}_1 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}_2$ *and* $Comp(x_2^{\|}) = \tilde{x}_2$*. Then there exists a state of* $G^{\|}$*,* $x_1^{\|} \in X^{\|}$ *and string of transitions labelled by* $t^{\|} \in \Sigma_1^*$ *such that* $Comp(x_1^{\|}) = \tilde{x}_1$ *and* $x_1^{\|} \overset{t^{\|}}{\mapsto}_{G^{\|}} x_2^{\|}$*.*

Proof: This proof follows from the same construction of Lemma 17 except that the lists L_t, L_x^1 and L_x^2 could be constructed to correspond to any state $x_2^{\|}$ such that $x_2^{\|} \in Comp^{-1}(\tilde{x}_2)$ and $x_2^{\|}$ is the n-tuple corresponding to the list of states in L_x^2. ■

Theorem 35 *Suppose a similar module system* $\{G_1, \ldots, G_n\}$ *is given that is used to construct* $G^{\|}$ *and* $\tilde{G}$*. A state* $x^{\|} \in X^{\|}$ *deadlocks according to the transition rules of* $G^{\|}$ *if and only if the state* $Comp(x^{\|}) = \tilde{x}$ *deadlocks according to the transition rules of* $\tilde{G}$*.*

Proof: This theorem is demonstrated in several parts. Suppose that $x^{\|} \in X^{\|}$ deadlocks according to the transition rules of $G^{\|}$. Therefore, there are no global events $\sigma_g \in \Sigma_g$ that could synchronize a state transition in all component states of $x^{\|}$. Due to the construction of $\tilde{G}$, there are therefore no global event $\sigma_g \in \Sigma_g$ that could synchronize a state transition in all component states of $\tilde{x}$. Also, because there no private events $\sigma_{pi} \in \Sigma_{pi}$ that could occur at any of the component states of $x^{\|}$. There can therefore be no events $\sigma_{p1} \in \Sigma_{p1}$ that could occur at any of the states of $\tilde{x}$ according to the transition rules of G_1. Consequently there are no events $\sigma_{p1} \in \Sigma_{p1}$ that could occur at the state $\tilde{x}$ according to the transition rules of $\tilde{G}$.

Now suppose there is an event $\sigma_g \in \Sigma_g$ that can occur at state $x^{\|}$ and synchronize a state update at all component states according to the transition rules of $G^{\|}$. Due

to the construction of $\tilde{G}$, σ_g must also be able to occur at state $\tilde{x}$ according to the transition rules of $\tilde{G}$ and synchronize an update at all states in $\tilde{x}$.

Now suppose there is an event $\sigma_{pi} \in \Sigma_{pi}$ that can occur at state $x^{\|}$ and cause a private state update at exactly one component state of $x^{\|}$ according to the transition rules of G_i. Due to the construction of $\tilde{G}$, the corresponding σ_{pi} must also be able to occur at state $\tilde{x}$ according to the transition rules of $G^{\|}$ and cause a state update at one of the component states. ■

Theorem 36 *Suppose a similar module system $\{G_1, \ldots, G_n\}$ is given that is used to construct $G^{\|}$ and $\tilde{G}$. A state $x^{\|} \in X^{\|}$ is reachable according to the transition rules of $G^{\|}$ if and only if the state $Comp(x^{\|}) = \tilde{x}$ is reachable according to the transition rules of $\tilde{G}$.*

Proof: This theorem is demonstrated in two parts. For the first part of this proof, suppose that a state $x^{\|} \in X^{\|}$ is reachable according to the transition rules of $G^{\|}$. Therefore, there is a string of transitions labelled by $t^{\|}$ such that $x_0^{\|} \overset{t^{\|}}{\mapsto}_{G^{\|}} x^{\|}$. Because of Lemma 16, there is a string of transitions labelled by $\tilde{t}$ such that $\tilde{x}_0 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}$ where $Comp(x^{\|}) = \tilde{x}$.

For the second part of this proof, suppose that a state $\tilde{x} \in \tilde{X}$ is reachable according to the transition rules of $\tilde{G}$. Therefore, there is a string of transitions labelled by $\tilde{t}$ such that $\tilde{x}_0 \overset{\tilde{t}}{\mapsto}_{\tilde{G}} \tilde{x}$. Using Lemma 18 there is a string of transitions labelled by $t^{\|}$ such that for some $x_0^{\|'} \in Comp^{-1}(\tilde{x}_0)$, $x_0^{\|'} \overset{t^{\|}}{\mapsto}_{G^{\|}} x^{\|}$ Because $Comp^{-1}(\tilde{x}_0) = \{x_0^{\|}\}$, there is a string of transitions labelled by $t^{\|}$ such that $x_0^{\|} \overset{t^{\|}}{\mapsto}_{G^{\|}} x^{\|}$ where $Comp(x^{\|}) = \tilde{x}$. ■

Theorem 37 *Suppose a similar module system $\{G_1, \ldots, G_n\}$ is given that is used to construct $G^{\|}$ and $\tilde{G}$. A state $x^{\|} \in X^{\|}$ is blocking according to the transition rules*

of $G^{\|}$ if and only if the state $Comp(x^{\|}) = \tilde{x}$ is blocking according to the transition rules of $\tilde{G}$.

Proof: Note that according to the constructions of $G^{\|}$ and $\tilde{G}$, a state $x^{\|} \in X^{\|}$ is marked if and only if all states in $Comp^{-1}(Comp(x^{\|}))$ and $Comp(x^{\|})$ are marked.

Suppose that the state $x^{\|} \in X^{\|}$ is nonblocking according to the transition rules of $G^{\|}$. Therefore, there is a state $x_m^{\|} \in X_m^{\|}$ and a string of transitions labelled by $t^{\|}$ such that $x^{\|} \overset{t^{\|}}{\underset{G^{\|}}{\mapsto}} x_m^{\|}$. Because of Lemma 16, there is a string of transitions labelled by $\tilde{t}$ such that $\tilde{x} \overset{\tilde{t}}{\underset{\tilde{G}}{\mapsto}} \tilde{x}_m$ where $Comp(x_m^{\|}) = \tilde{x}_m$. Because $Comp(x_m^{\|}) = \tilde{x}_m$, $\tilde{x}_m$ must be marked.

Suppose that the state $\tilde{x} \in \tilde{X}$ is nonblocking according to the transition rules of $\tilde{G}$. Therefore, there is a state $\tilde{x}_m \in \tilde{X}_m$ and a string of transitions labelled by $\tilde{t}$ such that $\tilde{x} \overset{\tilde{t}}{\underset{\tilde{G}}{\mapsto}} \tilde{x}_m$. By Lemma 17 there is a string of transitions labelled by $t^{\|}$ and a state $x_2^{\|}$ such that $x^{\|} \overset{t^{\|}}{\underset{G^{\|}}{\mapsto}} x_2^{\|}$ and $Comp(x_2^{\|}) = \tilde{x}_m$. Because $Comp(x_2^{\|}) = \tilde{x}_m$, and $\tilde{x}_m$ is marked, then $x_2^{\|}$ must also be marked. Therefore $x^{\|}$ is nonblocking. ■

8.4.2 The Size of the State Space of $\tilde{G}$

It should be apparent that the $\tilde{G}$ constructed from the similar module system $\{G_1, \ldots, G_n\}$ has a smaller state space than the $\vec{G}$ or the $G^{\|}$ automata constructed from the same modules. The worst-case size of the $\tilde{G}$ automaton state space is now quantified. Suppose that $k = |X|$, the size of the state space of the individual modules. Consider the following example that shows how the $\tilde{G}$ construction uses a bounded number of states even if the number of modules is unbounded.

Example 27 *Consider the similar module system G_1, G_2 as in Example 20 and the corresponding $\tilde{G}_2$ constructed from these automata which can be seen in Fig-*

ure 8.7. The automata G_1, G_2 *use* $X = \{1, 2, 3\}$ *as their state space and* $\tilde{G}_2$ *uses* $\{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$ *as its state space.*

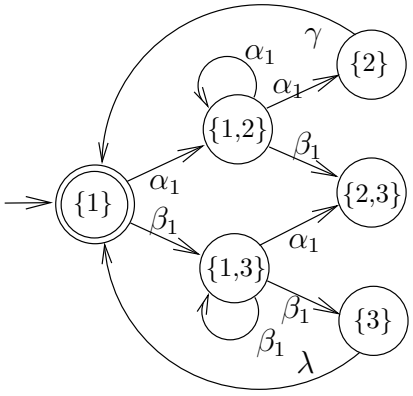

Figure 8.7: The automaton $\tilde{G}_2$ constructed from G_1, G_2.

Now consider the same set of similar modules except with three components G_1, G_2, G_3 *and corresponding* $\tilde{G}_3$ *automaton discussed in Example 25. The automata* G_1, G_2, G_3 *use* $X = \{1, 2, 3\}$ *as their state space and* $\tilde{G}$ *for this set of modules is* $2^X = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ *as its state space. Note that the largest cardinality subset of* X *is* $\{1, 2, 3\}$.

Now consider the same set of similar modules except with three components G_1, G_2, G_3, G_4 *and corresponding* $\tilde{G}_4$ *which can be seen in Figure 8.8. The automata* G_1, G_2, G_3 *use* $X = \{1, 2, 3\}$ *as their state space and* $\tilde{G}$ *for this set of modules is* $2^X = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ *as its state space. The only difference between the* $\tilde{G}_4$ *automaton and the* $\tilde{G}$ *automaton for the three module system seen in Figure 8.6 is that there is an added self-loop at state* $\{1, 2, 3\}$ *for* α_1 *and* β_1 *events. Therefore, after a certain number of modules is used to form the system, the construction of the* $\tilde{G}$ *automaton is unaffected if the number of modules is increased.*

Now consider the same set of n *similar modules* $G_1, \cdots, G_n$ *with* $n \geq 4$. *The*

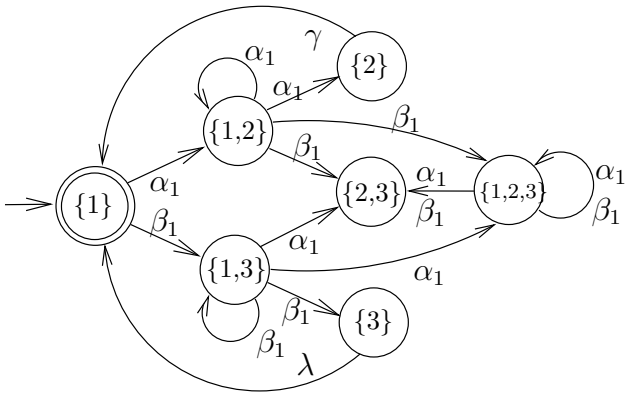

Figure 8.8: The automaton $\tilde{G}_4$ constructed from G_1, G_2, G_3, G_4.

automaton $\tilde{G}_n$ constructed from this set of modules still uses $2^X = \{\{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$ *as its state space and is structurally equivalent to the $\tilde{G}_4$ automaton. That is, all defined transitions in $\tilde{G}_n$ are also defined in $\tilde{G}_n$. Therefore, the $\tilde{G}_n$ construction is equivalent to the $\tilde{G}_4$ construction for no matter how many modules are used as long as $n \geq 4$. Furthermore, the size of the state space of $\tilde{G}_n$ does not change for $n \geq 3$. This is due to the fact that* $|X| = 3$.

Given an n-module system, each state of $\tilde{G}$ can have at most n component states and component states are not repeated in each state of $\tilde{G}$, so the maximum number of component states in $\tilde{G}$ is $\max(n, k)$. Therefore the worst-case size of $\tilde{G}$ needs to be analyzed in two cases, $n \geq k$ and $n < k$.

Case 1 *: $n < k$*

Each of the states of $\tilde{G}$ has at most n component states. There are possibly $\begin{pmatrix} k \\ i \end{pmatrix}$ states with i component states. Therefore if the sum of the possible number of state

sets for each number of components is taken from i to n, then the number of possible state sets for $\tilde{G}$ is:

$$\sum_{i=1}^{n} \left(\begin{array}{c} k \\ i \end{array} \right). \tag{8.3}$$

Note that for $n < k$, the number of possible state sets for $\tilde{G}$ is bounded by $2^k - 1$.

Case 2 *: $n \geq k$*

For this case, any state set in the power set 2^X is a possible state of $\tilde{G}$ except for the empty set $\emptyset$. Therefore, the size of the state space of $\tilde{G}$ is at most

$$|2^X \setminus \{\emptyset\}| = 2^k - 1. \tag{8.4}$$

This means that if $n \geq k$ then the size of $\tilde{G}$ is independent of the number of modules and therefore the computational complexity of testing state reachability, blocking and deadlock-freeness can be done without regard to the number of components.

Therefore if $\tilde{X}_{\tilde{G}}$ is the set of reachable states in $\tilde{G}$, then

$$|\tilde{X}_{\tilde{G}}| \leq \left\{ \begin{array}{ll} \sum_{i=1}^{n} \left(\begin{array}{c} k \\ i \end{array} \right) & \text{if } n < k \\ 2^k - 1 & \text{if } n \geq k \end{array} \right\} \tag{8.5}$$

Therefore, no matter how many modules comprise a similar module system, $|\tilde{X}_{\tilde{G}}| \leq 2^k - 1$. This is much less than the worst-case number of reachable states in $G^{\|}$, k^n. It seems very surprising at first that the verification of such important properties of systems such as state reachability and blocking can be performed with such a large reduction in the number of composed states that need to be searched over. It is also surprising that using the $\tilde{G}$ construction, the difficulty of performing

these verifications is nearly independent of the number of modules. This construction shows more of the computational advantages gained when investigating modular systems with symmetry between components.

8.5 Verification for Local Specifications

Now topics related to the verification of similar module systems with respect to local specifications are discussed. Only the local behavior without accounting for the interaction of the other system modules needs to be accounted for when verifying if local specifications are achieved when the modules are actually interacting.

Definition 21 *A set of languages* $\{K_1, \ldots, K_n\}$ *such that for all* $i \in \{1, \ldots, n\}$, $K_i \subseteq \Sigma_i^*$ *are called a set of similar language specifications if language* K_i *is a specification on module* G_i *in a similar module system* $\{G_1, \ldots, G_n\}$ *and for all* $i \in \{1, \ldots, n\}$, $\Psi_{1i}(K_1) = K_i$.

Suppose it needs to be checked if for all i that $P_i\left(\mathcal{L}_m(G_1\| \cdots \|G_n)\right) = K_i$. This property can be checked solely by verifying that $\mathcal{L}_m(G_i) = K_i$ for similar module systems. The following corollary is a direct extension of Theorem 30.

Corollary 12 *Given a local language specification* K_i *and similar module system* $\{G_1, \ldots, G_n\}$, $P_i\left(\mathcal{L}_m(G_1\| \cdots \|G_n)\right) = K_i$ *if and only if* $\mathcal{L}_m(G_i) = K_i$. *Likewise,* $P_i\left(\mathcal{L}(G_1\| \cdots \|G_n)\right) = \overline{K_i}$ *if and only if* $\mathcal{L}(G_i) = \overline{K_i}$.

Verifying $\mathcal{L}_m(G_i) = K_i$ and $\mathcal{L}(G_i) = \overline{K_i}$ are both known to be computationally simple if K_i is specified by a deterministic automaton and G_i is likewise deterministic. This greatly simplifies previously known verification methods based on more general modular systems because local behavior in a composed similar module system can be tested by investigating a single module.

8.6 Control Operations

Now properties related to the control of similar systems are explored. First the control framework is introduced which is a version of the standard decentralized supervisory control framework specialized for similar module systems.

Definition 22 *A set of controllers* $\{S_1, \ldots, S_n\}$ *are called* Similar Controllers *if they are all exact copies of one another with respect to* $\Psi_{ij}(\cdot)$ *operations. Therefore,* $\Sigma_{ci} = \Psi_{1i}(\Sigma_{c1})$, $\Sigma_{oi} = \Psi_{1i}(\Sigma_{o1})$ *and if controller* S_1 *disables events* $\gamma_1 \subseteq \Sigma_{c1}$ *after observing a string* s_1*, then controller* i *disables* $\Psi_{1i}(\gamma_1) \subseteq \Psi_{1i}(\Sigma_{c1})$ *after observing string* $\Psi_{1i}(s_1)$.

The controllers $\{S_1, \ldots, S_n\}$ are non-blocking for $\{G_1, \ldots, G_n\}$ if

$$\overline{\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n))} = \mathcal{L}((S_1/G_1)\| \cdots \|(S_n/G_n)).$$

8.7 Control for Local Specifications

There are potentially great reductions in computational effort for many similar module system control problems with local specifications. For a similar module system $\{G_1, \ldots, G_n\}$ and a set of similar language specifications $\{K_1, \ldots, K_n\}$, suppose it is desirable to know if there exists a set of similar controllers $\{S_1, \ldots, S_n\}$ such that

$$\forall i \in \{1, \ldots, n\} \quad \left(\begin{array}{l} P_i\left[\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n))\right] = K_i \\ \qquad \wedge P_i\left[\mathcal{L}((S_1/G_1)\| \cdots \|(S_n/G_n))\right] = \overline{K_i}. \end{array} \right)$$

This problem can be solved by looking only at the local behavior of G_1 and the locally observable and controllable event sets, Σ_{o1} and Σ_{c1}, respectively.

Theorem 38 *For a similar module system* $\{G_1, \ldots, G_n\}$ *with respective local event sets* $\{\Sigma_1, \ldots, \Sigma_n\}$*, observable event sets* $\{\Sigma_{o1}, \ldots, \Sigma_{on}\}$*, controllable event sets* $\{\Sigma_{c1}, \ldots, \Sigma_{cn}\}$ *and local behavior specifications* $\{K_1, \ldots, K_n\}$ *such that for all* $i, j \in \{1, \ldots, n\}$*,* $K_i \neq \emptyset$ *and* $K_i \subseteq \mathcal{L}_m(G_i)$*,* $K_j = \Psi_{ij}(K_i)$*, there exists a set of similar controllers* $\{S_1, \ldots, S_n\}$ *such that for all* $i \in \{1, \ldots, n\}$*,*

$$P_i\left[\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n))\right] = K_i$$

and

$$P_i\left[\mathcal{L}((S_1/G_1)\| \cdots \|(S_n/G_n))\right] = \overline{K_i}$$

if and only if

1. K_1 *is controllable with respect to* $\mathcal{L}(G_1)$ *and* Σ_{uc1}*.*

2. K_1 *is observable with respect to* $\mathcal{L}(G_1)$*,* P_{o1} *and* Σ_{c1}*.*

3. K_1 *is* $\mathcal{L}_m(G_1)$*-closed.*

Proof: It was shown in Corollary 12 that $P_i\left[\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n))\right] = K_i$ and $P_i\left[\mathcal{L}((S_1/G_1)\| \cdots \|(S_n/G_n))\right] = \overline{K_i}$ if and only if $\mathcal{L}_m(S_i/G_i) = K_i$ and $\mathcal{L}(S_i/G_i) = \overline{K_i}$. Due to the symmetry of the model, controllers and specifications, $\mathcal{L}_m(S_i/G_i) = K_i$ and $\mathcal{L}(S_i/G_i) = \overline{K_i}$ if and only if $\mathcal{L}_m(S_1/G_1) = K_1$ and $\mathcal{L}(S_1/G_1) = \overline{K_1}$. The result then follows using the controllability and observability theorem of supervisory control theory [44]. ■

Theorem 38 shows that controller existence for local behavior specifications can be decided by testing controller existence locally and apart from the interaction of other modules. The controllability and observability theorem is known to be constructive so therefore there are known methods for synthesizing the local controllers

$\{S_1, \ldots, S_n\}$ such that local non-blocking specifications are satisfied at all nodes when the modules interact and the conditions of Theorem 38 are satisfied. These results also generalize to the cases where marking properties are not of a concern.

Despite these very positive results a caveat is in order with respect to the nature of language semantics. All local modules in a similar module system may be non-blocking and deadlock free, but blocking and deadlock may both still occur globally. Consider the following example.

Example 28 *Consider the automaton $G_1 \| G_2$ introduced in Example 20. $G_1 \| G_2$ is blocking and contains two deadlock states, $(2,3)$ and $(3,2)$. These states are reached when an α event is followed immediately by a β event or when a β event is followed immediately by an α event. Neither G_1 or G_2 can observe the interleaving of α and β events due to the restriction that the local systems cannot observe events that are private to other modules. However, when the language generated by the system $G_1 \| G_2$ is projected down to either the Σ_1 or Σ_2 event sets, the behavior is equivalent to G_1 or G_2, respectively, and these automata are individually non-blocking and deadlock-free.*

Example 28 shows one of the major limitations of language semantics and motivates why attention should not be restricted solely to local behavior. For concurrent systems, blocking and deadlock properties are inherently global in nature which prompts the investigations of global specifications.

Suppose a set of trim similar specification automata $\{H_1, \ldots, H_n\}$ is given such that $\forall i \in \{1, \ldots, n\}$ $\mathcal{L}_m(H_i) = K_i$. One way to ensure that the global composed system is non-blocking after the local controllers are designed would be to verify that $\{K_1, \ldots, K_n\}$ are non-conflicting by testing if $H^{\|}$ is non-blocking. This can be verified fairly efficiently by testing if the $\tilde{H}$ construction is non-blocking as seen in

Theorem 37. Another approach to ensure that the global controlled system is non-blocking would be to use global control specifications instead of local specifications. This situation is explored in the following section.

8.8 Control for Global Specifications

Instead of considering a set of local similar specifications $\{K_1, \ldots, K_n\}$, suppose a global language specification K is given and it needs to be decided if there exist non-blocking similar controllers $\{S_1, \ldots, S_n\}$ for a similar module system $\{G_1, \ldots, G_n\}$ such that $\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n)) = K$. Due to the similarity of the controllers and system modules one would think that K would need to exhibit a degree of symmetry with respect to the occurrence of private events. This is exactly the case which is found in Theorem 39 below.

Theorem 39 *For a similar module system $\{G_1, \ldots, G_n\}$ with respective local projection operations $\{P_1, \ldots, P_n\}$, observation projections $\{P_{o1}, \ldots, P_{on}\}$, controllable event sets $\{\Sigma_{c1}, \ldots, \Sigma_{cn}\}$ and global behavior specification K such that $K \neq \emptyset$ and $K \subseteq \mathcal{L}_m(G_1\| \cdots \|G_n)$, a set of non-blocking similar controllers $\{S_1, \ldots, S_n\}$ exists such that $\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n)) = K$ if and only if*

1. *$P_1(K)$ is controllable with respect to $\mathcal{L}(G_1)$ and Σ_{uc1}.*

2. *$P_1(K)$ is observable with respect to $\mathcal{L}(G_1)$, P_{o1} and Σ_{c1}.*

3. *$P_1(K)$ is $\mathcal{L}_m(G_1)$-closed.*

4. *K is symmetric with respect to $\{\Sigma_1, \ldots, \Sigma_n\}$.*

5. *K is decomposable with respect to $\{P_1, \ldots, P_n\}$.*

6. *$\{P_1^{-1}(P_1(K)), \ldots, P_n^{-1}(P_n(K))\}$ are non-conflicting.*

Proof: This theorem is shown in two parts. Assume there exists a set of controllers $\{S_1, \ldots, S_n\}$ such that

$$K = \mathcal{L}_m((S_1/G_1)\| \cdots \|(S_i/G_i)\| \cdots \|(S_n/G_n)) \tag{8.6}$$

$$\overline{K} = \mathcal{L}((S_1/G_1)\| \cdots \|(S_i/G_i)\| \cdots \|(S_n/G_n)). \tag{8.7}$$

By the definition of $\Psi_{1i}(\cdot)$:

$\Psi_{1i}(\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_i/G_i)\| \cdots \|(S_n/G_n))) =$

$\mathcal{L}_m((S_i/G_i)\| \cdots \|(S_1/G_1)\| \cdots \|(S_n/G_n))$.

The parallel composition operation is commutative, so:

$\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_i/G_i)\| \cdots \|(S_n/G_n)) =$

$\mathcal{L}_m((S_i/G_i)\| \cdots \|(S_1/G_1)\| \cdots \|(S_n/G_n))$.

$\Rightarrow K = \Psi_{1i}(K)$. This proves the fourth part of the implication.

By Equations 8.6 and 8.7 there exists a set of controllers $\{S_1, \ldots, S_n\}$ such that

$K = \mathcal{L}_m((S_1/G_1)\| \cdots \|(S_i/G_i)\| \cdots \|(S_n/G_n))$

$\overline{K} = \mathcal{L}((S_1/G_1)\| \cdots \|(S_i/G_i)\| \cdots \|(S_n/G_n))$

$\Rightarrow$

there exists a set of controllers $\{S_1, \ldots, S_n\}$ such that $\forall i \in \{1, \ldots, n\}$

$P_i(K) = P_i(\mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n)))$

$P_i(\overline{K}) = P_i(\mathcal{L}((S_1/G_1)\| \cdots \|(S_n/G_n)))$

$\Rightarrow$ using Corollary 12:

there exists a set of controllers $\{S_1, \ldots, S_n\}$ such that $\forall i \in \{1, \ldots, n\}$

$P_i(K) = \mathcal{L}_m(S_i/G_i)$

$P_i(\overline{K}) = \mathcal{L}(S_i/G_i)$

$\Rightarrow$ from the controllability and observability theorem of supervisory control theory,

1. $P_i(K)$ is controllable with respect to $\mathcal{L}(G_i)$ and Σ_{uci}.

2. $P_i(K)$ is observable with respect to $\mathcal{L}(G_i)$, P_{oi} and Σ_{ci}.

3. $P_i(K)$ is $\mathcal{L}_m(G_i)$-closed.

This proves the first three parts of the implication.

It is known that:

$K = P_1^{-1}(\mathcal{L}_m(S_1/G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(S_n/G_n))$

$\overline{K} = P_1^{-1}(\mathcal{L}(S_1/G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}(S_n/G_n))$

$\Rightarrow$ using Corollary 12:

$K = P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))$,

$\overline{K} = P_1^{-1}(\overline{P_1(K)}) \cap \cdots \cap P_n^{-1}(\overline{P_n(K)})$

$\Rightarrow$ using Lemma 12:

$K = P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))$,

$\overline{K} = \overline{P_1^{-1}(P_1(K))} \cap \cdots \cap \overline{P_n^{-1}(P_n(K))}$

This proves the fifth and sixth parts of the implication. This completes one direction of the proof.

It is now shown that the controllers exist if the six conditions hold.

Because of the first three conditions it is known that there exists a controller S_1 such that $P_1(K) = \mathcal{L}_m(S_1/G_1)$ and $\overline{P_1(K)} = \mathcal{L}(S_1/G_1)$.

Because of the decomposability condition, $K = P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))$.

Because of the symmetry condition and Lemma 13 it is known there exists controllers $S_1, \ldots, S_n$ such that $\forall i \in \{1, \ldots, n\}$: $(P_i(K) = \mathcal{L}_m(S_i/G_i))$.

Therefore by substitution,

$K = P_1^{-1}(\mathcal{L}_m(S_1/G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}_m(S_n/G_n))$.

This implies that

$K = \mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n))$.

Because of Condition 6 and Lemma 12:

$\overline{P_1^{-1}(P_1(K)) \cap \cdots \cap P_n^{-1}(P_n(K))} = P_1^{-1}(\overline{P_1(K)}) \cap \cdots \cap P_n^{-1}(\overline{P_n(K)})$.

$\Rightarrow$ Because of Conditions 5 and 6, $\overline{K}$ is decomposable and:

$\overline{K} = P_1^{-1}(\overline{P_1(K)}) \cap \cdots \cap P_n^{-1}(\overline{P_n(K)})$.

Because of $\overline{P_1(K)} = \mathcal{L}(S_1/G_1)$, the symmetry condition and Lemma 13 it is known that there exists controllers $S_1, \ldots, S_n$ such that $\forall i \in \{1, \ldots, n\}$: $(\overline{P_i(K)} = \mathcal{L}(S_i/G_i))$.

Therefore, for the set of similar controllers $\{S_1, \ldots, S_n\}$,

$\overline{K} = P_1^{-1}(\mathcal{L}(S_1/G_1)) \cap \cdots \cap P_n^{-1}(\mathcal{L}(S_n/G_n))$

$\Rightarrow$

Therefore there exists a set of similar controllers $\{S_1, \ldots, S_n\}$ such that

$K = \mathcal{L}_m((S_1/G_1)\| \cdots \|(S_n/G_n))$

$\overline{K} = \mathcal{L}((S_1/G_1)\| \cdots \|(S_n/G_n))$. ■

The six necessary and sufficient conditions for controller existence in Theorem 39 can be divided into two types. The first three conditions (local controllability, observability and $\mathcal{L}_m$-closure) are essentially existence conditions for local controllers to achieve local projections of global behavior. These conditions are inherent to many supervisory control problems and have been well known from the early papers in supervisory control theory [44, 54]. However, the combination of the last three conditions (symmetry, decomposability and non-conflictingness) is unique to this problem setting.

The symmetry condition ensures that if K is decomposable, then its decomposition is a similar module system. This condition in hindsight should be expected

when one considers Theorem 32. If the behavior of all controllers operating on the modules is similar with respect to a renaming of local events, then the specification would necessarily be symmetric if it can be achieved.

The decomposability condition ensures that the specification K can be constructed from its local behavior projections $P_1, \ldots, P_n$. Therefore, because it is known that the local specifications are achieved by the local behavior, i.e., $P_i(K) = \mathcal{L}_m(S_i/G_i)$, the decomposability condition forces the global controlled behavior to be equivalent to the global behavior specification. The non-conflicting condition ensures that the local non-blocking behavior of each controller ensures that the global controlled behavior is non-blocking.

Taken together, Conditions 4 through 6 imply that the global specification needs to be expressible as similar non-conflicting local specifications if there exists a set of similar controllers that can be coupled with the similar module system to achieve the specification.

Suppose a trim automaton H is given such that $\mathcal{L}_m(H) = K$ where K satisfies the conditions of Theorem 39. Using the decomposition algorithm, Algorithm 11, a set of trim automata $\{H_1, \ldots, H_n\}$ can be constructed from H such that $\forall i \in \{1, \ldots, n\}$ $\mathcal{L}_m(H_i) = K_i$ and $P_1^{-1}(K_1) \cap \cdots \cap P_n^{-1}(K_n) = K$ from Theorems 33 and 34. The local similar controllers $\{S_1, \ldots, S_n\}$ can then be synthesized very quickly using known methods for centralized system using the result of Theorem 38.

Note that Conditions 1 through 6 together imply that the language K is controllable with respect to $\mathcal{L}(G_1 \| \cdots \| G_n)$, co-observable with respect to $\mathcal{L}(G_1 \| \cdots \| G_n)$ and $\mathcal{L}_m(G_1 \| \cdots \| G_n)$-closed, but the reverse implication does not hold because of the assumptions on the controllers being used. If the controllers were allowed to be asymmetric, a larger class of specifications could be achieved, but this would require

an extra amount of coordination in the control synthesis. The model is designed so that once one control module is designed, the implementation of more controllers is merely a matter of copying that first controller.

8.9 Discussion

A special class of permutation symmetric systems has been introduced that can be used to model a wide variety of real-world processes. A method has been shown that can be used to test reachability, global blocking and deadlock-freeness without enumerating all possible combinations of system states. It was also shown that the verification of local behavior for these systems can be performed in an off-line manner without system interaction. Necessary and sufficient conditions for the existence of local controllers for the similar module model have also been introduced.

A method for decomposing an automaton representing global system behavior into automata representing the local, modular subsystems has been shown that is computationally reasonable and runs in polynomial time with respect to the size of the local specification. This result is important because the standard methods for performing decompositions currently known in the standard supervisory control literature take exponential time with respect to the size of the global model in the worst case.

It would be interesting to develop more efficient tests for when the necessary and sufficient conditions for controller existence in Theorem 39 are satisfied. There are several known methods for testing these properties, but these methods do not take advantage of the possible symmetry in the systems discussed here and hence can be computationally expensive. Once these conditions can be tested efficiently, local control modules can be synthesized easily using Algorithm 11 and known centralized

controller synthesis methods. It would also be interesting to develop online methods for controlling these systems when the necessary and sufficient conditions for controller existence do not hold, but safety conditions need to be satisfied. It would also be desirable to synthesize controllers that achieve behavior that is "maximal" in some sense as was done for centralized systems in Chapter V.

CHAPTER IX

CONCLUSIONS

9.1 Chapter Overview

This chapter reviews the main results of this thesis and then proposes several areas for continued future research.

9.2 Thesis Overview

This thesis discusses computational problems related to the control and verification of distributed systems. The framework of supervisory control of discrete-event systems is used where system modules are modelled as finite-state automata that coordinate on the occurrence of common events.

It is shown in Chapter IV that in general many important control and verification problems for these systems are PSPACE-complete. The results of this chapter are also disappointing because (as was mentioned previously), it is generally believed that deterministic finite-state automata problems are fairly simple and these results indicate that many modular problems using more general system and specification models are also intractable. This motivates the work in the later chapters of this thesis for finding methods to avoid this computational difficulty for problems of particular interest.

Despite the negative results regarding the time-complexity of the problems discussed in Chapter IV, it is shown in this chapter that the problems are in PSPACE. Therefore, there are *always* space-efficient solutions to the problems discussed here for deterministic finite-state automata modules. These results are in a sense positive in that the *state* explosion problem inherent to distributed systems can be avoided, as far as computation space is concerned,. In the worst case the size of the state space of a composed system is exponential in the number of modules, but at worst only a small fraction of those modules need to be stored in memory to solve many problems if necessary.

Chapter V introduces several new online decentralized control protocols to avoid the computational difficulty of calculating precomputed control actions. These new online protocols rely on a new state estimator function. These protocols have a common easily testable sufficient safety condition that is effectively an actuator selection problem.

A sensor selection problem is discussed in Chapter VI where it was shown that even for centralized control systems a minimal sensor selection is difficult to approximate. The sensor selection problem is converted to a type of edge colored directed graph *st*-cut problem and heuristic methods are developed to approximate minimal solutions to this problem. It is shown how these methods for the edge colored directed graph *st*-cut problem can also be used to approximate solutions to a communicating controller problem.

The graph cutting method for solving the sensor selection problems are very powerful in that they can be used to force an event be observable or not without greatly modifying methods for solving the problem. This method is also useful when for an important actuator selection problem where it is desired to find the

minimal number of actuators needed to synthesize a controller that satisfies a safety specification. Many of the methods also discussed could be used to solve a related sensor selection problem where the cost of selecting a sensor is non-uniform.

Chapter VII discusses a special class of distributed system that contain a property called state permutation symmetry. A special version of the μ-calculus is introduced such that all permutation equivalent states in a composed distributed system satisfy the same μ-calculus formulas. A quotient structure is introduced for these systems that can be used to more efficiently test if the systems satisfy propositions in the restricted μ-calculus.

Chapter VII discusses the case where the control modules are perfectly symmetric, but it might be the case in many real-world problems that the systems may be slightly asymmetric. For instance, for some control systems, one controller might have more leeway in enforcing global control actions to avoid situations where blocking and deadlock may occur. This is an interesting problem for future investigations that could build on the symmetric system investigations in Chapter VII.

Chapter VIII investigates a special class of symmetric distributed systems where the behavior of the system modules is restricted to be either global or private. A very efficient decomposition method is given to convert a composed version of these special systems into distributed modules. These systems also allow for the construction of quotient automata with even smaller state spaces than the ones presented in Chapter VII for testing important system properties. Concise necessary and sufficient conditions exist for testing for controller existence on these systems that are based on previously known properties. These properties can also be used to reduce the computational difficulty of solving controller verification problems.

9.3 Ongoing Research

This thesis has shown that many important problems related to the control and verification of distributed discrete-event systems are computationally difficult to solve in general. However, despite this computational difficulty, these problems are becoming increasingly important in the modern world and methods need to be developed to handle them. Therefore, methods need to be developed to solve special cases of these problems that have particular real-world relevance. To this end, this thesis shows practical methods to solve several special problem cases.

This avenue of research is of course ongoing and as new applications areas of supervisory control are investigated new special cases of the control and verification problems will gain practical relevance. This highlights an important aspect of solution methods for the problems discussed in this thesis; although the methods shown here are theoretical nature, they are necessarily driven by practical application and this practical relevance should necessarily continue to be an important motivating factor for future research.

With the need for practical relevance in mind, future efforts to solve the computationally difficult problems discussed in this thesis could perhaps be driven by average case analysis. Throughout this thesis in particular and control theory in general, when computational methods are developed to solve problems, the strongest emphasis is usually placed on worst case analysis. However, for many applications, worst case analysis may not be particularly relevant. Therefore, it may be commonly sufficient to develop solution methods that are optimal in some sense for the "average" case. Unfortunately, average case analysis is generally more difficult than worst case analysis as it is difficult at times to quantify what an average case of a problem

is.

The interacting automata modelling formalism used in this thesis is generally considered to be the simplest one for distributed systems that is sufficiently expressible to model real systems. Therefore, an additional avenue for future research would be to use more general system models which include Petri nets and hybrid automata which would be able to model a much larger class of systems. There has already been some investigations into computational methods for the control and verification of these system models, but all of the control and verification problems discussed in this thesis are at least as difficult when these models are used and generally much more so. Indeed, even some of the simplest problems such as state reachability are undecidable with these models. Therefore it would be particularly useful to develop methods to avoid the computational difficulty of control and verification problems for these systems. Indeed, the methods shown in this thesis such as the sufficient safe control protocols, approximation methods and symmetry reductions would hopefully also be useful for these more general systems.

APPENDICES

APPENDIX A

The 2-Controller $\mathcal{M}$ Automaton

Suppose a specification automaton $H = (X^H, x_0^H, \Sigma, \delta^H)$, a system automaton $G = (X^G, x_0^G, \Sigma, \delta^G)$, observable event sets Σ_{o1} and Σ_{o2} and controllable event sets Σ_{c1} and Σ_{c2} are given to test if $\mathcal{L}(H)$ is co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2}, Σ_{c1} and Σ_{c2}. This can be done using the $\mathcal{M}$ automaton construction shown in [67]. Let us define:

$$\mathcal{M} = (X^{\mathcal{M}}, x_0^{\mathcal{M}}, \Sigma, \delta^{\mathcal{M}}, X_m^{\mathcal{M}})$$

where

$$\begin{aligned} X^{\mathcal{M}} &:= X^H \times X^H \times X^H \times X^G \cup \{d\}, \\ x_0^{\mathcal{M}} &:= (x_0^H, x_0^H, x_0^H, x_0^G), \\ X_m^{\mathcal{M}} &:= \{d\}. \end{aligned}$$

Let us define the set of conditions that together imply a violation of co-observability. Note that these conditions are only defined for the controllable events. For $\sigma \in \Sigma_c$[1],

[1]This condition is not mentioned in [67].

the following set of conditions are called the $(*)$ conditions.

$$\left.\begin{array}{ll} \delta^H(x_1,\sigma) & \text{is defined if } \sigma \in \Sigma_{c1} \\ \delta^H(x_2,\sigma) & \text{is defined if } \sigma \in \Sigma_{c2} \\ \delta^H(x_3,\sigma) & \text{is not defined} \\ \delta^G(x_4,\sigma) & \text{is defined} \end{array}\right\} \quad (*)$$

The transition relation $\delta^{\mathcal{M}}$ is defined as follows.

For $\sigma \notin \Sigma_{o1}$ and $\sigma \notin \Sigma_{o2}$,

$$\delta^{\mathcal{M}}((x_1,x_2,x_3,x_4),\sigma) = \left\{\begin{array}{l} (\delta^H(x_1,\sigma),x_2,x_3,x_4) \\ (x_1,\delta^H(x_2,\sigma),x_3,x_4) \\ (x_1,x_2,\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ (\delta^H(x_1,\sigma),\delta^H(x_2,\sigma),\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ d \quad \text{if } (*) \end{array}\right\}$$

For $\sigma \notin \Sigma_{o1}$ and $\sigma \in \Sigma_{o2}$,

$$\delta^{\mathcal{M}}((x_1,x_2,x_3,x_4),\sigma) = \left\{\begin{array}{l} (\delta^H(x_1,\sigma),x_2,x_3,x_4) \\ (x_1,\delta^H(x_2,\sigma),\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ (\delta^H(x_1,\sigma),\delta^H(x_2,\sigma),\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ d \quad \text{if } (*) \end{array}\right\}$$

For $\sigma \in \Sigma_{o1}$ and $\sigma \notin \Sigma_{o2}$,

$$\delta^{\mathcal{M}}((x_1,x_2,x_3,x_4),\sigma) = \left\{\begin{array}{l} (x_1,\delta^H(x_2,\sigma),x_3,x_4) \\ (\delta^H(x_1,\sigma),x_2,\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ (\delta^H(x_1,\sigma),\delta^H(x_2,\sigma),\delta^H(x_3,\sigma),\delta^G(x_4,\sigma)) \\ d \quad \text{if } (*) \end{array}\right\}$$

For $\sigma \in \Sigma_{o1}$ and $\sigma \in \Sigma_{o2}$,

$$\delta^{\mathcal{M}}((x_1, x_2, x_3, x_4), \sigma) = \left\{ \begin{array}{l} (\delta^H(x_1, \sigma), \delta^H(x_2, \sigma), \delta^H(x_3, \sigma), \delta^G(x_4, \sigma)) \\ d \quad \text{if } (*) \end{array} \right\}$$

For $\sigma \in \Sigma$, $\delta^{\mathcal{M}}(d, \sigma)$ is undefined.

The state d is reachable from the initial state in $\mathcal{M}$ if and only if $\mathcal{L}(H)$ is co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2}, Σ_{c1} and Σ_{c2}.

APPENDIX B

The 2-Controller $\mathcal{M}_d$ Automaton

Suppose a specification automaton $H = (X^H, x_0^H, \Sigma, \delta^H)$, a system automaton $G = (X^G, x_0^G, \Sigma, \delta^G)$, observable event sets Σ_{o1} and Σ_{o2} and controllable event sets Σ_{c1} and Σ_{c2} are given such that it should be tested if $\mathcal{L}(H)$ is D&A co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2}, Σ_{c1} and Σ_{c2}. This can be done using the $\mathcal{M}_d$ automaton construction shown in [83]. Let us define:

$$\mathcal{M}_d = (X^{\mathcal{M}_d}, x_0^{\mathcal{M}_d}, \Sigma, \delta^{\mathcal{M}_d}, X_m^{\mathcal{M}_d})$$

where

$$\begin{aligned} X^{\mathcal{M}_d} &:= X^G \times X^H \times X^G \times X^H \times X^H \cup \{d\}, \\ x_0^{\mathcal{M}_d} &:= (x_0^G, x_0^H, x_0^G, x_0^H, x_0^H), \\ X_m^{\mathcal{M}_d} &:= \{d\}. \end{aligned}$$

Let us define the set of conditions that together imply a violation of D&A co-observability. Note that these conditions are only defined for the controllable events.

For $\sigma \in \Sigma_c$, the following set of conditions are called the $(*)$ conditions.

$$\left.\begin{array}{ll} \delta^G(x_1,\sigma) & \text{is defined if } \sigma \in \Sigma_{c1} \\ \delta^H(x_2,\sigma) & \text{is not defined if } \sigma \in \Sigma_{c1} \\ \delta^G(x_3,\sigma) & \text{is defined if } \sigma \in \Sigma_{c2} \\ \delta^H(x_4,\sigma) & \text{is not defined if } \sigma \in \Sigma_{c2} \\ \delta^H(x_4,\sigma) & \text{is defined} \end{array}\right\} \quad (*)$$

The transition relation $\delta^{\mathcal{M}_d}$ is defined as follows.

For $\sigma \notin \Sigma_{o1}$ and $\sigma \notin \Sigma_{o2}$,

$$\delta^{\mathcal{M}_d}((x_1,x_2,x_3,x_4,x_5),\sigma) = \left\{\begin{array}{l} (\delta^G(x_1,\sigma),\delta^H(x_2,\sigma),x_3,x_4,x_5) \\ (x_1,x_2,\delta^G(x_3,\sigma),\delta^H(x_4,\sigma),x_5) \\ (x_1,x_2,x_3,x_4,\delta^H(x_5,\sigma)) \\ (\delta^G(x_1,\sigma),\delta^H(x_2,\sigma),\delta^G(x_3,\sigma),\delta^G(x_4,\sigma),\delta^H(x_5,\sigma)) \\ d \quad \text{if } (*) \end{array}\right\}$$

For $\sigma \notin \Sigma_{o1}$ and $\sigma \in \Sigma_{o2}$,

$$\delta^{\mathcal{M}_d}((x_1,x_2,x_3,x_4),\sigma) = \left\{\begin{array}{l} (\delta^G(x_1,\sigma),\delta^H(x_2,\sigma),x_3,x_4,x_5) \\ (x_1,x_2,\delta^G(x_3,\sigma),\delta^H(x_4,\sigma),\delta^H(x_5,\sigma)) \\ (\delta^G(x_1,\sigma),\delta^H(x_2,\sigma),\delta^G(x_3,\sigma),\delta^G(x_4,\sigma),\delta^H(x_5,\sigma)) \\ d \quad \text{if } (*) \end{array}\right\}$$

For $\sigma \in \Sigma_{o1}$ and $\sigma \notin \Sigma_{o2}$,

$$\delta^{\mathcal{M}_d}((x_1, x_2, x_3, x_4), \sigma) =$$
$$\left\{ \begin{array}{l} (\delta^G(x_1, \sigma), \delta^H(x_2, \sigma), x_3, x_4, \delta^H(x_5, \sigma)) \\ (x_1, x_2, \delta^G(x_3, \sigma), \delta^H(x_4, \sigma), x_5) \\ (\delta^G(x_1, \sigma), \delta^H(x_2, \sigma), \delta^G(x_3, \sigma), \delta^G(x_4, \sigma), \delta^H(x_5, \sigma)) \\ d \quad \text{if } (*) \end{array} \right\}$$

For $\sigma \in \Sigma_{o1}$ and $\sigma \in \Sigma_{o2}$,

$$\delta^{\mathcal{M}_d}((x_1, x_2, x_3, x_4), \sigma) =$$
$$\left\{ \begin{array}{l} (\delta^G(x_1, \sigma), \delta^H(x_2, \sigma), \delta^G(x_3, \sigma), \delta^G(x_4, \sigma), \delta^H(x_5, \sigma)) \\ d \quad \text{if } (*) \end{array} \right\}$$

For $\sigma \in \Sigma$, $\delta^{\mathcal{M}_d}(d, \sigma)$ is undefined.

The state d is reachable from the initial state in $\mathcal{M}_d$ if and only if $\mathcal{L}(H)$ is D&A co-observable with respect to $\mathcal{L}(G)$, Σ_{o1}, Σ_{o2}, Σ_{c1} and Σ_{c2}.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers for partial observation. *Theoretical Computer Science*, 303:7–34, 2003.

[2] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kahn, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag, Berlin Heidelberg, 1999.

[3] S. Bala. Intersection of regular languages and star hierarchy. In *Proc. 29th Int. Colloquium Automata, Languages and Programming*, pages 159–169, 2002.

[4] G. Barrett. *Modeling, Analysis and Control of Centralized and Decentralized Logical Discrete-Event Systems*. PhD thesis, Electrical Engineering and Computer Science Department, The University of Michigan, 2000. Available at `http://www.eecs.umich.edu/umdes`.

[5] A. Bergeron. Sharing out control in distributed processes. *Theoretical Computer Science*, 139:163–186, 1995.

[6] V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

[7] B.A. Brandin, R. Malik, and P. Dietrich. Incremental system verification and synthesis of minimally restrictive behaviors. In *Proc. of 2000 American Control Conference*, pages 4056–4061, 2000.

[8] H.-D. Burkhard. Fairness and control in multi-agent systems. *Theoretical Computer Science*, 189:109–127, 1997.

[9] S. Buss, C. Papadimitriou, and J. Tsitsiklis. On the predictability of coupled automata: An allegory about chaos. *Complex Systems*, 5:525–539, 1991.

[10] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, MA, 1999.

[11] Y.L. Chen, S. Lafortune, and F. Lin. Design of nonblocking modular supervisors using event priority functions. *IEEE Trans. Auto. Contr.*, 45(3):432–452, March 2000.

[12] H. Cho. Designing observation functions in supervisory control. In *Proc. Korean Automatic Control Conference*, pages 523–528, 1990.

[13] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Auto. Contr.*, 33(3):249–260, March 1988.

[14] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 2002.

[15] D.Z. Du and K.I. Ko. *Theory of Computational Complexity*. John Wiley and Sons, Inc., 2000.

[16] E.A. Emmerson and A.P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, August 1996.

[17] J.M. Eyzell and J.E.R. Cury. Exploiting symmetry in the synthesis of supervisors for discrete event systems. *IEEE Trans. Auto. Contr.*, 46(9):1500–1505, September 2001.

[18] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[19] P. Gohari and W.M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):643–652, 2000.

[20] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin. Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 6:379–427, 1996.

[21] A. Haji-Valizadeh and K.A. Loparo. Minimizing the cardinality of an events set for supervisors of discrete-event dynamical systems. *IEEE Trans. Auto. Contr.*, 41(11):1579–1593, November 1996.

[22] D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173:143–161, 2002.

[23] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.

[24] I.N. Herstein. *Topics in Algebra*. John Wiley & Sons, Inc., 1982.

[25] P. G. Hoel, S. C. Port, and C. J. Stone. *Introduction to Probability Theory*. Houghton Mifflin Co., 1971.

[26] Hoffman. *Graph Isomorphism and Permutation Groups, LNCS 132*. Springer-Verlag, 1982.

[27] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, MA, USA, 1979.

[28] M.D.A Huth and M.D. Ryan. *Logic in Computer Science Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, UK, 2000.

[29] S. Jiang, V. Chandra, and R. Kumar. Decentralized control of discrete event systems with multiple local specilizations. In *Proc. of 2001 American Control Conference*, pages 959–964, 2001.

[30] S. Jiang and R. Kumar. Supervisory control of discrete event systems with CTL* temporal logic specifications. Preprint.

[31] S. Jiang and R. Kumar. Decentralized control of discrete event systems with specializations to local control and concurrent systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):653–660, 2000.

[32] S. Jiang, R. Kumar, and H.E. Garcia. Optimal sensor selection for discrete-event systems with partial observation. *IEEE Trans. Auto. Contr.*, 48(3):369–381, 2003.

[33] S. Khuller, G. Kortsarz, and K. Rohloff. Approximating the minimal sensor selection for supervisory control. Preprint.

[34] J.F. Knight and K.M. Passino. Decidability of a temporal logic used in discrete-event systems analysis. *International Journal of Control*, 52(6):1489–1506, 1990.

[35] D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th Symp. on the Foundations of Computer Science*, pages 254–266, 1977.

[36] O. Kupferman and M. Vardi. Verification of fair transition systems. *Chicago Journal of Theoretical Computer Science*, 1998(2):1–37, 1998.

[37] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[38] H. Lamouchi and J.G. Thistle. Effective control synthesis for DES under partial observations. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 22–28, 2000.

[39] K.-J. Lange and P. Rossmanith. The emptiness problem for intersections of regular languages. In I. Havel, editor, *Proc. of the 17th Conf. on Mathematical Foundations of Computer Science, number 629 in LNCS*, pages 346–354. Springer-Verlag, 1992.

[40] R.J. Leduc, B. Brandin, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control: Serial case. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 4116–4121, 2001.

[41] R.J. Leduc, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control: AIP example. In *39th Allerton Conf. on Comm., Contr., and Comp.*, 2001.

[42] F. Lin. Analysis and synethsis of discrete event systems using temporal logic. *Control Theory and Advanced Technologies*, 9:341–350, 1993.

[43] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

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

[45] F. Lin and W. M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Trans. Auto. Contr.*, 35(12):199–224, December 1990.

[46] F. Lin and W. M. Wonham. Verification of nonblocking in decentralized supervision. *Control-Theory and Advanced Technology*, 7(1):223–232, March 1991.

[47] P. Madhusudan and P.S. Thiagarajan. Distributed controller synthesis for local specifications. In *Proc. 28th Int. Colloquium Automata, Languages and Programming*, pages 396–407, 2001.

[48] A. Overkamp and J. van Schuppen. Maximal solutions in decentralized supervisory control. *SIAM J. Control Optimization*, 39(2):492–511, 2000.

[49] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[50] J. L. Peterson. *Petri Net: Theory and the Modeling of Systems*. Prentice Hall Inc., Englewood Cliffs, NJ, 1981.

[51] J. Prosser. *Supervisor Synthesis for Partially Observed Discrete-Event Systems*. PhD thesis, Drexel University, 1996.

[52] M. H. Queiroz and J. E. R. Cury. Modular control of composed systems. In *Proc. of 2000 American Control Conference*, 2000.

[53] P.J. Ramadge. Some tractable supervisory control problems for discrete-event systems modeled by Büchi automata. *IEEE Trans. Auto. Contr.*, 34(1):10–19, 1989.

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

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

[56] S. L. Ricker and K. Rudie. Incorporating knowledge into discrete-event control systems. *IEEE Trans. Auto. Contr.*, 45(9):1656–1668, 2000.

[57] K. Rohloff and S. Lafortune. Interacting similar discrete-event systems. Preprint.

[58] K. Rohloff and S. Lafortune. PSPACE-completeness of automata intersection decision problems with applications to supervisory control. Preprint.

[59] K. Rohloff and S. Lafortune. Symmetry reductions for a class of distributed discrete-event systems. Preprint.

[60] K. Rohloff and S. Lafortune. Advances in state estimation and controller synthesis for general decentralized control. Technical Report CGR01-11, Department of Electrical Engineering and Computer Science, University of Michigan, 2001. Available at `http://www.eecs.umich.edu/umdes`.

[61] K. Rohloff and S. Lafortune. On the computational complexity of the verification of modular discrete-event systems. In *Proc. 41st IEEE Conf. on Decision and Control*, Las Vegas, Nevada, December 2002.

[62] K. Rohloff and S. Lafortune. The control and verification of similar agents operating in a broadcast network. In *Proc. 42nd IEEE Conf. on Decision and Control*, Maui, Hawaii, December 2003.

[63] K. Rohloff and S. Lafortune. On the synthesis of safe control policies in decentralized control of discrete event systems. *IEEE Trans. Auto. Contr.*, 48(6):1064–1068, 2003.

[64] K. Rohloff and S. Lafortune. Supervisor existence for modular discrete-event systems. In *Proc. 2nd IFAC Conf. on Control Systems Design*, Bratislava, Slovakia, September 2003.

[65] K. Rohloff and J. H. van Schuppen. Approximation methods for optimal sensor selections. Preprint.

[66] K. Rohloff, T.-S. Yoo, and S. Lafortune. Deciding co-observability is PSPACE-complete. *IEEE Trans. Auto. Contr.*, 48(11):1995–1999, 2003.

[67] K. Rudie and J.C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Trans. Auto. Contr.*, 40(7):1313–1318, 1995.

[68] K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Auto. Contr.*, 37(11):1692–1708, November 1992.

[69] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal Comput. Sys. Sci.*, 4(2):177–192, 1970.

[70] H. Stark and J. W. Woods. *Probability, Random Processes and Estimation Theory for Engineers*. Prentice Hall, Upper Saddle River, NJ, second edition, 1994.

[71] S. Takai and T. Ushio. On-line decentralized supervisory control of discrete event systems. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 7–8, December 2000.

[72] J.G. Thistle and W.M. Wonham. Synthesizing processes and schedulers from temporal specifications. *International Journal of Control*, 44(4):943–976, 1986.

[73] S. Tripakis. Undecidable problems of decentralized observation and control. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 4104–4109, 2001.

[74] J. Tsitsiklis. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals and Systems*, 2:95–107, 1989.

[75] J.H. van Schuppen. Decentralized control of discrete event systems. In V.D. Blondel and A. Megretski, editors, *Sixty open problems in the mathematics of systems and control*. Princeton University Press, 2003.

[76] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[77] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin Heidelberg, 2001.

[78] Y. Willner and M. Heyman. Supervisory control of concurrent discrete event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[79] K. Wong and J. H. van Schuppen. Decentralized supervisory control of discrete event systems with communications. In *Proc. of WODES 1996, International Workshop on Discrete Event Systems*, pages 284–289. Published by IEE, London, England, August 1996.

[80] K.C. Wong and W.M. Wonham. Modular control and coordination of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 8:247–297, 1998.

[81] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete event systems. *Maths. of Control, Signals and Systems*, 1(1):13–30, 1988.

[82] T.-S. Yoo and S. Lafortune. On the computational complexity of some problems arising in partially-observed discrete-event systems. In *Proc. of the American Control Conference*, pages 307–312, 2001.

[83] T.-S. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 13(3):335–377, 2002.

[84] T.-S. Yoo and S. Lafortune. NP-completeness of sensor selection problems arising in partially-observed discrete-event systems. *IEEE Trans. Auto. Contr.*, 47(9):1495–1499, 2002.

ABSTRACT

COMPUTATIONS ON DISTRIBUTED DISCRETE-EVENT SYSTEMS

by

Kurt Rohloff

Chairperson: Professor Stéphane Lafortune

This thesis explores computational issues related to the control and verification of systems with distributed structure. The framework of supervisory control theory and discrete-event systems is used where system modules are modelled as sets of finite state automata whose behavior coordinates on the occurrence of common events. It is shown that in general many problems related to the supervision of these systems are PSPACE-complete. There are methods for solving these problems that are more efficient in memory than the current state-of-the-art methods, but there are most likely no time-efficient general solution methods that would aid in the study of such "large-scale" systems. This thesis explores methods for avoiding the computational difficulty of solving these problems.

For decentralized control situations a new state estimator is presented that accounts for past local control actions when calculating the set of estimated system

states. The new state estimator is used to develop new decentralized control protocols with a common sufficient safety condition.

It is also shown that it is difficult to approximate minimal solutions to a sensor selection problem for partial observation control situations. Heuristic methods for solving this approximation problem based on a type of edge-colored graph cutting problem are then discussed. It is also shown how to convert a type of communicating controller problem into this edge-colored graph cutting problem.

A notion of state permutation symmetry that defines an equivalence class for the distributed system states is introduced. A method is shown to reduce the complexity of verifying μ-calculus propositions for systems with state permutation symmetry. A special class of symmetric distributed systems is also shown that allows for an even greater reduction in the difficulty of testing several fundamental system properties. Control and verification problems related to both local and global specifications for these special systems are then explored.