Modular Verification and Supervisory Controller Design for Discrete-Event Systems Using Abstraction and Incremental Construction

by

Richard Charles Hill

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Mechanical Engineering)
in The University of Michigan
2008

To John, Paul, George, Ringo, and Audry

ACKNOWLEDGEMENTS

I have been very lucky to have not one, but two excellent advisors, Professor Dawn Tilbury and Professor Stéphane Lafortune. They were both very available and generous with their time, and let me set the direction of my work, while offering guidance and assistance as I needed it. Professor Tilbury's contributions to the research presented here, as well as to my overall professional development have been substantial. Her support and guidance have helped me learn how to do research and have prepared me well for the academic career that lies before me. Professor Lafortune is the person from whom I first became acquainted with the field of Discrete-Event Systems. His course and subsequent advisement have illuminated many difficult concepts and have helped me to gain a true appreciation for the field.

I would also like to acknowledge the contributions of Professor José Cury and Professor Max de Queiroz of the Federal University of Santa Catarina in Florianópolis, Brazil. My time spent in Brazil working with them led to the bulk of the work presented in Chapter 5. Professor Cury and Professor Queiroz helped to provide me a sounding board for the work, and made my wife and I feel quite at home during our time in Brazil.

Most of all, I would like to thank my wife Audry for her support and patience. She was always there to help me see the lighter side of things and to get me through the sometimes long days of my graduate studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

The aim of the work described in this dissertation is to reduce the complexity and improve the ease with which supervisory controllers are designed for discrete-event systems (DES). These goals are achieved by proposing methodologies that design controllers modularly and that employ abstraction. The details of these proposed approaches will be described and their correctness proven. The complexity reduction provided by these approaches will also be demonstrated through application to illustrative examples.

1.1 Introduction to Discrete-Event Systems and Supervisory Control

DES are dynamic systems characterized by discrete states and event-driven evolution. Care is taken to distinguish DES from digital or discrete-time systems. Whereas digital systems are continuous systems sampled at discrete intervals of time, DES are fundamentally discrete. The state of a DES could be a buffer being empty or full, a machine being idle or busy, or a transmission being in second or third gear. Furthermore, DES evolve according to events, like a part arriving at a machine or a continuous signal entering some range of values.

Figure 1.1 shows an example of a generic, isolated machine modeled as a DES using a finite state automaton [77]. The modeled states represent the machine being *Idle* (I), *Working* (W) or *Broken* (B), as opposed to perhaps more customary continuous states like the position or cutting force of a tool bit. Transitions between these discrete states are indicated by the events *start* (s), *finish* (f), *break* (b), and *repair* (r), as opposed to evolving according to time.

A DES model of a system such as a milling machine or an automobile can be

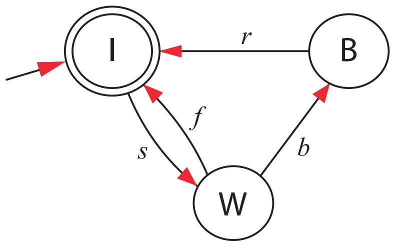

Figure 1.1: Simple DES example

surprisingly useful in answering many questions about its behavior. A DES model is appropriate for designing the high-level coordinating control for a complex system, or for answering questions that are fundamentally discrete such as, "has this machine experienced a fault?"

The development and implementation of logic controllers, and the analysis of DES in general, traditionally has been handled in a rather ad hoc manner. When designing a factory, for instance, it may seem that the control of the flow of parts through a series of machines would be very simple: supply part to machine A, when machine A finishes its operation, move part to machine B, and so forth. In this day and age, however, much more complex and flexible types of operation are often required of systems. For example, it may be desired that the same machine perform operations on several different types of parts in order to maximize utilization of the factory, or that multiple instances of the same type of part be processed by a factory in various stages of completion at the same time. In these situations, it is possible that a machine can finish multiple parts before the next machine in the process is ready, thereby causing a buffer to overflow, possibly damaging the parts or a machine. Another possible problem could be that a machine gets starved of parts that it needs, either due to a fault upstream of the machine or just due to poor design. This kind of situation could cause the whole factory to reach a deadlock, wasting precious production time and ultimately, money.

While again it may seem that these types of problems should be easily avoided just by the common sense of the designer, one might be surprised just how quickly the complexity of a system grows beyond the designer's comprehension. Problems encountered in one's everyday life, like a desktop PC crashing or a car's check engine light coming on for no apparent reason, help to indicate just how difficult the control and analysis of DES can be. It is with these problems in mind that academia has begun to look at developing tools that are capable of making theoretical guarantees about the operation of DES. For example, it is desirable to know that when a factory commences operation, it is not going reach a deadlock during its operation. Or if a sensor on a car fails, then the diagnostic system will be able to recognize the failure within a certain amount of time.

In recent years, a substantial body of work has been built up to provide a theoretical framework for answering questions about DES. References [5] and [77] provide a good introduction to the field. At its most basic level, a controller for a discrete system is just another DES model that runs in parallel with the plant. A typical control specification usually can be generated heuristically. For instance, a specification might model the desired flow of a part through a factory or the desired sequence of operations of some machine. Difficulties arise, however, when the desired control specification cannot be achieved or if multiple goals conflict with each other. With this in mind, most of the theoretical results in DES control have focused on the supervisory control framework introduced by Ramadge and Wonham [58]. Supervisory control is characterized by restricting the operation of a system to prevent undesirable events from happening, rather than commanding events to happen. In this sense, a supervisory controller works on top of a DES and disables events to keep a system *safe* and *live*. Safety means that the controlled behavior of the system is kept within a desired set of behaviors. Liveness means that the system will never reach a deadlock. *Nonblocking* is a type of liveness which indicates that the system can always reach some “goal” state. This usage of the term blocking is different than typically used in manufacturing. In traditional manufacturing terminology, blocking rather means that a machine cannot advance to the next state at that moment in time. Therefore, in manufacturing blocking is a temporal property, while in supervisory control blocking means that the system can never reach a goal state.

Another goal common to supervisory controller design is *controllability*, which guarantees that the proposed control actually can be implemented, that is, the controller does not require events to be disabled that cannot be controlled. It is also desirable that the resulting control be *optimal*. In this context, optimal means that the controlled behavior is as large as possible. In other words, the control is maximally permissive.

1.2 Motivating Problem: State-Space Explosion

Despite all the advancements that have been made in the area of DES theory, application to real-life systems has been somewhat slow. A significant hurdle to the adoption of these methods is the state-space explosion that occurs in modeling systems of the size most commonly found in industry. Consider two instances of the machine shown in Fig. 1.1 where the event labels are appended by a "1" for the first machine and a "2" for the second machine. Since each machine has three states and since the machines are completely independent, an automaton model of their concurrent operation will need $3 \times 3 = 9$ states. Figure 1.2 offers a model of the operation of the two machines where each state is given by a pair (q_1, q_2) where the first element represents the state of the first machine and the second element represents the state of the second machine. From this example one can see that the state space grows exponentially with the number of components in the system.

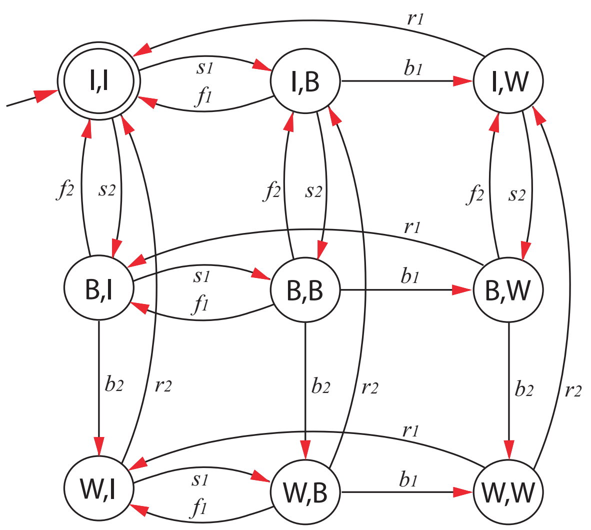

Figure 1.2: Model of concurrent operation of two machines

Consider the slightly larger Flexible Manufacturing System (FMS) example shown in Fig. 1.3 consisting of six machines and five buffers, where it is desired that the buffers never underflow or overflow [9]. For now, we will assume that each machine model has three states and each buffer model has two states. Here each buffer has the capacity to hold a single part and its states are *empty* and *full*. A more in-depth explanation of this example can be found in Section 3.4. The traditional approach to this control problem is to build a single monolithic supervisor for the entire combined system. The maximum possible size of the controller for this simple system is then 23,328 states ($3^6 \times 2^5$). Since these components are not independent, the number of reachable states of the controlled monolithic system will be less than this number. One can, however, imagine the difficulty that arises for systems of an even more realistic scale.

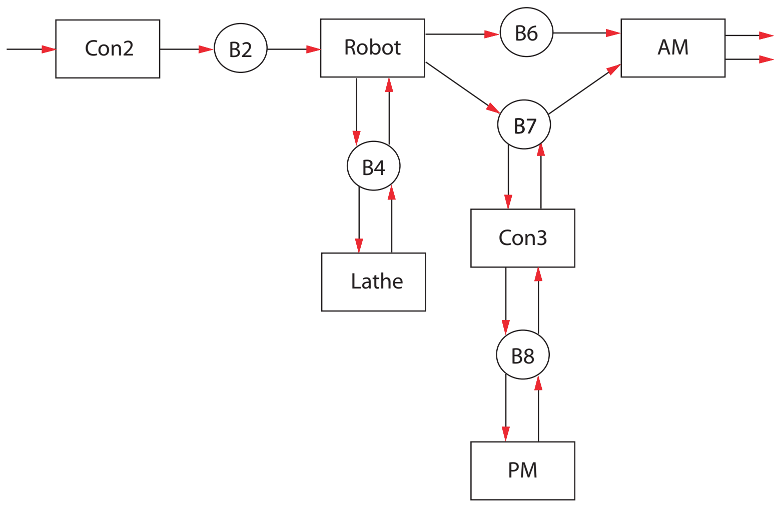

Figure 1.3: Flexible Manufacturing System (FMS) example

1.3 Prior Work

Most problems of interest in the field of DES suffer from this same problem of complexity, no matter the particular framework or approach taken. As a result, a significant portion of the work being done in the area attempts to address the state-space explosion problem.

1.3.1 Hierarchical supervisory control

One possible solution to the problem of complexity is to apply a hierarchical approach to supervisory controller design. In this approach, the supervisor is designed based on an abstracted version of the monolithic system. By abstracting away details of the system, less computation is necessary if the design or analysis can be performed on the simplified version of the system. The challenge of this approach is to be able to provide consistency between the desired control designed for the abstracted "high-level" system and its effect on the actual, unabstracted plant at the "low-level."

The concept of hierarchical supervisory control was initially proposed by [80]. This work introduced a formal notion of *hierarchical consistency* which guaranteed that the control designed on the high-level system could be implemented on the actual low-level plant. This work also introduced a notion of *output-control-consistency* that can be employed to guarantee optimality of the behavior achieved at the low-level. The theory of hierarchical supervisory control was then extended by [75]. This work demonstrated conditions on the controller structure and abstraction that provide hierarchical consistency. This work also introduced the notion of an *observer* which can be used to guarantee that the property of nonblocking is preserved between levels of the hierarchy. Related work proposes a hierarchical approach that aggregates states of an automaton model to generate a reduced order model [4] [30].

A drawback of these hierarchical approaches to control is that they require that the monolithic system be built before the abstraction is applied. Yet, due to the state-space explosion problem, the monolithic system may be too large to build in the first place. Furthermore, generating the abstraction for such a large system can be computationally prohibitive, even if the monolithic system can be generated. Another limitation of these works is that the requirements on the abstractions employed can be quite stringent and can limit the amount of model reduction that can be achieved.

1.3.2 Modular supervisory control

Another approach for reducing the complexity associated with supervisor design is to employ a modular approach to control. In modular supervisory control, a series of smaller supervisors that each meet a single component specification are constructed, rather than building a single monolithic supervisor that satisfies all specifications simultaneously. The work of [57] introduced this approach and builds each supervisor with respect to the full global plant. In contrast, the work of [8] builds a local modular supervisor with respect to the subset of the full plant that is relevant to the given specification. For the FMS example first shown in Fig. 1.3, each modular supervisor would be built to monitor a single buffer. The partitioning shown in Fig. 1.4 demonstrates the subplants that each modular supervisor would be built with respect to under the approach of [8].

The advantage of modular supervisory control is that it avoids building the full

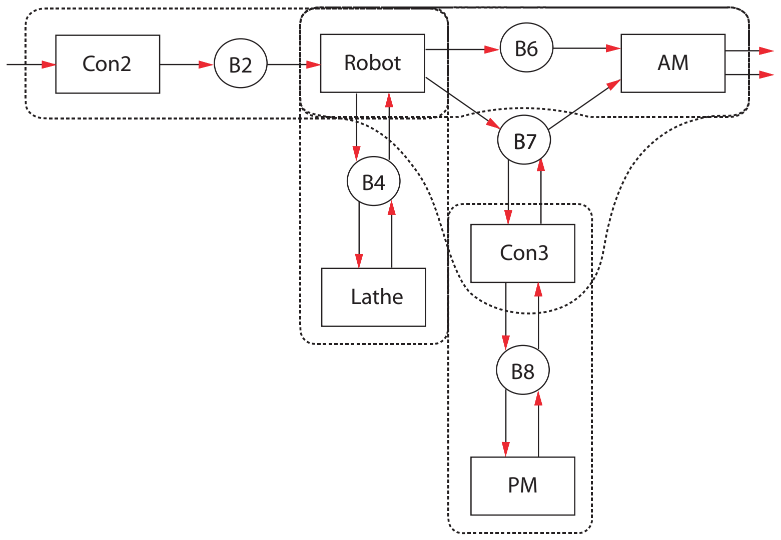

Figure 1.4: Modular control of the FMS example

monolithic system, thereby avoiding the state-space explosion problem. These modular approaches to control are able to provide safety, but do not guarantee nonblocking unless the modular supervisors are shown a priori to be *nonconflicting*. Unfortunately, verifying nonconflict typically is as computationally expensive as building the monolithic system [5]. If the modular supervisors are nonconflicting, then the behavior achieved by their conjunction is also optimal.

Work also exists for constructing modular supervisors in the case where a global specification is not given in a component-wise manner [23] [32] [33]. These works, however, do not address blocking. The work of [2] also addresses the synthesis of modular supervisors without addressing blocking.

Decentralized supervisory control is another approach that is similar to modular supervisory control. In this approach, a series of smaller supervisors are built for each component specification with respect to an abstraction of the global plant. This approach to control was introduced by [45]. As was the case with modular supervisory control, a set of decentralized supervisors will not guarantee nonblocking behavior when acting in conjunction unless they are nonconflicting.

1.3.3 Incremental and modular verification

As noted in the previous section, a major limitation of modular and decentralized approaches to supervisory control is that they do not guarantee nonblocking unless the component supervisors are nonconflicting. With this in mind, results have been generated to help reduce the complexity associated with verifying nonconflict [21] [54]. Both of these approaches essentially construct the global system incrementally using abstraction in order to reduce the complexity of verification. In particular, the work of [54] employs an abstraction with the observer property introduced by [75] that preserves *observation equivalence*, while [21] employs an abstraction that preserves *conflict equivalence*.

Other works exist that verify controllability incrementally [2] [20]. The work [20] in particular focuses on the nuances of verifying controllability when the event sets of the component languages are different from one another.

1.3.4 Conflict resolution

In recent years, research has emerged that attempts to combine aspects of the hierarchical, modular, and incremental approaches of earlier works. In particular, the work of [76] introduced an approach to control that builds modular supervisors then adds another level of control to resolve the conflict among the supervisors. Abstraction is employed to further reduce the complexity of the approach. Nonblocking control is achieved by requiring the observer property of the abstraction and optimality is achieved by additionally requiring output-control-consistency. This work provides very general results, but it is rather theoretically complex and computational tools do not exist for carrying out the proposed methods. The results of [17] address some of these limitations by generating similar results specifically for the framework of supervisory control with natural projection employed as the abstraction.

1.3.5 Structural decentralized supervisory control

When the notion of decentralized supervisory control was introduced in Section 1.3.2, it was noted that the property of nonconflict guarantees that the decentralized supervisors will not block one another if they are individually nonblocking. Other work has been developed which proposes a different set of conditions on the synchronization of two local subsystems that will also result in the controlled subsystems not blocking one another [42]. This work is referred to as structural decentralized control and has been extended by [60] [61] to employ abstraction to further reduce the computational complexity of the approach. These works do not, however, specify how to construct components that possess the required structural properties.

1.3.6 Interface-based supervisory control

The work of [38] [41] offers another approach to modular verification where controllability and nonblocking are verified by introducing interfaces that limit the interaction between components. In this approach, all events that are shared between a pair of components must be included in the event set of their interface and each of the shared events is classified as either a “request” or an “answer.” The intuition behind this is that one of the components is the “high level” that requests an action and the other component is the “low level” that answers when the requested action has been completed. An aspect of this work that distinguishes it from some other approaches is that the high level is not just an abstracted version of the low level. Rather, the two levels operate concurrently and the order of their common events is synchronized through their interface.

If the system consisting of the high level, the low level, and the interface meets a series of conditions that qualify it as being *interface consistent*, then controllability and nonblocking can be verified modularly without constructing the full monolithic system. This approach to verification is different than the other approaches introduced earlier that verify nonconflict in that it is truly modular. The other works are efficient in the way that they incrementally build a system using abstraction, but ultimately they are building the global system. Those works therefore are limited in the amount of reduction in computational complexity they are able to achieve by the abstraction they employ. In addition to reducing the complexity of verification, the use of interfaces also allows for components to be replaced in the system without reanalyzing and redesigning the entire system.

One drawback is that the structure imposed on the system limits the flexibility and optimality of the control. Additionally, since the architecture consists of only two levels and the analysis requires that the high-level module be composed with all the interfaces simultaneously, exponential growth of the state space still occurs, though generally at a reduced rate. Another limitation of this work is that it does not provide a method to generate these interfaces. The examples to which this approach has been applied have had their interfaces constructed in an ad hoc manner. Work has been done, however, that develops algorithms to construct component supervisors that are optimal with respect to a given set of interfaces [39].

Similar work that also applies a structured interface to assist with modular verification is presented in [11]. In this work, a modified version of finite state machines termed modular finite state machines are employed to model the system components. Furthermore, there is no high level or low level, interfaces just exist between each of the components that interact, that is, which share events. It is shown that this structure allows for the modular verification of some properties, though nonblocking is not one of them.

1.4 Contributions

As can be seen from the previous section, addressing the state-space explosion problem with regard to the analysis and control of DES is an active area of research. The work of this dissertation specifically proposes three new approaches that similarly aim to reduce the overall computational complexity of generating safe, nonblocking supervisory control. The inspiration for the approaches developed in this dissertation derives from many of the works mentioned in the previous section. The three approaches of this dissertation also share some interesting similarities with some of the previously mentioned works that were developed independently over the same approximate time frame.

1.4.1 Approach I: Incremental Hierarchical Supervisor Construction

The first approach to controller design provides a unique procedure for constructing a set of modular supervisors that provide safe, nonblocking control without having to construct the unabstracted monolithic system and without having to verify nonconflict. This approach will be referred to as Incremental Hierarchical Supervisor Construction (IHSC) and its details have been presented in a paper at the International Workshop on Discrete-Event Systems [26] and in a paper in the International Journal of Control [27].

The modular supervisors of this approach are constructed incrementally so that each successive supervisor is built with respect to a larger portion of the global plant. Figure 1.5 illustrates how the subsystems could be generated for the FMS example introduced earlier. Each time a new supervisor is constructed, an abstraction with the observer property is applied to abstract away those elements of the current subsystem that are not needed for any of the specifications that have not yet been addressed. This incremental approach with abstraction allows global information to be obtained without constructing the full unabstracted monolithic system. Additionally, nonconflict never has to be verified since in this approach the supervisors are built to be nonconflicting by construction.

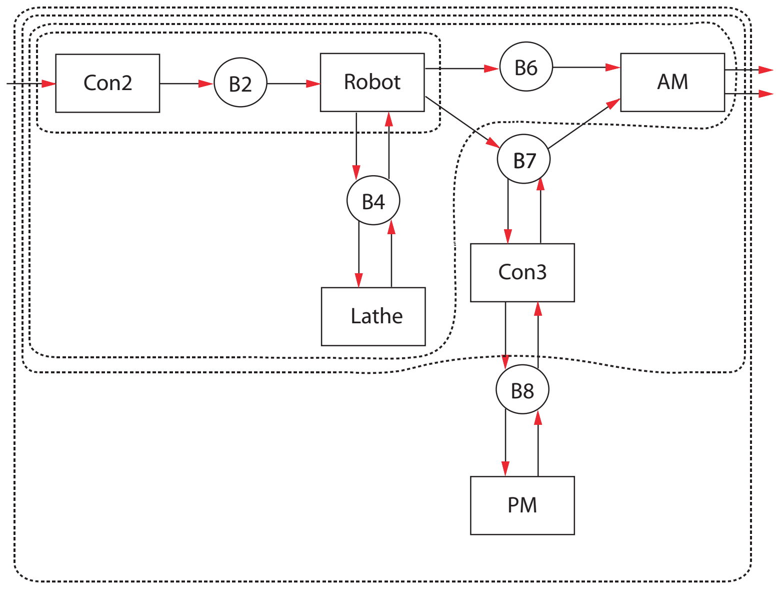

Figure 1.5: IHSC approach to partitioning the FMS example

Through application to some illustrative examples, this approach to supervisor construction is demonstrated in many cases to greatly reduce the size of the automata that must be constructed as compared to existing approaches. This reduced state size indicates that the computational complexity of constructing the supervisors has been reduced.

1.4.2 Approach II: Equivalence-Based Conflict Resolution

The second approach to control proposed in this dissertation has a similar structure to the work of [17], where modular supervisors are constructed along with an additional level of control to resolve conflict among the supervisors. The work presented here, however, is unique in that it employs a different type of abstraction. Specifically, an abstraction that preserves conflict properties rather than an observer-type abstraction is employed. In this work, requirements for achieving safe, nonblocking control are provided and an algorithm for designing the conflict-resolving control is proposed. This approach will be referred to as Equivalence-Based Conflict Resolution (EBCR) and its details have been presented in a paper at the American Control Conference [28].

The EBCR approach incrementally composes modular supervisors applying abstraction each time a new module is added. At each step if the resulting composition is blocking, then a coordinator is constructed to resolve the conflict. The abstraction employed in this approach generates a greater reduction in model state size than the observer-type abstraction employed by other works. The EBCR approach is able to employ this less restrictive abstraction because it is solely trying to capture enough information to identify conflicts among modules. In most existing work, the abstraction needs to be coarse enough that a supervisor can be designed to satisfy a given specification without blocking based on the abstracted system. A detailed analysis of these abstractions can be found in [49].

The algorithm developed for constructing the conflict-resolving control is also a new approach to solving the general state avoidance problem. The algorithm produces a new covering-based approach to control that is shown to be less restrictive than any state-feedback methodology that currently exists in the literature. Furthermore, this covering-based control can be applied to nondeterministic and partially-observed systems and is constructed with polynomial complexity.

1.4.3 Approach III: Multi-Level Interface-Based Control

In the third approach of this dissertation, the two-level hierarchical interface-based approach to supervisory control developed in [38] [39] [41] is generalized to multiple levels. The more general multiple-level architecture is advantageous because it allows a system to be partitioned into smaller pieces, thereby further improving the advantage in complexity and reconfigurability offered by an interface-based approach to control. In this dissertation, a promising methodology for constructing the interfaces required of this approach is also developed. This overall approach will be referred to as Multi-Level Interface-Based Control (MLIBC) and its details have been presented in a paper at the American Control Conference [25].

In the IHSC and EBCR approaches, the monolithic system is essentially built incrementally using abstraction. By constructing the system in this manner, global information can be obtained without constructing the full, unabstracted monolithic system. The reduction in complexity provided by these approaches is, therefore, dependent on the amount of abstraction that can be achieved at each step of the process. If insufficient abstraction can be achieved for a given system, then the state space will still grow exponentially, though at a possibly reduced rate compared to building the unabstracted monolithic system. In the MLIBC approach, the use of interfaces allows analysis and design to be performed locally, thereby avoiding the possibility of exponential growth that comes with building the global system.

One drawback of an interface-based approach is that the additional structure imposed on the system can lead the resulting control to be suboptimal. The exchange of optimality for a reduction in computational complexity is an element common to all three approaches of this dissertation. Often this trade-off is acceptable since the primary goal is to meet the controller specifications without blocking; optimality is of secondary importance. Another limitation of an interface-based approach to control is the problem of actually generating the interfaces. In this work a methodology is proposed by which the interfaces and component supervisors are synthesized simultaneously such that the interfaces end up being abstractions of the controlled modules. In applying this methodology for interface construction, the MLIBC approach begins to resemble the IHSC approach. The difference here, however, is that the abstraction does not need the observer property and hence can often achieve a greater reduction than the abstraction used in the IHSC approach. In a sense, using this approach adds extra requirements on the structure of the component supervisors in exchange for less requirements on the abstraction.

1.5 Outline

The organization for the remainder of this dissertation is as follows. Chapter 2 will provide some theoretical background and notation that is common to the whole document. Chapters 3, 4, and 5 provide the details of the IHSC, EBCR, and MLIBC approaches respectively. Chapter 6 summarizes the contributions of this dissertation and proposes some directions for future work. The Appendix includes the proofs for some results presented in the body of the dissertation. The FMS example of Fig. 1.3 will be referred to throughout this dissertation to help compare and contrast the different approaches proposed herein.

CHAPTER 2

Discrete-Event System Background

2.1 Modeling of DES

DES are commonly modeled graphically using a formalism such as automata or Petri nets. In this work DES will be modeled by possibly nondeterministic automata like the one pictured in Fig. 1.1 and represented by the five-tuple $G = (Q, \Sigma_\tau, \delta, q_0, Q_m)$, where Q is the set of states, $\Sigma_\tau = \Sigma \cup \{\tau\}$ is the set of events including the silent event τ, $\delta : Q \times \Sigma_\tau \to 2^Q$ is the state transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marked states representing successful termination of a process. In Fig. 1.1, states are shown by circles with an arrow marking the initial state and double circles indicating marked states. Transitions between states are marked by arrows with a label indicating the event name. Let Σ_τ^* be the set of all finite strings of elements of Σ_τ, including the empty string ε. The function δ is extended to $\delta : Q \times \Sigma_\tau^* \to 2^Q$ in the natural way. The notation $\delta(q, s)!$ for any $q \in Q$ and any $s \in \Sigma_\tau^*$ denotes that $\delta(q, s)$ is nonempty. The notation $\Sigma_G(q)$ will represent the set of *feasible events* of state q in automaton G, that is, those events $\sigma \in \Sigma_\tau$ for which $\delta(q, \sigma)!$. A string $s \in \Sigma_\tau^*$ will be said to be *accepted* by an automaton G if $\delta(q_0, s)!$.

The set of strings of events generated by an automaton model is called a *language* and serves as another representation of the behavior of a DES. We will assume that languages are constructed without use of the silent event τ. Let us now define the concept of a language formally. Let $P_\tau : \Sigma_\tau^* \to \Sigma^*$ be the natural projection which erases the silent event τ from strings $s \in \Sigma_\tau^*$. The general natural projection operator is defined later in equation (2.2). The *generated* and *marked languages* of G, denoted by $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$ respectively, are defined by $\mathcal{L}(G) = \{P_\tau(s) \in \Sigma^* \mid \delta(q_0, s)!\}$ and

$\mathcal{L}_m(G) = \{P_\tau(s) \in \Sigma^* \mid \delta(q_0, s) \cap Q_m \neq \emptyset\}$. For the string $s = ru \in \Sigma_\tau^*$ formed from the catenation of the strings r and u, r is called a prefix of s and is denoted $r \leq s$. The notation $\overline{K}$ represents the set of all prefixes of strings in the language K, and is referred to as the *prefix-closure* of K. The following eligibility operator will be employed to denote which events in the set Σ are enabled in the language L following the occurrence of a string $s \in \Sigma^*$, $\text{Elig}_L(s) := \{\sigma \in \Sigma \mid s\sigma \in L\}$.

An automaton is said to be *nonblocking* when from all of its reachable states a marked state can be reached. From a language point of view, this is defined as $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$. If an automaton enters a state from which it cannot reach a marked state, the automaton is said to have *blocked*.

2.2 Supervisory Control

Supervisory control of DES requires that the event set Σ_τ be partitioned into controllable and uncontrollable events, $\Sigma_\tau = \Sigma_c \dot{\cup} \Sigma_u$, where controllable events can be disabled and uncontrollable events cannot. Traditionally, the theory of supervisory control [58] has been developed for application to deterministic automata models that do not include the silent event τ and are characterized by the fact that any string can take the automaton to only a single state, that is, $\delta(q, s)$ has only a single element for a given state q and string s. Nondeterministic automata can arise due to abstraction where events are hidden by replacing them by the silent (uncontrollable) event τ. Let $\Sigma_h \subseteq \Sigma$ represent the set of events that have been hidden. Most of the work of this dissertation will assume deterministic automata. The EBCR approach to supervisory control discussed in Chapter 4 is an exception in that the abstraction it employs can lead to nondeterminism. Note that languages are not sufficient for capturing nondeterministic behavior.

We will define a supervisor, denoted $\mathcal{S}$, to be a mapping that, upon observation of a string generated by a plant G, outputs a list of events to be enabled. Since uncontrollable events must always be enabled, the mapping $\mathcal{S} : \mathcal{L}(G) \to 2^\Sigma$ implicitly includes all uncontrollable events. It is the goal of supervisory control to restrict the behavior of the uncontrolled plant to meet some given specification. This type of language-based formulation of supervisory control is very common in the DES field and implements an event-feedback law. An event-feedback law is characterized by the fact that the control action it generates depends not on the current state of the plant automaton G, but rather on the history of events that brought the system to that state. A less common formulation of supervisory control implements a state-feedback law that bases its control only on the plant's current state. This type of control will be addressed as part of the EBCR approach presented in Chapter 4.

Given a set of allowed behaviors $K \subseteq \mathcal{L}_m(G)$ and the set of uncontrollable events $\Sigma_u \subseteq \Sigma$, the existence of a language-based supervisor that can successfully restrict the operation of the plant within the behavior allowed by the specification is guaranteed by satisfaction of the following *language controllability* condition [5]:

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K} \tag{2.1}$$

If the above expression holds, it is said that the language K is language controllable with respect to the language $\mathcal{L}(G)$. Note, language controllability is fundamentally a property of a language's prefix closure. At times the above property will also be referred to as Σ_u-*controllability* in order to distinguish the set of uncontrollable events being employed.

The operation of two automata together is captured via the *synchronous composition* (parallel composition) operator, denoted by $\|$. By representing the supervisor mapping $\mathcal{S}$ as an automaton S, and the open-loop plant as a separate automaton G, the closed-loop or supervised behavior of the system can be modeled using the synchronous composition operator, $S\|G$. Throughout this dissertation we assume all automata have the same event set Σ_τ. When two automata operate concurrently they will synchronize on all events except τ, as specified by the following definition of synchronous composition.

Definition 2.1. The *synchronous composition* of two automata G_1 and G_2, where $G_1 = (Q_1, \Sigma_\tau, \delta_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_\tau, \delta_2, q_{02}, Q_{m2})$ is the automaton

$$G_1\|G_2 = (Q_1 \times Q_2, \Sigma_\tau, \delta, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$$

where the transition function $\delta : (Q_1 \times Q_2) \times \Sigma_\tau \to 2^{(Q_1 \times Q_2)}$ is defined for $q_1 \in Q_1, q_2 \in Q_2$ and $\sigma \in \Sigma_\tau$ as:
for $\sigma = \tau,\ \delta((q_1, q_2), \sigma) =$

$$\begin{cases} \delta_1(q_1,\tau) \times \{q_2\} & \text{if } \delta_1(q_1,\tau)! \text{ and } \neg\delta_2(q_2,\tau)! \\ \{q_1\} \times \delta_2(q_2,\tau) & \text{if } \neg\delta_1(q_1,\tau)! \text{ and } \delta_2(q_2,\tau)! \\ (\delta_1(q_1,\tau) \times \{q_2\}) \cup (\{q_1\} \times \delta_2(q_2,\tau)) & \text{if } \delta_1(q_1,\tau)! \text{ and } \delta_2(q_2,\tau)! \end{cases}$$

for $\sigma \in \Sigma,\ \delta((q_1, q_2), \sigma) =$

$$\delta_1(q_1, \sigma) \times \delta_2(q_2, \sigma) \qquad\qquad \text{if } \delta_1(q_1, \sigma)! \text{ and } \delta_2(q_2, \sigma)!$$

else $\delta((q_1, q_2), \sigma)$ is empty. $\diamond$

In terms of generated languages, $\mathcal{L}(G_1)\|\mathcal{L}(G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. In some cases we will assume that languages are defined over different event sets. In order to precisely define the synchronous composition in this case, we will define the *natural projection* operator, $P_i : \Sigma^* \to \Sigma_i^*$, as follows:

$$\begin{aligned} P_i(\varepsilon) &:= \varepsilon \\ P_i(e) &:= \begin{cases} e, & e \in \Sigma_i \subseteq \Sigma \\ \varepsilon, & e \notin \Sigma_i \subseteq \Sigma \end{cases} \\ P_i(se) &:= P_i(s)P_i(e), s \in \Sigma^*, e \in \Sigma \end{aligned} \tag{2.2}$$

Given a string $s \in \Sigma^*$, the projection P_i erases those events in the string that are in the alphabet Σ but not in the subset alphabet Σ_i. We can also define the inverse projection as follows:

$$P_i^{-1}(t) := \{s \in \Sigma^* : P_i(s) = t\} \tag{2.3}$$

The effect of the inverse projection P_i^{-1} is to extend the local alphabet Σ_i to Σ. In terms of automata, this is represented by adding self-loops at every state for each event in the set $(\Sigma - \Sigma_i)$. These self-looped events are in essence enabled at every state and as such do not meaningfully restrict the behavior of the system. With this in mind, we will refer to these events as being *irrelevant* and will not count them when talking about the total number of transitions in an automaton and will not draw them in figures. If an event is not self-looped at every state then it is logically referred to as being *relevant.* The notation $\Sigma(G)$ will be employed to denote the relevant event set of the automaton G. The same notation will be used to denote the relevant event sets of languages also. The following is taken from [2] and defines the notion of an irrelevant event in terms of languages.

Definition 2.2. [2] For a language $L \subseteq \Sigma^*$, an event $\sigma \in \Sigma$ is said to be irrelevant for L, if we have for all s, $t \in \Sigma^*$

$$st \in L \text{ if and only if } s\sigma t \in L.$$

Otherwise σ is called relevant for L. $\diamond$

The projection definitions given by equations (2.2) and (2.3) can be naturally extended from strings to languages and then applied to give a formal definition of the synchronous composition for languages defined over different event sets. In the following, $P_i : \Sigma^* \to \Sigma_i^*$ where $\Sigma = \cup \Sigma_i$.

$$L_1 \| L_2 \| \cdots \| L_n := P_1^{-1}(L_1) \cap P_2^{-1}(L_2) \cap \cdots \cap P_n^{-1}(L_n) \tag{2.4}$$

Also note that $\|$ is a commutative and associative operation. In addition to determining the existence of a supervisor that can achieve a given specification, it is also desirable that the controlled system be nonblocking. If and only if $K \subseteq \mathcal{L}_m(G)$ is $\mathcal{L}_m(G)$-*closed* ($K = \overline{K} \cap \mathcal{L}_m(G)$) and Σ_u-controllable with respect to the language $\mathcal{L}(G)$, then a nonblocking supervisor exists such that the supervised behavior exactly equals the admissible language $\overline{K}$, and the set of marked behaviors exactly equals K. If it is desired to implement a marking nonblocking supervisor, then only the Σ_u-controllability condition is necessary [77]. A marking supervisor can in essence unmark states of the uncontrolled plant G.

In the case that the controllability condition of equation (2.1) does not hold, and a supervisor able to provide the behavior of K does not exist, it is desirable to find the largest sublanguage of K for which such a supervisor does exist. This supremal controllable sublanguage is denoted by $\hat{K} = \sup \mathcal{C}(K, L) \subseteq K$. This is thought of as the optimal solution of the supervisory control problem in the sense that it is least restrictive.

In this dissertation the plant G and specification E are modeled by deterministic finite state automata given in the following component-wise manner:

$$G = G_1 \| \cdots \| G_n \text{ and } E = E_1 \| \cdots \| E_p$$

In terms of languages, the plant and specification are defined respectively:

$$L_m = L_{m,1} \| \cdots \| L_{m,n} \text{ and } \overline{K_{spec}} = \overline{K_{spec,1}} \| \cdots \| \overline{K_{spec,p}} \tag{2.5}$$

One way in which we will reduce the complexity of supervisor construction will be to build a series of modular supervisors, one for each specification, rather than a large monolithic supervisor that addresses all specifications simultaneously. As stated earlier, modular supervisors will not block one another if they are *nonconflicting.* A set of automata $H_1, H_2, \ldots, H_n$ is nonconflicting if the synchronous composition

$H_1 \| H_2 \| \ldots \| H_n$ is nonblocking. A set of automata being nonconflicting implies its corresponding set of marked languages $K_1, K_2, \ldots, K_n$ is also nonconflicting. Nonconflict of a set of languages is defined as $\overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_n} = \overline{K_1 \cap K_2 \cap \ldots \cap K_n}$. In words, if nonconflicting languages share a prefix, they must share a string containing that prefix.

Another way in which we will reduce complexity is through the use of abstraction. In particular, we will employ language-based abstractions based on the natural projection operation defined earlier. We will also employ equivalence-based abstractions that act to merge equivalent states of an automaton to generate a reduced-order model.

2.3 Language-Based Abstraction

One type of abstraction that will be employed in this dissertation is the language-based natural projection operation defined by equation (2.2). In particular, we will employ the natural projection abstraction as part of the IHSC approach that will be presented in Chapter 3. The idea with this approach is that if an event is not relevant to any of the remaining specifications, then we do not necessarily need to keep track of it and hence it can be considered for erasure from our models. However, it is possible that erasure of some of these events can hide some aspect of the behavior of the system that is relevant to the remaining specifications. The natural projection operation will also be employed in the construction of the interfaces employed in the MLIBC approach detailed in Chapter 5.

Since the growth of the state space comes as a result of the composition of the modules, we would like to apply our abstraction incrementally on each module before a composition is performed. A result that will be helpful to us in this regard is that the projection distributes across the synchronous composition operation if no events relevant to more than one component are erased. This fact is expressed by the following property from [63], where I represents a set of indices:

Proposition 2.3. *[63] For $i \in I$ let $L_i \subseteq \Sigma^*$ and $L := \bigcap_{j \in I} L_j$. Let $\Sigma_{com} = \cup\{\Sigma(L_i) \cap \Sigma(L_j) \mid i, j \in I \ \wedge \ i \neq j\}$. Then*

$$\Sigma_{com} \subseteq \Sigma_k \Rightarrow P_k(L) = \bigcap_{j \in I} P_k(L_j)$$

A restriction on the projection operation that will be needed later is that the projection also be an L_m-*observer*. From [73], a characterization of the L_m-*observer property* is given by the following:

Definition 2.4. [73] Let there be a natural projection $P_i : \Sigma^* \rightarrow \Sigma_i^*$ with languages $L_m \subseteq L \subseteq \Sigma^*$. Then P_i is an L_m-observer for L if:

$$(\forall s \in L)(\forall t \in \Sigma_i^*) P_i(s)t \in P_i(L_m) \Longrightarrow (\exists u \in \Sigma^*)$$
$$\text{such that } su \in L_m, \text{ and } P_i(su) = P_i(s)t \quad \diamond$$

If in the above L_m is replaced by L, then it is said that P_i simply has the *observer property* [75]. Intuitively, to have the observer property means that any branching of the system can be observed in the abstracted version of the system. The idea behind the observer property is that any control based on the abstracted model has the same intended effect on the actual plant. Strictly speaking, a projection P_i could hide a branching and still have the observer property if all paths of the branching had the same observed future. In this manner, even though there was branching the control action would be the same no matter which branch was taken. The L_m-observer property requires that not only does any branching in the marked language have the same observed future, but also that the futures have the same marking.

It is known that in the worst case the projection of a DES can lead to an exponential growth of the state space, thereby indicating the time complexity of the operation is at worst exponential. However, it has been demonstrated in [72] that a projection with the observer property is guaranteed to result in an abstracted system that is no larger than the original system. Furthermore, [72] demonstrates that under these conditions the complexity of generating the projected model is at worst polynomial in time. As far as finding a projection that possesses the observer property, [15] presents a polynomial time algorithm for finding an extension of the set of observable events for a projection such that the projection is an observer.

The observer property is maintained across synchronous composition in the same manner that the projection operation distributes across synchronous composition. The following result is taken from [54]:

Theorem 2.5. *[54] Using the definitions of Proposition 2.3, if the natural projection P_k is an $L_{m,j}$-observer for L_j for each $j \in I$ and if $\Sigma_{com} \subseteq \Sigma_k$, then P_k is an L_m-observer for $L = \bigcap_{j \in I} L_j$ where $L_m := \bigcap_{j \in I} L_{m,j}$.*

2.4 State-Based Abstraction

A second approach to abstraction that will be employed in this dissertation reduces the size of an automaton model by merging equivalent states. Many types of relations can be employed for identifying equivalent states. Specifically, in the EBCR approach of Chapter 4 we will generate reduced models that preserve conflict properties, a notion introduced in [50].

Definition 2.6. [50] Two automata H_1 and H_2 are said to be *conflict equivalent* if for any third automaton T, H_1 and T are nonconflicting if and only if H_2 and T are nonconflicting. If the automata H_1 and H_2 are conflict equivalent we write, $H_1 \simeq_{conf} H_2$. $\diamond$

Note that conflict equivalence respects the property of blocking. Also, two languages can be defined as conflict equivalent in a similar manner to Definition 2.6. Using the fact that nonconflict of two automata implies their marked languages are nonconflicting means that $H_1 \simeq_{conf} H_2$ implies $\mathcal{L}_m(H_1) \simeq_{conf} \mathcal{L}_m(H_2)$. The converse, however, does not hold since the automaton representation of a given language is not unique. Specifically, two automata can generate the same language and not be conflict equivalent. This is demonstrated by the example presented in Fig. 2.1. In the figure, automata G_1 and G_2 generate the same language, but G_1 conflicts with G_3 while G_2 does not.

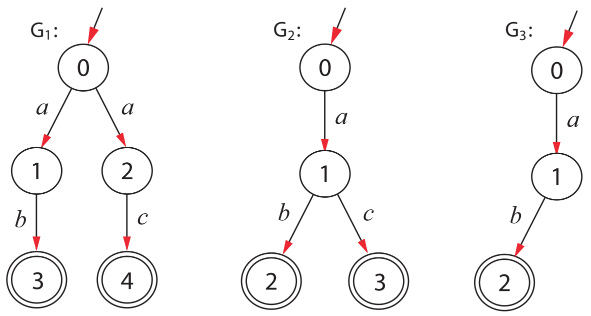

Figure 2.1: Illustrative example of nondeterminism

More generally, when nondeterministic automata models are considered, language equivalence is insufficient for capturing certain system properties. A very strong equivalence relation for states of automata is *bisimulation equivalence* [51]. In the following, we will use the notation $q \xrightarrow{\sigma} q'$ to represent that state q' is reached from state q by the event $\sigma \in \Sigma_\tau$.

Definition 2.7. Let there be two (possibly nondeterministic) automata $G_1 = (Q_1, \Sigma_\tau, \delta_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_\tau, \delta_2, q_{02}, Q_{m2})$. An equivalence relation $\sim$ on the states of these automata is said to be a *bisimulation equivalence* if for any $q_1 \in Q_1$ and $q_2 \in Q_2$, $q_1 \sim q_2$ implies that for any $\sigma \in \Sigma_\tau$:

$$\begin{aligned} &(i) \text{ if } q_1 \xrightarrow{\sigma} q_1' \text{ then } \exists q_2' \text{ such that } q_2 \xrightarrow{\sigma} q_2' \text{ and } q_1' \sim q_2'; \\ &(ii) \text{ if } q_2 \xrightarrow{\sigma} q_2' \text{ then } \exists q_1' \text{ such that } q_1 \xrightarrow{\sigma} q_1' \text{ and } q_1' \sim q_2'; \\ &(iii)\ q_1 \in Q_{m1} \text{ if and only if } q_2 \in Q_{m2}. \qquad \diamond \end{aligned}$$

Bisimulation equivalence of two states can be thought of as the two states having the same future behavior (including the silent event τ). Two automata are said to be bisimulation equivalent if their initial states are bisimulation equivalent $q_{01} \sim q_{02}$. If two automata are bisimulation equivalent, most properties of interest are consistent between the two, including blocking.

Another equivalence relation called *weak bisimulation* or *observation equivalence* [51] exists where states are considered equivalent if they have the same "observed" futures. That is, their futures must be the same when the silent event τ is projected away. The notation $q \overset{\sigma}{\Rightarrow} q'$ will be employed to denote that there exists a string $s \in \Sigma_\tau^*$ such that $q \xrightarrow{s} q'$ and $P_\tau(s) = \sigma$. This concept of observation equivalence is similar to the notion of the *observer property* employed in [17] [26] [54] [76]. Refer to [49] for a more detailed examination of the relationship between conflict equivalence, observation equivalence, and projections with the observer property.

Conflict-equivalent abstraction in general provides a greater reduction in the state size of a model than either an observation-equivalent abstraction or a projection with the observer property [49]. A drawback of a conflict-equivalent abstraction is that it is not as straightforward to implement; it is implemented via heuristics and a select set of rules [19] [21]. Also, a unique minimal reduction does not exist in general.

CHAPTER 3

Incremental Hierarchical Supervisor Construction

The Incremental Hierarchical Supervisor Construction (IHSC) approach proposed in this chapter addresses the problem of complexity in supervisory control by adopting some of the elements of existing hierarchical and modular approaches, while at the same time avoiding some of their weaknesses. Specifically, it is desired to build a set of nonblocking modular supervisors while avoiding construction of the unabstracted monolithic system, as is required by traditional hierarchical approaches, and without having to verify that the component supervisors are nonconflicting, as is required by traditional modular approaches to control.

With the IHSC approach, construction of the unabstracted monolithic system will be avoided by building the global system incrementally using abstraction. Verification of nonconflict will be avoided by taking advantage of special cases where nonconflict occurs by construction. Specifically, languages are nonconflicting if they are disjoint or if they are subsets of one another. We will define disjoint to mean that the languages do not share any relevant events. It will be assumed in this work that the global plant and specification are given modularly as in equation (2.5), and that the specification languages are prefix-closed. It will be further assumed that each of the component plants have been organized such that they do not share relevant events. This situation is referred to as a *product system* [58]. The automata models of this chapter will also be deterministic, that is, they do not include the silent event τ and any given string can only take an automaton to a single state.

For the Flexible Manufacturing System (FMS) introduced in Chapter 1, a possible partition generated by our proposed approach is shown in Fig. 3.1. The systems in each of the two inner dashed blocks are supervised by their own supervisor and the outer dashed blocks represent the systems supervised by the third, fourth, and fifth supervisors. Nonconflict of the set of the supervisor languages is guaranteed since the two modules on the first level of the hierarchy are disjoint, and the behavior allowed by the supervisors on subsequent levels are each a subset of the behavior allowed by the supervisors on the previous levels. The problem one might recognize is that the fifth supervisor acts on the full system, which is counter to the goal of a modular approach. The solution is to perform an abstraction on the supervised systems in the inner blocks before moving up a level of the hierarchy to design the remaining supervisors. If an aspect of a lower level component is not relevant to any of the remaining modular specifications, then it can be abstracted away. In this way, abstractions are performed incrementally, rather than on the entire system at once. Also, a greater level of abstraction can be achieved than if the abstraction were performed on the monolithic system at the very end. For instance, even though the highest level supervisor is built with respect to the full plant, a significant amount of reduction over the abstracted monolithic solution is achievable in most cases. This is because the last modular supervisor is only trying to guarantee the additional satisfaction of the last specification, rather than all of the specifications simultaneously.

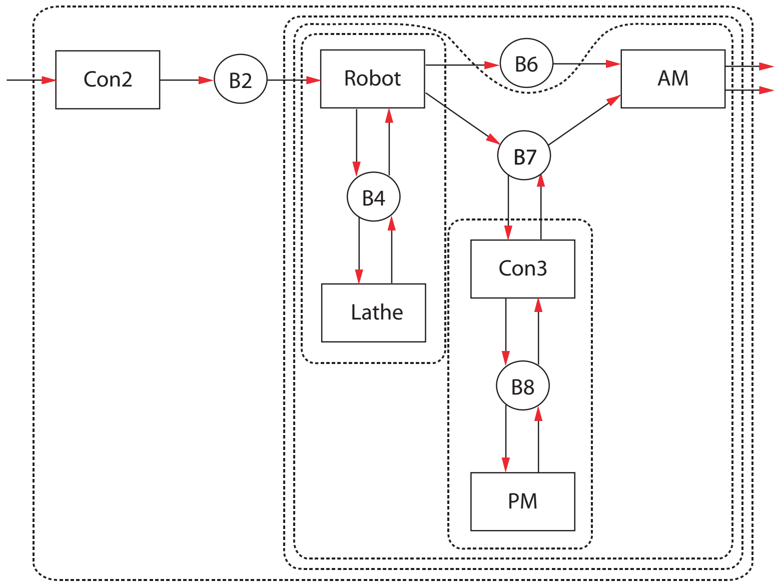

Figure 3.1: IHSC approach to partitioning the FMS example

The outline of the rest of this chapter is as follows. Section 3.1 uses a small example to demonstrate the IHSC approach to generating modular supervisors. Section 3.2 demonstrates that when no abstraction is employed, the resulting supervisors provide maximally permissive control. Section 3.3 shows that when abstraction is employed, the conjunction of the resulting supervisors will be controllable and nonblocking, though not optimal. Section 3.4 discusses the strengths and weaknesses of the approach of this chapter and offers some heuristics for its implementation through application to two moderately large examples. Section 3.5 summarizes the contributions of this chapter and outlines some areas for further investigation.

3.1 Supervisor Construction Algorithm

In this section we will outline the procedure by which the set of modular supervisors is generated through application to a portion of the FMS example introduced in Chapter 1 and shown here in Fig. 3.2. The plant consists of a *Robot*, a Conveyor (*Con3*), a Painting Machine (*PM*), and an Assembly Machine (*AM*). The two buffers connecting the various machines, *B7* and *B8*, serve as the specifications for the system where it is desired that the buffers do not underflow or overflow.

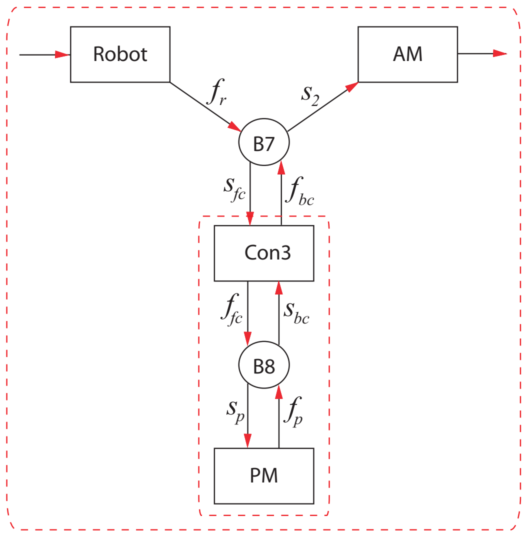

Figure 3.2: Portion of the FMS example

The automata models for the different machines are shown in Fig. 3.3. Note that each of the starting events, s_i, are controllable, and each of the finishing events, f_i, are uncontrollable. Also recall that all automata have the same alphabet, though irrelevant events are not pictured in the figures. The finite state automata models for the buffers are shown in Fig. 3.4. These buffer models are marked for states for which the buffer is full because our approach requires that the specification languages be prefix-closed, that is, the supervisor cannot change the marking of the uncontrolled plant. If it is necessary to ensure that the system reaches a state where the buffers are empty, another specification could be added.

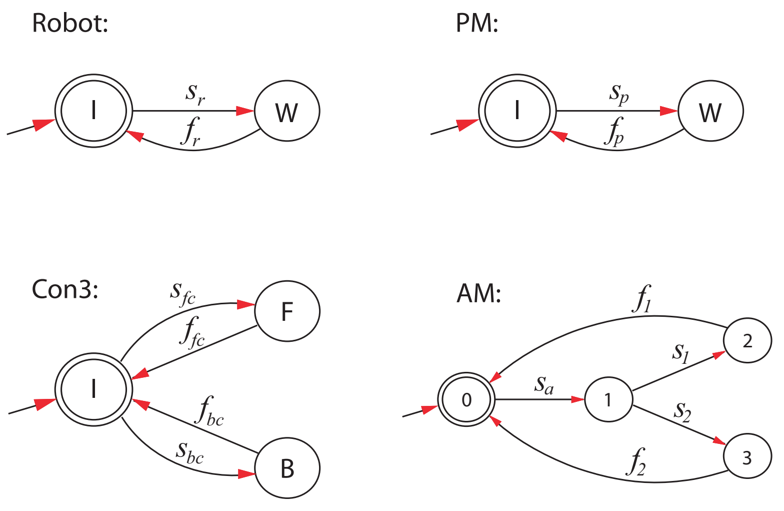

Figure 3.3: Automata models of each machine in FMS portion

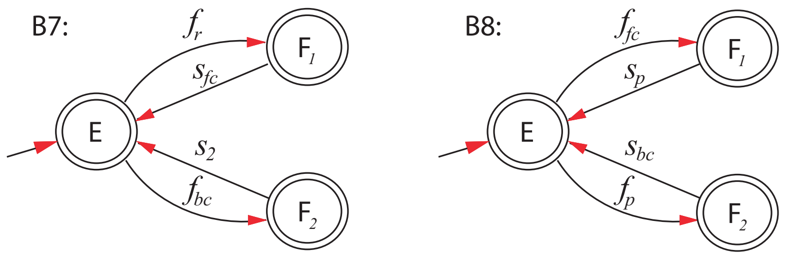

Figure 3.4: Automata models of each buffer in FMS portion

We will first present the procedure, and then go through the example. Each series of languages will be indexed sequentially. This procedure and the proofs to follow will also assume only one specification is addressed per level of the hierarchy in order to simplify the notation.

Algorithm 3.1. *Incremental Hierarchical Supervisor Construction*
Input: $L_{m,1}, \ldots, L_{m,n}, K_{spec,1}, \ldots, K_{spec,p}$
Step 1: Choose an initial specification - Pick a specification $K_{spec,1}$ and group it with all plant submodules $\{L_1, L_2, \ldots, L_{i_1}\}$ with which it shares a relevant event. We will assume that all plant submodules have already been organized such that they do not share relevant events with each other; all interaction takes place through the specifications. Composing system components so that the plant submodules are

disjoint from one another can lead to exponential growth, though in most cases on a much smaller scale than building the monolithic system. Figure 3.5 illustrates an example relationship between the relevant event sets of the various plant and specification components in the context of the global event set Σ.

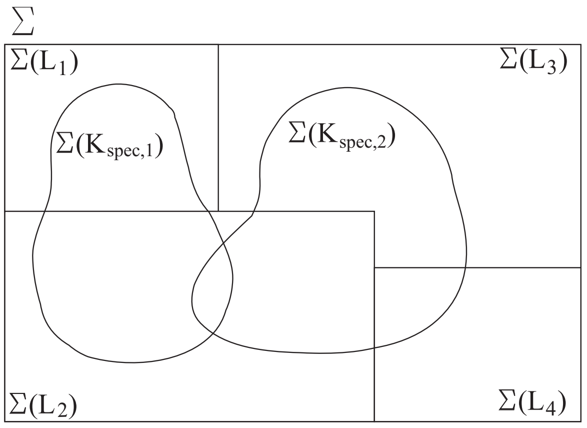

Figure 3.5: Example relationship between relevant event sets

Step 2: Perform abstraction - Perform a projection on the languages generated by the plant submodules from the previous step. The subscript a is used to represent these abstracted languages; specifically, let $L_{i,a} = P_1(L_i)$ and $L_{m,i,a} = P_1(L_{m,i})$ where $i \in \{1, 2, \ldots, i_1\}$. Only those events that are not relevant to any specifications on the current or remaining levels of the hierarchy may be considered for erasure. Also, the respective $L_{m,i}$-observer property must be maintained for each of the subplants on the current or remaining levels of hierarchy.

On the first level of the hierarchy, P_1 is therefore defined to maintain the observer property with respect to all $L_{m,i}$, that is, for all $i \in \{1, \ldots, n\}$. This means that all events that are relevant to only a single component and that do not violate the respective observer properties will be erased by P_1. In practice, however, in this first step we will actually only consider those languages with indices in $\{1, 2, \ldots, i_1\}$. Those events that do not qualify for erasure make up the set Σ_1 and the associated projection is defined $P_1 : \Sigma^* \to \Sigma_1^*$. Since we have not examined those languages outside the set $\{L_1, L_2, \ldots, L_{i_1}\}$, we do not yet know the exact composition of Σ_1.
Step 3: Compose subplant members - Perform a synchronous composition of all abstracted plant submodules on this level of hierarchy.

$$L'_{1,a} = L_{1,a} \| L_{2,a} \| \cdots \| L_{i_1,a}$$

Since no relevant events are shared among the individual L_i, the projection P_1 distributes across the synchronous composition by Proposition 2.3. Therefore, $L'_{1,a} = \|_{i \in I_1} P_1(L_i) = P_1(\|_{i \in I_1} L_i)$, where $I_1 = \{1, 2, \ldots, i_1\}$. Furthermore, $L'_{1,a} = P_1(L'_1)$ where $L'_1 = \|_{i \in I_1} L_i$. The projection P_1 also possesses the $L'_{m,1}$-observer property by Theorem 2.5, where $L'_{m,1} = \|_{i \in I_1} L_{m,i}$. Both L'_1 and $L'_{m,1}$ have alphabets equal to the global event set Σ and all abstracted languages on this level of the hierarchy have alphabets equal to Σ_1.

Recall, there are some events in Σ_1 which are as of yet unknown. These events, however, are not relevant to L'_1, and as such do not cause a violation of the $L'_{m,1}$-observer property if they are projected away.

The same projection P_1 used in generating the abstracted modular plant $L'_{1,a}$, is also used to generate the corresponding abstracted specification, $K_{spec,1,a} = P_1(K_{spec,1})$. In our approach, events are only considered for abstraction if they are not relevant to the current or remaining specifications, therefore, P_1 erases only irrelevant events from $K_{spec,1}$. A representation of the event set Σ_1 can be seen in Fig. 3.6.

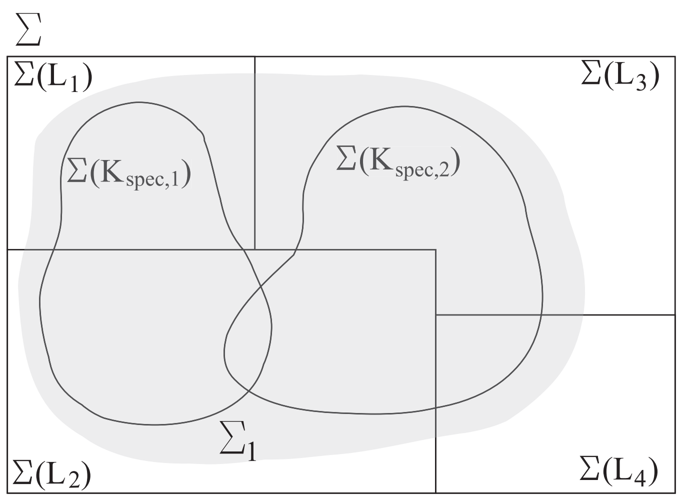

Figure 3.6: Example relationship between relevant event sets; Σ_1 is the shaded region

Step 4: Build supervisor for abstracted module - Following the traditional techniques for supervisor construction, build a nonblocking supervisor for the abstracted plant/specification pair. The component allowable language is defined as $K'_1 =$

$\overline{K_{spec,1}} \cap L'_{m,1}$ and its corresponding abstraction is denoted $K'_{1,a}$.

$$\begin{gathered} \hat{K}'_{1,a} = \sup \mathcal{C}(K'_{1,a}, L'_{1,a}) \\ \text{where } K'_{1,a} = \overline{K_{spec,1,a}} \cap L'_{m,1,a} \end{gathered} \tag{3.1}$$

In the expression for $K'_{1,a}$, the projection distributes across the intersection because P_1 does not erase any relevant events from $\overline{K_{spec,1}}$.

In the worst case, the resulting supervisor can be the empty set. However, this supervisor is not guaranteed to be optimal because of the abstraction. Therefore, it is possible that a less restrictive supervisor can be found by erasing fewer events (by making Σ_1 larger). A discussion of how to choose which events to retain can be found at the end of Section 3.3.

Step 5: Lift abstracted supervisor to the global level - In order to apply a supervisor generated in Step 4 to the global plant, a default control is implemented that enables all events that had been previously projected away. This is accomplished by the inverse projection operation and is represented by $P_1^{-1}(\overline{\hat{K}'_{1,a}})$.

Step 6: Move up to the next level of the hierarchy - In order to address those specifications for which a modular supervisor has not yet been built, we now move up to the next level of the hierarchy and repeat Steps 1-5. The language representing the closed-loop behavior of the subsystem from the previous level $\hat{K}'_{1,a}$ becomes a subplant on this level. For example, the allowable language K'_2 for specification $K_{spec,2}$ will be designed with respect to a "plant" L'_2, that is composed of $\hat{K}'_{1,a}$ from the first level as well as the subplants L_i that were not employed in the first level and that share relevant events with $K_{spec,2}$. Even though on this level $\hat{K}'_{1,a}$ is treated as a subplant, we will still refer to it in terms of its original name which is more consistent with its role on the first level as an allowable language.

Before the synchronous composition is performed, it is again necessary to perform an abstraction $P_2 : \Sigma^* \rightarrow \Sigma_2^*$ on each of the submodules such that the observer properties are maintained with respect to any of the plant components on the current or remaining levels of hierarchy. In practice, however, we will again only consider those languages $\{L_{i_1+1}, \ldots, L_{i_2}\}$ on this level of the hierarchy. Therefore, we end up with a set consisting of abstracted languages $L_{i,a} = P_2(L_i)$ where $i \in \{i_1+1, \ldots, i_2\}$.

$$\begin{gathered} L'_{2,a} = P_2(\overline{\hat{K}'_{1,a}}) \| L''_{2,a} \\ \text{where } L''_{2,a} = L_{i_1+1,a} \| \cdots \| L_{i_2,a} \end{gathered} \tag{3.2}$$

Building our "plant" in this manner is useful in that it causes the new supervisor to be nonconflicting with the supervisor from the previous level. Figure 3.7 redraws Fig. 3.6 in terms of the newly defined L''_j languages.

The subplants $\{L_{i_1+1}, \ldots, L_{i_2}\}$ that go into generating $L''_{2,a}$ do not share any relevant events with the specification from the first level. Also, by definition the projection P_2 does not erase events relevant to specifications on the current or higher levels of the hierarchy. Therefore, the only relevant events P_2 erases from $\{L_{i_1+1}, \ldots, L_{i_2}\}$ are not relevant to any component specifications. More specifically, the relevant events P_2 erases from $\{L_{i_1+1}, \ldots, L_{i_2}\}$ are the same events erased by P_1, we just did not know it at the time.

The relevant events erased from the component $\hat{K}'_{1,a}$, however, can be relevant to the specification from the first level if they are not relevant to any of the remaining specifications. Furthermore, events that are relevant only to $\{L_1, L_2, \ldots, L_{i_1}\}$ that violated the respective observer property on the first level can also be reconsidered for erasure. As such, each time we move up a level of the hierarchy more abstraction is possible, that is, $\Sigma_2 \subseteq \Sigma_1$. Figure 3.7 shows how the set Σ_2 fits into the larger picture.

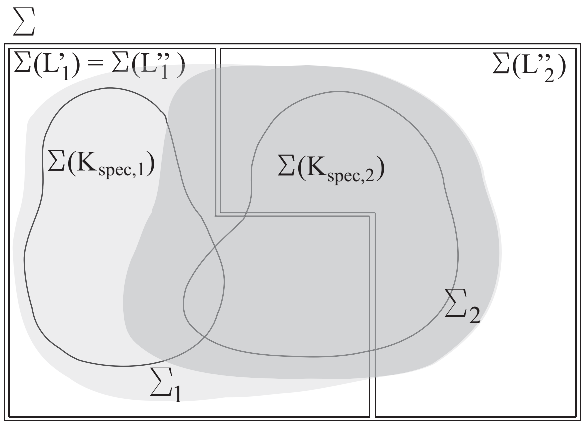

Figure 3.7: Example relationship between relevant event sets; Σ_2 is the darkly shaded region

The behavior allowed by the supervisor on this level of the hierarchy is determined in the same manner as equation (3.1), $\hat{K}'_{2,a} = \sup \mathcal{C}(K'_{2,a}, L'_{2,a})$.

The logic outlined above tells us that the set of relevant events erased from $\{L_{i_1+1}, \ldots, L_{i_2}\}$ by P_2 is disjoint from the set of relevant events erased from $P_1^{-1}(\overline{\hat{K}'_{1,a}})$. This disjointness will be used in the proofs of Section 3.3 to define the projection P_2 as occurring in two stages. Additionally, since all the components making up L'_2 have disjoint sets of relevant events, the projection still distributes by Proposition 2.3. Therefore, $L'_{2,a} = P_2(L'_2)$ where:

$$\begin{gathered} L'_2 = \overline{\hat{K}'_{1,a}} \| L''_2 = P_1^{-1}(\overline{\hat{K}'_{1,a}}) \cap L''_2 \\ \text{where } L''_2 = L_{i_1+1} \| \cdots \| L_{i_2} \end{gathered} \tag{3.3}$$

Furthermore, P_2 possesses the $L'_{m,2}$-observer property by Theorem 2.5, where $L'_{m,2} = P_1^{-1}(\hat{K}'_{1,a}) \cap L''_{m,2}$ and $L''_{m,2} = L_{m,i_1+1} \| \cdots \| L_{m,i_2}$.
Step 7: Repeat until finished - Continue to repeat this process until there are no more specifications left.
Output: $\hat{K}'_{1,a}, \ldots, \hat{K}'_{1,p}$ $\diamond$

Now we illustrate the procedure through a brief FMS example.

Example 3.2.
Step 1: Choose an initial specification - We will choose $L_{B8} = \mathcal{L}(\mathit{B8})$ to be our first specification where the subplants that it shares relevant events with are $L_{C3} = \mathcal{L}(\mathit{Con3})$ and $L_{PM} = \mathcal{L}(\mathit{PM})$. Note, the order in which we address specifications and the controlled behavior that results is not unique.
Step 2: Perform abstraction - All events relevant to the plant submodules comprising the first level of the hierarchy, *Con3* and *PM*, are also relevant to the current or remaining specifications, thus $L_{C3,a} = P_1(L_{C3})$ and $L_{PM,a} = P_1(L_{PM})$ where P_1 erases only irrelevant events from the languages. In terms of the automata, this means only events that are self-looped at every state of *Con3* and *PM* are projected away. In practice, irrelevant events do not need to be projected or added. For the purposes of the proofs to follow, however, it is useful to have sets of languages over the same alphabet.
Step 3: Compose subplant members - Composing the individual subplants gives us the plant and specification for our first module, $L'_{1,a} = L_{C3,a} \| L_{PM,a}$ and $\overline{K_{spec,1,a}} = P_1(L_{B8})$ respectively.

Step 4: Build supervisor for abstracted module - The abstracted allowable language for the first specification is $K'_{1,a} = L_{B8,a} \cap L_{m,C3,a} \cap L_{m,PM,a}$. It turns out $K'_{1,a}$ is not Σ_u-controllable and hence the supremal controllable sublanguage must be taken. The resulting language is $\hat{K}'_{1,a} = \sup \mathcal{C}(K'_{1,a}, L'_{1,a})$.
Step 5: Lift abstracted supervisor to the global level - This is accomplished by the inverse projection operation. An automaton which generates $\hat{K}'_{1,a}$ and represents the first modular supervisor for our example is shown in Fig. 3.8. Since we do not picture irrelevant events, the events added by the inverse projection are not shown in Fig. 3.8.

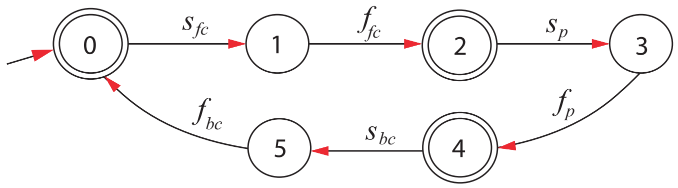

Figure 3.8: Automaton representing the supervisor which marks the language $\hat{K}'_{1,a}$

Step 6: Move up to the next level of the hierarchy - For our example, we move up a level of hierarchy to build a supervisor for the second specification $L_{B7} = \mathcal{L}(B7)$. Our new grouping consists of the remaining machines, $L_R = \mathcal{L}(Robot)$ and $L_{AM} = \mathcal{L}(AM)$, as well as the supervised language from the first level $\hat{K}'_{1,a}$. At this point, those events not relevant to the remaining specification are s_r, s_a, s_1, f_1, f_2, s_p, f_p, f_{fc}, and s_{bc}. It is interesting to note that the last four events in the above list were relevant to the first specification and as such could not be considered for erasure on the first level of the hierarchy. We must first, however, check to see if abstraction of any of these events will cause a violation of the individual $L_{m,i}$-observer properties that we need to guarantee nonblocking. Examination of Fig. 3.3 and Fig. 3.8 will help us to do this.

Specifically, abstraction of the event s_r causes a problem because it represents a transition from a marked state to an unmarked state without an unobservable string of transitions back to a marked state. In other words, if s_r is unobservable then $P_2(s_r) = \varepsilon$ is an element of $P_2(L_{m,R})$, but there is no string u such that $s_r u \in L_{m,R}$ and $P_2(s_r u) = \varepsilon$. Applying similar logic, event s_{bc} cannot be abstracted either. Therefore, $\Sigma_2 = \{s_{fc},\ f_{bc},\ s_{bc},\ s_r,\ f_r,\ s_2\}$. We also now know, $\Sigma_1 = \Sigma_2 \cup \{s_p,\ f_p,\ f_{fc}\}$. The resulting reduction of machine AM has a single state and generates the language $L_{AM,a} = P_2(L_{AM})$. An automaton which generates the reduction $P_2(\hat{K}'_{1,a})$ has 3 states and 3 transitions. The *Robot* subplant and specification *B7* are not reduced by the projection P_2. That is, $L_{R,a} = P_2(L_R)$ and $L_{B7,a} = P_2(L_{B7})$ where P_2 erases only irrelevant events.

Therefore, the plant for the second specification is $L'_{2,a} = P_2(\overline{\hat{K}'_{1,a}}) \cap L_{R,a} \cap L_{AM,a}$. Following Step 4, the abstracted allowable language is then $K'_{2,a} = L_{B7,a} \cap L'_{m,2,a}$. This language again fails to be Σ_u-controllable. Therefore taking the supremal controllable sublanguage, the new allowable language for the second specification is $\hat{K}'_{2,a} = \sup \mathcal{C}(K'_{2,a}, L'_{2,a})$. A minimal automaton generator for this language has 8 states and 8 transitions. The final step again is to lift this abstraction up to the global level. Specifically, the events s_a, s_1, f_1, f_2, s_p, f_p, and f_{fc} must be added.
Step 7: Repeat until finished - Since there are no specifications left, we are done. $\diamond$

For the purposes of comparison, the monolithic solution of our example generates a supervisor whose automaton representation consists of 68 states and 157 transitions, while the two modular supervisors are significantly smaller. Specifically, a minimal automaton which generates $\hat{K}'_{1,a}$ has 6 states and 6 transitions and a minimal automaton which generates $\hat{K}'_{2,a}$ has 8 states and 8 transitions (irrelevant events are not considered when counting transitions). Even though the highest level supervisor is built in essence with respect to the full plant, it will almost always be significantly smaller than the monolithic supervisor since it is built to address only those strings relevant to the last specification, rather than the conjunction of all the specifications. It will be shown in Section 3.4 the improvement in complexity can be more dramatic in systems larger than the simple one used here for illustrative purposes.

It is interesting to note that if the modular supervisor for *B7* had been built first and the supervisor for *B8* been built second, the modular supervisors would have had 8 and 10 states and 11 and 13 transitions respectively. Some guidelines for how to best choose the ordering in which the specifications are addressed will be discussed in Section 3.4.

In this case, it turns out that the behavior allowed by the two modular supervisors is identical to the behavior allowed by the single monolithic supervisor and hence is optimal. This will not be the case in general since each of the modular supervisors were designed with respect to an abstracted plant. Some discussion of optimality can be found at the end of Section 3.3.

The procedure presented above assumed one specification is addressed per level of hierarchy. This approach, however, is still valid if multiple specifications are addressed per level, though it will be demonstrated later that in most cases it is computationally advantageous to address only a single specification. If multiple specifications are addressed within a level, then they must be chosen to address a disjoint set of subplants. The disjointness is needed to provide nonconflict.

3.2 Optimal Control Without Abstraction

We will now consider the modular supervisors constructed by the algorithm of the previous section without the use of any abstraction. Specifically, it will be shown that the conjunction of the modular supervisors will provide the same behavior as the monolithic supervisor. This implies that not only will the conjunction of modular supervisors satisfy the given specifications in a nonblocking manner, but further, that the provided supervision will be optimal in the sense of being least restrictive. In the next section these arguments will be reconsidered with the addition of abstraction.

Before we get to the main result of this section, we must present some propositions. The first proposition is a result from [2] which is useful in showing that if an allowable language is Σ_u-controllable with respect to a subset of plant subsystems, it is Σ_u-controllable with respect to the full plant.

Proposition 3.3. *[2] Let K, $L \subseteq L' \subseteq \Sigma^*$ be languages. Also let $\Sigma_u \subseteq \Sigma$. If K is Σ_u-controllable with respect to L', then K is Σ_u-controllable with respect to L.*

This next proposition shows that the intersection of two nonconflicting Σ_u-controllable languages is itself Σ_u-controllable. The result is in general well-known and can be found in [57].

Proposition 3.4. *[57] Let K_1, K_2, $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. Also let $\Sigma_u \subseteq \Sigma$. If K_1 and K_2 are nonconflicting and Σ_u-controllable with respect to L, then K is Σ_u-controllable with respect to L.*

Since we are building each successive level of allowable languages with respect to an uncontrolled plant intersected with a controlled language from the previous level, this next proposition is used to show controllability of the intersection of languages from successive levels of our hierarchy. The proof of this result follows from logic presented in [2].

Proposition 3.5. *Let K_1, K_2, $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. Also let $\Sigma_u \subseteq \Sigma$ and K_1 and K_2 be nonconflicting. If K_2 is Σ_u-controllable with respect to $\overline{K_1} \cap L$, and K_1 is Σ_u-controllable with respect to L, then K is Σ_u-controllable with respect to L.*

Proof. See proof in Appendix. □

This next proposition is used to demonstrate L_m-closure of the intersection of languages from successive levels of our hierarchy.

Proposition 3.6. *Let K_1, K_2, $L_{m,1}$, and $L_{m,2} \subseteq \Sigma^*$ be languages and let $K_2 \subseteq K_1$ and $L_m = L_{m,1} \cap L_{m,2}$. If K_2 is closed with respect to $K_1 \cap L_{m,2}$ and K_1 is $L_{m,1}$-closed, then K_2 is L_m-closed.*

Proof.

$$\begin{aligned} K_2 &= \overline{K_2} \cap (K_1 \cap L_{m,2}) \\ &= \overline{K_2} \cap ((\overline{K_1} \cap L_{m,1}) \cap L_{m,2}) \\ &= \overline{K_2} \cap (L_{m,1} \cap L_{m,2}) = \overline{K_2} \cap L_m \ \diamond \end{aligned}$$

□

This next proposition is a result from [2] that will prove useful for showing optimality for our incremental approach to supervisor construction when no abstraction is employed. In the following, the requirement that $(\Sigma(K) \cup \Sigma(L_1)) \cap \Sigma(L_2) \cap \Sigma_u = \emptyset$ means that neither K nor L_1 can share any relevant uncontrollable events with L_2.

Proposition 3.7. *[2] Let K, L_1, $L_2 \subseteq \Sigma^*$ be languages and let $L = L_1 \cap L_2$. Furthermore, let:*

$$\begin{aligned} \hat{K} &= \sup \mathcal{C}(K, L) \\ \hat{K}_1 &= \sup \mathcal{C}(K, L_1) \end{aligned}$$

If $(\Sigma(K) \cup \Sigma(L_1)) \cap \Sigma(L_2) \cap \Sigma_u = \emptyset$, then $\overline{\hat{K}_1} \cap L = \overline{\hat{K}} \cap L$.

It is also desirable to extend the result of Proposition 3.7 to find an expression in terms of the nonprefix-closed languages, $\hat{K}$ and $\hat{K}_1$. This is accomplished in the following corollary.

Corollary 3.8. *Under the conditions of Proposition 3.7 with the additional requirements that $K \subseteq L_{m,1}$ and K be $L_{m,1}$-closed, $\hat{K}_1 \cap L_{m,2} = \hat{K} \cap L_{m,2}$.*

Proof. The result of Proposition 3.7 is $\overline{\hat{K}_1} \cap L = \overline{\hat{K}} \cap L$. Intersecting both sides with L_m gives the result $\overline{\hat{K}_1} \cap L_m = \overline{\hat{K}} \cap L_m$ where $L_m = L_{m,1} \cap L_{m,2}$. Since $K \subseteq L_{m,1}$, the $\sup \mathcal{C}$ operation preserves $L_{m,1}$-closure. Therefore, $\hat{K}$ and $\hat{K}_1$ are $L_{m,1}$-closed and hence $\hat{K}_1 \cap L_{m,2} = \hat{K} \cap L_{m,2}$. □

With all of these propositions in place, we can now generate a lemma that will allow us to show the main result of this section. Specifically, the lemma will allow us to show that an incrementally built language can be supremal. The setup for Lemma 3.9 is that there are two specifications that can interact through a common subplant. Instead of building a monolithic supervisor directly, we build a modular supervisor for one specification then build the second modular supervisor with respect to a "plant" made up of the portion of the monolithic plant not addressed yet and the modular supervised language just constructed. This lemma addresses the step of our approach where we move up a level of the hierarchy.

Lemma 3.9. *Let K_1, K_2, L_1, $L_2 \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$ and $L = L_1 \cap L_2$. Let each $K_i \subseteq L_{m,i}$. Also let K_1 be $L_{m,1}$-closed and:*

$$\begin{aligned}
\hat{K}_1 &= \sup \mathcal{C}(K_1, L_1) \\
K_2' &= \hat{K}_1 \cap K_2 \\
L_{m,2}' &= \hat{K}_1 \cap L_{m,2} \\
L_2' &= \overline{L_{m,2}'}
\end{aligned}$$

If $(\Sigma(K_1) \cup \Sigma(L_1)) \cap \Sigma(L_2) = \emptyset$ then $L_2' = \overline{\hat{K}_1} \cap L_2$ and:

$$\sup \mathcal{C}(K_2', L_2') = \sup \mathcal{C}(\hat{K}_1 \cap K_2, \overline{\hat{K}_1} \cap L_2) = \sup \mathcal{C}(K_1 \cap K_2, L_1 \cap L_2) = \sup \mathcal{C}(K, L)$$

Proof. ($\subseteq$) First we will show $\sup \mathcal{C}(K_2', L_2') \subseteq \sup \mathcal{C}(K, L)$.

Note, $\sup \mathcal{C}(K_2', L_2') \subseteq K_2' \subseteq \hat{K}_1$. This containment provides that $\sup \mathcal{C}(K_2', L_2')$ and $\hat{K}_1$ are nonconflicting. Furthermore, by construction $\sup \mathcal{C}(K_2', L_2')$ is controllable with respect to L_2'. $L_2' = \overline{L_{m,2}'} = \overline{\hat{K}_1} \cap L_2$ since $\hat{K}_1$ and $L_{m,2}$ are nonconflicting due to the given property that $(\Sigma(K_1) \cup \Sigma(L_1)) \cap \Sigma(L_2) = \emptyset$. Therefore, Proposition 3.3 gives us that $\sup \mathcal{C}(K_2', L_2')$ is controllable with respect to $\overline{\hat{K}_1} \cap L$. By construction, $\hat{K}_1$ is controllable with respect to L_1. Therefore, $\hat{K}_1$ is controllable with respect to L by Proposition 3.3 also. These results along with Proposition 3.5 then provide that $\hat{K}_1 \cap \sup \mathcal{C}(K_2', L_2') = \sup \mathcal{C}(K_2', L_2')$ is controllable with respect to L. Furthermore, $\sup \mathcal{C}(K_2', L_2') \subseteq K_2' \subseteq K$. Therefore, $\sup \mathcal{C}(K_2', L_2') \in \mathcal{C}(K, L)$ and hence $\sup \mathcal{C}(K_2', L_2') \subseteq \sup \mathcal{C}(K, L)$.

($\supseteq$) Now we need to show that $\sup \mathcal{C}(K_2', L_2') \supseteq \sup \mathcal{C}(K, L)$.

Since the $\sup \mathcal{C}$ operator is monotonic, $K_2' \subseteq \hat{K}_1$ implies that $\sup \mathcal{C}(K_2', L_2') \subseteq \sup \mathcal{C}(\hat{K}_1, L_2')$. Furthermore, $K_2' = K_2 \cap \hat{K}_1 = K_2 \cap L_{m,2}'$ since it is given that $K_2 \subseteq L_{m,2}$. Therefore, $\sup \mathcal{C}(K_2', L_2') = \sup \mathcal{C}(K_2 \cap L_{m,2}', L_2')$ and:

$$\sup \mathcal{C}(K_2', L_2') = \sup \mathcal{C}(\hat{K}_1, L_2') \cap \sup \mathcal{C}(K_2 \cap L_{m,2}', L_2') \tag{3.4}$$

Since $\hat{K}_1$ is controllable with respect to L_1 by construction and $L_2' \subseteq \overline{K_1} \subseteq \overline{L_{m,1}} = L_1$, Proposition 3.3 gives us that $\hat{K}_1$ is controllable with respect to L_2' also. Therefore, $\sup \mathcal{C}(\hat{K}_1, L_2') = \hat{K}_1$. This result along with the fact that $\sup \mathcal{C}(K_2 \cap L_{m,2}', L_2') \subseteq K_2 \subseteq L_{m,2}$, means that the expression in equation (3.4) is equivalent to:

$$\sup \mathcal{C}(K_2', L_2') = \hat{K}_1 \cap \sup \mathcal{C}(K_2 \cap L_{m,2}', L_2') \cap L_{m,2} \tag{3.5}$$

The fact that $K \subseteq K_1$ implies that $\sup \mathcal{C}(K, L) \subseteq \sup \mathcal{C}(K_1, L)$. Furthermore, $\sup \mathcal{C}(K, L) \subseteq \sup \mathcal{C}(K_1, L) \cap L_{m,2}$ since $\sup \mathcal{C}(K, L) \subseteq K \subseteq K_2 \subseteq L_{m,2}$. Applying Corollary 3.8, $\sup \mathcal{C}(K_1, L) \cap L_{m,2} = \sup \mathcal{C}(K_1, L_1) \cap L_{m,2}$ since $(\Sigma(K_1) \cup \Sigma(L_1) \cap \Sigma(L_2) \cap \Sigma_u = \emptyset$ and $K_1 \subseteq L_{m,1}$ is $L_{m,1}$-closed. Therefore:

$$\sup \mathcal{C}(K, L) \subseteq \hat{K}_1 \cap L_{m,2} = L_{m,2}' \tag{3.6}$$

The fact that $\sup \mathcal{C}(K, L) \subseteq K \subseteq K_2$, along with equation (3.6), gives the result that $\sup \mathcal{C}(K, L) \subseteq K_2 \cap L_{m,2}'$. This implies that $\sup \mathcal{C}(\sup \mathcal{C}(K, L), L_2') \subseteq \sup \mathcal{C}(K_2 \cap L_{m,2}', L_2')$. Since $\sup \mathcal{C}(K, L)$ is controllable with respect to L by construction and $L_2' = \overline{\hat{K}_1} \cap L_2 \subseteq L_1 \cap L_2 = L$, Proposition 3.3 provides that $\sup \mathcal{C}(K, L)$ is controllable with respect to L_2' also. Therefore, $\sup \mathcal{C}(\sup \mathcal{C}(K, L), L_2') = \sup \mathcal{C}(K, L)$, which implies:

$$\sup \mathcal{C}(K, L) \subseteq \sup \mathcal{C}(K_2 \cap L_{m,2}', L_2') \tag{3.7}$$

Combining equations (3.6) and (3.7):

$$\sup \mathcal{C}(K, L) \subseteq \hat{K}_1 \cap \sup \mathcal{C}(K_2 \cap L_{m,2}', L_2') \cap L_{m,2}$$

And finally noting equation (3.5) gives us our desired result:

$$\sup \mathcal{C}(K, L) \subseteq \sup \mathcal{C}(K_2', L_2')$$

□

The languages employed in the following theorem use the same notation introduced in Section 3.1. In this case, however, the languages are not equivalent to their previously introduced counterparts because in this section no abstraction is employed. The only exceptions are the languages L_j'' and $L_{m,j}''$ because they were not built with abstraction in Section 3.1 either. Also, the proof of this theorem assumes that only a single specification is addressed per level of hierarchy in order to reduce the complexity of the notation. The results of this and subsequent proofs, however, still hold if multiple specifications are addressed on a given level of the hierarchy.

Theorem 3.10. *The set of modular supervisors constructed by the procedure of Section 3.1 without any abstraction will provide the same behavior as the monolithic supervisor:*

$$\bigcap_{j=1}^{p} \hat{K}_j' = \bigcap_{j=1}^{p} \sup \mathcal{C}(K_j', L_j') = \sup \mathcal{C}(K, L) = \hat{K} \tag{3.8}$$

Proof.

• Throughout this theorem, the following hold:

$$\begin{aligned} K_j' &= \overline{K_{spec,j}} \cap L_{m,j}' \\ K_j'' &= \overline{K_{spec,j}} \cap L_{m,j}'' \end{aligned}$$

• Recalling the modular definitions of equation (2.5), where n corresponds to the total number of subplants and p corresponds to the number of component specifications:

$$\begin{aligned} L &= L_1 \cap \cdots \cap L_n = \bigcap_{j=1}^{p} L_j'' \\ K &= \overline{K_{spec}} \cap L_m = \bigcap_{j=1}^{p} K_j'' \end{aligned}$$

• Now we will show by induction that $\hat{K}_j' = \sup \mathcal{C}(\bigcap_{i=1}^{j} K_i'', \bigcap_{i=1}^{j} L_i'')$ and that $\hat{K}_j'$ is $\bigcap_{i=1}^{j} L_{m,i}''$-closed.

• Beginning on the first level of the hierarchy, the allowable language is built with respect to a subset of the full plant without any control included, that is, $L'_{m,1} = L''_{m,1}$. Therefore, on the first level:

$$\hat{K}'_1 = \sup \mathcal{C}(K''_1, L''_1) \tag{3.9}$$

• Hence $\hat{K}'_1$ is optimal with respect to its portion of the uncontrolled plant L''_1. Also note, since K''_1 is by construction $L''_{m,1}$-closed, $\hat{K}'_1$ is $L''_{m,1}$-closed too since the $\sup \mathcal{C}$ operation preserves closure.

• Moving up to successive levels of the hierarchy, each new allowable language is constructed in a similar manner:

$$\hat{K}'_j = \sup \mathcal{C}(K'_j, L'_j) \tag{3.10}$$

except now, $L'_{m,j}$ consists of a plant subsystem as well as the controlled subplant from the first level.

$$L'_{m,j} = \hat{K}'_{j-1} \cap L''_{m,j} \tag{3.11}$$

• Building the allowable language with respect to an $L'_{m,j}$ constructed in this manner will prove useful since it results in each new allowable language $\hat{K}'_j$ being a subset of the allowable language from the previous level.

$$\begin{aligned}
\hat{K}'_j &= \sup \mathcal{C}(K'_j, L'_j) \\
&\subseteq K'_j = \overline{K_{spec,j}} \cap L'_{m,j} \\
&\subseteq L'_{m,j} = \hat{K}'_{j-1} \cap L''_{m,j} \subseteq \hat{K}'_{j-1}
\end{aligned}$$

This logic can be repeated to show that:

$$\hat{K}'_j \subseteq \bigcap_{i=1}^{j-1} \hat{K}'_i \tag{3.12}$$

• Based on the induction hypothesis, we will assume that:

$$\hat{K}'_{j-1} = \sup \mathcal{C}(\bigcap_{i=1}^{j-1} K''_i, \bigcap_{i=1}^{j-1} L''_i) \tag{3.13}$$

We will also assume that $\hat{K}'_{j-1}$ is closed with respect to $\bigcap_{i=1}^{j-1} L''_{m,i}$.

• The language $L''_{m,j}$ does not share any relevant events with any of the $\hat{K}'_i$ and L''_i from the previous levels. Also, equation (3.12) and the fact that $\hat{K}'_i \subseteq L''_{m,i}$ provide that $\hat{K}'_{j-1} \subseteq \bigcap_{i=1}^{j-1} L''_{m,i} \subseteq \bigcap_{i=1}^{j-1} L''_i$. If we additionally consider equation (3.13) and the fact that $\hat{K}'_{j-1}$ is $\bigcap_{i=1}^{j-1} L''_{m,i}$-closed, Lemma 3.9 can be applied to show:

$$\begin{aligned}\hat{K}'_j &= \sup\mathcal{C}(K'_j, L'_j) \\ &= \sup\mathcal{C}(\hat{K}'_{j-1} \cap K''_j, \overline{\hat{K}'_{j-1}} \cap L''_j) \\ &= \sup\mathcal{C}(\bigcap_{i=1}^{j-1} K''_i \cap K''_j, \bigcap_{i=1}^{j-1} L''_i \cap L''_j) \qquad (3.14)\end{aligned}$$

• Noting equation (3.12), $\hat{K}'_1 \cap \ldots \cap \hat{K}'_{j-1} \cap \hat{K}'_j = \hat{K}'_j$. Therefore, equation (3.14) shows that the conjunction of modular supervisors up to a given point is supremal with respect to the associated subsystem. On this level, that means $\bigcap_{i=1}^{j} \hat{K}'_i$ is supremal with respect to $\bigcap_{i=1}^{j} L''_i$.

• By construction, $\hat{K}'_j$ is $L'_{m,j}$-closed. Examination of equation (3.11) and application of the logic of equation (3.12), therefore, shows that $\hat{K}'_j$ is closed with respect to $\bigcap_{i=1}^{j-1} \hat{K}'_i \cap L''_{m,j}$. Since $\bigcap_{i=1}^{j-1} \hat{K}'_i$ is also $\bigcap_{i=1}^{j-1} L''_{m,i}$-closed by the induction hypothesis, Proposition 3.6 provides that $\hat{K}'_j$ is $\bigcap_{i=1}^{j} L''_{m,i}$-closed.

• The above logic is repeated until all modular supervisors have been addressed. Specifically, Lemma 3.9 is successively applied to show the optimality of a supervisor when moving up a level of the hierarchy. Every time this lemma is applied, optimality with respect to a larger portion of the system is shown. The end result is that the conjunction of modular supervisors provides the same behavior as the monolithic supervisor:

$$\bigcap_{j=1}^{p} \hat{K}_j = \sup\mathcal{C}(\bigcap_{j=1}^{p} K''_j, \bigcap_{j=1}^{p} L''_j) = \sup\mathcal{C}(K, L) = \hat{K}$$

□

The above result states that the conjunction of the modular supervisors built without abstraction will provide the same behavior as the monolithic supervisor. Equivalence to the monolithic solution implies the conjunction of these modular supervisors provides the optimal solution in the sense of being least restrictive. It further implies that the conjunction of modular supervisors meets the given specifications in a nonblocking manner since the monolithic supervisor does. The problem with this result is that if no abstraction is employed, the procedure essentially requires that the full monolithic system be built. In the following we generate results with abstraction included.

3.3 Safe, Nonblocking Control With Abstraction

The motivation for applying abstraction is that by performing analysis and supervisor design on a simplified version of a system, complexity is decreased and understandability is improved. In the context of our approach, we would like to abstract away those elements of our system that are not relevant to those specifications being addressed at the current or higher levels of our hierarchy. For our abstraction, we will simply apply the natural projection operation defined earlier by equation (2.2), where it is further required that each projection P_j possesses the $L'_{m,j}$-observer property. The $L'_{m,j}$-observer property is needed for maintenance of nonblocking, but does not provide for optimality.

In order for the supervisor corresponding to the abstracted language to be applied to the global plant, it is necessary to incorporate a default control that permanently enables all events that had been previously hidden. From an implementation standpoint, this can be achieved by the inverse projection, $\widetilde{K}_j = P_j^{-1}(\overline{\hat{K}_{j,a}})$. In the definitions given below, the restriction to $\widetilde{K}_{j-1} \cap L''_j$ is added to reflect the actual behaviors of the system that can occur and to assist with the proofs that are to follow. In practice, however, these restrictions do not have to be implemented because strings outside the uncontrolled behavior of the plant and outside the behavior allowed by other modular supervisors will not occur anyway.

$$\begin{aligned} \widetilde{K}_j &= P_j^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{j-1} \cap L''_j \\ \widetilde{K}_{m,j} &= P_j^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{m,j-1} \cap L''_{m,j} \end{aligned} \tag{3.15}$$

In the proofs that follow, it is necessary to show that if an abstracted admissible language $\hat{K}_{j,a}$ is $\Sigma_{u,j}$-controllable with respect to $P_j(L'_j)$, then the lifted version $\widetilde{K}_j$ is Σ_u-controllable with respect to the unabstracted language L'_j. Note, $\Sigma_{u,j} = \Sigma_u \cap \Sigma_j$. Furthermore, we need to show that if the supervisor for the abstracted system provides nonblocking control, then the lifted supervisor provides nonblocking control of the actual system.

From examination of the definition of the lifted languages in equation (3.15), one can see that they are contained in one another, thereby maintaining nonconflict by construction. Furthermore, it can be shown $\widetilde{K}_{m,j} = \widetilde{K}_j \cap L_m$.

In demonstrating these results, we will find the following properties of the natural and inverse projections helpful [8]:

Property 1 $P(tu) = P(t)P(u)$.
Property 2 $\overline{P(L)} = P(\overline{L})$.
Property 3 $P^{-1}(tu) = P^{-1}(t)P^{-1}(u)$.
Property 4 $\overline{P^{-1}(L)} = P^{-1}(\overline{L})$.
Property 5 $L \subseteq P^{-1}(P(L))$.
Property 6 If $L_1 \subseteq L_2$, then $P(L_1) \subseteq P(L_2)$ and $P^{-1}(L_1) \subseteq P^{-1}(L_2)$
Property 7 Let $L_1,\ L_2\ \subseteq \Sigma_0^*,\ P_0 : \Sigma^* \to \Sigma_0^*$. Then, $P_0^{-1}(L_1) \cap P_0^{-1}(L_2) = P_0^{-1}(L_1 \cap L_2)$.
Property 8 For alphabets $\Sigma_0 \subseteq \Sigma_1 \cup \Sigma_2$, let $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ and $P_0 : (\Sigma_1 \cup \Sigma_2) \to \Sigma_0^*$. Then, $P_0(L_1 \| L_2) \subseteq P_0(L_1) \| P_0(L_2)$ and the equality holds provided that $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma_0$ (Proposition 2.3).

We will also need some additional propositions. The following proposition will be used in showing the maintenance of controllability properties between abstracted and lifted languages. It is proven using logic taken from [45].

Proposition 3.11. *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let $L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. Also let $\overline{\widetilde{K}_m} = P^{-1}(\overline{K_a}) \cap L$, $\Sigma_u \subseteq \Sigma$ and $\Sigma_{u,a} = \Sigma_u \cap \Sigma_a$. If K_a is $\Sigma_{u,a}$-controllable with respect to $P(L)$, then $\widetilde{K}_m$ is Σ_u-controllable with respect to L.*

Proof. See proof in Appendix. □

The proposition given below is employed to show nonblocking of the abstracted system implies nonblocking of the lifted system. The result follows closely from logic in [75].

Proposition 3.12. *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let $\overline{L_m} = L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. Also let $\widetilde{K} = P^{-1}(\overline{K_a}) \cap L$ and $\widetilde{K}_m = P^{-1}(\overline{K_a}) \cap L_m$. If the projection P possesses the L_m-observer property and $K_a \subseteq P(L_m)$, then $\overline{\widetilde{K}_m} = \widetilde{K}$.*

Proof. See proof in Appendix. □

This next proposition demonstrates that if the marking of an abstracted language is consistent with the marking of $P(L_m)$, then the marking of the lifted language will be consistent with the marking of L_m.

Proposition 3.13. *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let $\overline{L_m} = L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. If K_a is $P(L_m)$-closed, then $P^{-1}(\overline{K_a}) \cap L_m = P^{-1}(K_a) \cap L_m$.*

Proof.

$$\begin{aligned}
\overline{K_a} \cap P(L_m) &= K_a \\
P^{-1}(\overline{K_a} \cap P(L_m)) &= P^{-1}(K_a) \\
P^{-1}(\overline{K_a}) \cap P^{-1}(P(L_m)) &= P^{-1}(K_a) \\
P^{-1}(\overline{K_a}) \cap P^{-1}(P(L_m)) \cap L_m &= P^{-1}(K_a) \cap L_m \\
P^{-1}(\overline{K_a}) \cap L_m &= P^{-1}(K_a) \cap L_m
\end{aligned}$$

□

This final proposition is quite important and will be used to show that languages from successive levels of our hierarchy are nonconflicting.

Proposition 3.14. *Let $P : \Sigma^* \to \Sigma_a^*$ be a natural projection and let $K_1,\ K_2 \subseteq \Sigma^*$ be languages. Also let $\Sigma(K_2) \subseteq \Sigma_a$. If $P(K_2) \subseteq P(K_1)$ and P has the K_1-observer property, then K_1 and K_2 are nonconflicting.*

Proof. Let there be a string $s \in \Sigma^*$ such that $s \in \overline{K_1} \cap \overline{K_2}$. We must then show that K_1 and K_2 share a completion of this string. Since $\Sigma(K_2) \subseteq \Sigma_a$, $\exists t \in \Sigma_a^*$ such that $st \in K_2$. Also since $\Sigma(K_2) \subseteq \Sigma_a$, $P(s)t \in P(K_2) \subseteq P(K_1)$. By the K_1-observer property, $\exists u$ such that $su \in K_1$ and $P(su) = P(s)t$. It then follows that $su \in P^{-1}(P(su)) \subseteq P^{-1}(P(K_2)) = K_2$ since $\Sigma(K_2) \subseteq \Sigma_a$. Therefore, K_1 and K_2 are nonconflicting. □

The propositions provided above will be used to prove the main results of this chapter. Before we demonstrate these results, however, we need to examine the exact manner in which our languages are projected and then lifted back to the global alphabet.

In the procedure of Section 3.1, the projection operation $P_2 : \Sigma^* \to \Sigma_2^*$ was defined. This projection was constructed so that it maintained the observer property with respect to each of the elements that went into constructing L'_2, specifically, with respect to $\hat{K}'_{1,a}$ as well as each of the elements of $\{L_{i_1+1}, \ldots, L_{i_2}\}$. Since each of these component languages have disjoint sets of relevant events, P_2 has the $L'_{m,2}$-observer property by Theorem 2.5. It was also stated that P_2 does not erase any relevant events from the language $L''_2 = L_{i_1+1}\| \ldots \|L_{i_2}$ that would not also be erased by the projection $P_1 : \Sigma^* \to \Sigma_1^*$. Furthermore, the events erased by P_2 which are relevant to $\hat{K}'_{1,a}$ are disjoint from those which are relevant to L''_2.

These facts are true for the projection $P_j : \Sigma^* \to \Sigma_j^*$ in general. As such, in the proofs of the following lemmas it will be assumed that each projection P_j will be performed in two stages. That is, the projection P_1 will be performed first erasing the relevant events of L''_j followed by a projection $P'_j = \Sigma_1^* \to \Sigma_j^*$ which erases the events relevant to $\hat{K}'_{j,a}$. It is well known that the natural projection satisfies a chain rule property such that $P_j = P'_j \circ P_1$. The proofs to follow will also implement the inverse projection P_j^{-1} using the same two stages. Specifically, $P_j'^{-1}$ will be applied throughout the proofs to generate an intermediate lifted language $\widetilde{K}_{j,a}$. The remaining events will be added separately via the P_1^{-1} operation at the end leading to $\widetilde{K}_j$.

By Proposition 2.3, projection distributes across synchronous composition if no shared relevant events are abstracted away. Since the $L''_{m,j}$ do not share any relevant events, we can, therefore, define $\widetilde{L}_a = P_1(L)$ and $\widetilde{L}_{m,a} = P_1(L_m)$. Two other languages projected to the alphabet Σ_1 are defined $\widetilde{L}'_{m,j,a} = P_1(L'_{m,j})$ and $\widetilde{L}''_{m,j,a} = P_1(L''_{m,j})$. These newly defined languages can also be interpreted as versions of the abstracted languages with alphabets equal to Σ_j lifted to the event set Σ_1, rather than all the way to the global alphabet Σ. Figure 3.9 helps to illustrate the relationship between these newly defined languages.

In a similar manner to the expressions of equation (3.15), restrictions have been added to the following definitions of $\widetilde{K}_{j,a}$ in order to assist with the proofs that are to follow.

$$\begin{aligned}
\widetilde{K}_{j,a} &= P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{j-1,a} \cap \widetilde{L}''_{j,a} \\
\widetilde{K}_{m,j,a} &= P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{m,j-1,a} \cap \widetilde{L}''_{m,j,a}
\end{aligned} \tag{3.16}$$

Iteratively expanding the terms for the lower level supervisors, $\widetilde{K}_{j-1,a}$, leads to the

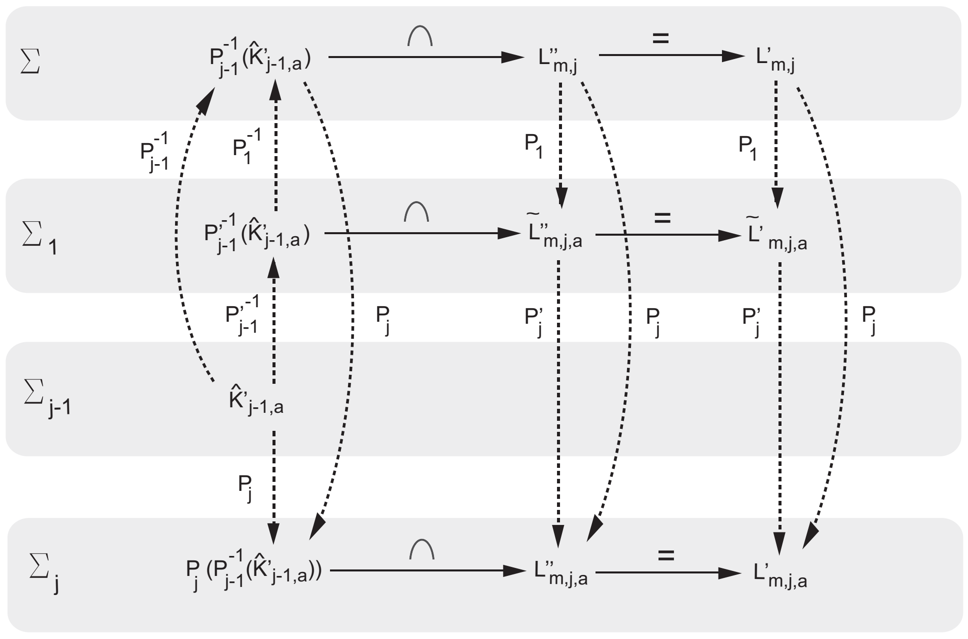

Figure 3.9: Diagram representing the relationship between various languages and alphabets

following:

$$\widetilde{K}_{j,a} = \bigcap_{i=1}^{j} P_i'^{-1}(\overline{\hat{K}'_{i,a}}) \cap \bigcap_{i=1}^{j} \widetilde{L}''_{i,a}$$

$$\widetilde{K}_{m,j,a} = \bigcap_{i=1}^{j} P_i'^{-1}(\overline{\hat{K}'_{i,a}}) \cap \bigcap_{i=1}^{j} \widetilde{L}''_{m,i,a} \tag{3.17}$$

Also note the following definitions of $\widetilde{L}'_{j,a}$ and $\widetilde{L}'_{m,j,a}$ that also correspond to their unabstracted counterparts.

$$\widetilde{L}'_{j,a} = P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}) \cap \widetilde{L}''_{j,a}$$

$$\widetilde{L}'_{m,j,a} = P'^{-1}_{j-1}(\hat{K}'_{j-1,a}) \cap \widetilde{L}''_{m,j,a}$$

It then follows that $\overline{\widetilde{L}'_{m,j,a}} = \widetilde{L}'_{j,a}$ since $P_1'^{-1}(\hat{K}'_{j-1,a})$ and $\widetilde{L}''_{m,j,a}$ are nonconflicting since they do not share any relevant events. Additionally, since each $K'_{j,a}$ is by construction $L'_{m,j,a}$-closed, where $L'_{m,j,a} = P'_j(\widetilde{L}'_{m,j,a})$, each $\hat{K}'_{j,a}$ is also $P'_j(\widetilde{L}'_{m,j,a})$-closed since the $\sup \mathcal{C}$ operation will maintain the closure. Therefore, Proposition 3.13 can be applied iteratively to the above expression for $\widetilde{K}_{m,j,a}$ to generate the following:

$$\widetilde{K}_{m,j,a} = P_j'^{-1}(\hat{K}'_{j,a}) \cap \widetilde{K}_{m,j-1,a} \cap \widetilde{L}''_{m,j,a} \tag{3.18}$$

Now noting that $\widetilde{K}_{j-1,a} \subseteq P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}})$ and $\widetilde{K}_{m,j-1,a} \subseteq P'^{-1}_{j-1}(\hat{K}'_{j-1,a})$, equation

(3.16) can be expressed:

$$\begin{aligned}
\widetilde{K}_{j,a} &= P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{j-1,a} \cap P_{j-1}'^{-1}(\overline{\hat{K}'_{j-1,a}}) \cap \widetilde{L}''_{j,a} \\
&= P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{j-1,a} \cap \widetilde{L}'_{j,a} \\
\widetilde{K}_{m,j,a} &= P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{m,j-1,a} \cap P_{j-1}'^{-1}(\hat{K}'_{j-1,a}) \cap \widetilde{L}''_{m,j,a} \\
&= P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{K}_{m,j-1,a} \cap \widetilde{L}'_{m,j,a} \qquad (3.19)
\end{aligned}$$

The expressions of equation (3.19) will prove useful in the following lemmas.

By definition, P_1 has the $L''_{m,j}$-observer property for all $j \in \{1, \ldots, m\}$. It then follows that since P_1 does not erase any shared relevant events, it will also possess the L_m-observer property by Theorem 2.5. Furthermore, since P_1 erases only irrelevant events from $P_{j-1}^{-1}(\overline{\hat{K}'_{j-1,a}})$, P'_j will have the observer property with respect to $P_{j-1}'^{-1}(\overline{\hat{K}'_{j-1,a}})$. Since in addition P'_j erases only irrelevant events from $\widetilde{L}''_{m,j,a}$, it also possesses the $\widetilde{L}'_{m,j,a}$-observer property again by Theorem 2.5.

We will now present the first main result of the chapter, Lemma 3.15. This lemma shows that the behavior allowed by the conjunction of modular supervisors constructed by the procedure of Section 3.1 is nonblocking. This will be proven by induction, where the induction step shows that the intersection of a sequential set of languages that represent the supervised behavior of each module is nonconflicting with the supervised behavior from the next level of the hierarchy.

Lemma 3.15. *The conjunction of modular supervisors constructed by the procedure of Section 3.1 is nonblocking if such a set of nonempty modular supervisors exists, that is*

$$\overline{\widetilde{K}_{m,1} \cap \widetilde{K}_{m,2} \cap \ldots \cap \widetilde{K}_{m,p}} = \widetilde{K}_1 \cap \widetilde{K}_2 \cap \ldots \cap \widetilde{K}_p$$

Proof.

• The first step of this proof is to show that $\overline{\bigcap \widetilde{K}_{m,j,a}} = \bigcap \widetilde{K}_{j,a}$.

• On the first level of the hierarchy, $\widetilde{L}'_{m,1,a} = \widetilde{L}''_{m,1,a}$. Furthermore, since P_1 is the projection employed on the first level, $\hat{K}'_{1,a} \subseteq L''_{m,1,a} = \widetilde{L}''_{m,1,a}$. Therefore, $\hat{K}'_{1,a} = P_1'^{-1}(\hat{K}'_{1,a})$ and $\widetilde{L}''_{m,1,a}$ are nonconflicting and $\overline{\widetilde{K}_{m,1,a}} = \widetilde{K}_{1,a}$ as built in equations (3.16) and (3.18):

$$\overline{P_1'^{-1}(\hat{K}'_{1,a}) \cap \widetilde{L}''_{m,1,a}} = P_1'^{-1}(\overline{\hat{K}'_{1,a}}) \cap \widetilde{L}''_{1,a}$$

• Moving up to successive levels of the hierarchy, each $\widetilde{L}'_{m,j,a}$ now consists of a plant subsystem as well as the controlled subplant from the previous level. Since

$\overline{\widetilde{L}'_{m,j,a}} = \widetilde{L}'_{j,a}$ and $\hat{K}'_{j,a} \subseteq L'_{m,j,a} = P'_j(\widetilde{L}'_{m,j,a})$, we can apply Proposition 4.7.

$$\overline{P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{m,j,a}} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{j,a}$$

• Intersecting both sides of the above with $\overline{\widetilde{K}_{m,j-1,a}} = \widetilde{K}_{j-1,a}$ gives us:

$$\overline{P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{m,j,a}} \cap \overline{\widetilde{K}_{m,j-1,a}} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{j,a} \cap \widetilde{K}_{j-1,a} \tag{3.20}$$

• It is now necessary to show that $P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{m,j,a}$ and $\widetilde{K}_{m,j-1,a}$ are nonconflicting.

• First note the following relation which comes from examination of expressions that are analogous to equations (3.1) and (3.3).

$$\hat{K}'_{j,a} \subseteq K'_{j,a} \subseteq L'_{m,j,a} = P'_j(\widetilde{L}'_{m,j,a}) \subseteq P'_j(\widetilde{L}''_{m,j,a})$$

• Also recalling that each P'_j erases only irrelevant events from $\widetilde{L}''_{m,j,a}$, the following result holds:

$$P'^{-1}_j(\hat{K}'_{j,a}) \subseteq P'^{-1}_j(P'_j(\widetilde{L}''_{m,j,a})) = \widetilde{L}''_{m,j,a} \tag{3.21}$$

• taking the prefix-closure of both sides of the above provides the following additional result,

$$P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \subseteq \overline{\widetilde{L}''_{m,j,a}} = \widetilde{L}''_{j,a} \tag{3.22}$$

• Since $K'_{j,a}$ is by construction $P'_j(\widetilde{L}'_{m,j,a})$-closed, $\hat{K}'_{j,a}$ is also $P'_j(\widetilde{L}''_{m,j,a})$-closed since the sup$\mathcal{C}$ operation will maintain the closure. Therefore, Proposition 3.13 and equation (3.21) can be employed to transform $P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{m,j,a}$ into:

$$P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{m,j,a} = P'^{-1}_j(\hat{K}'_{j,a}) \cap (P'^{-1}_{j-1}(\hat{K}'_{j-1,a}) \cap \widetilde{L}''_{m,j,a}) = P'^{-1}_j(\hat{K}'_{j,a}) \cap P'^{-1}_{j-1}(\hat{K}'_{j-1,a}) \tag{3.23}$$

Likewise employing equation (3.22),

$$P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{j,a} = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap (P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}) \cap \widetilde{L}''_{j,a}) = P'^{-1}_j(\overline{\hat{K}'_{j,a}}) \cap P'^{-1}_{j-1}(\overline{\hat{K}'_{j-1,a}}) \tag{3.24}$$

Referencing equation (3.17) and repeating the logic of equations (3.23) and (3.24):

$$\widetilde{K}_{m,j-1,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\overline{\hat{K}'_{i,a}}) \cap \bigcap_{i=1}^{j-1} \widetilde{L}''_{m,i,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\hat{K}'_{i,a}) \tag{3.25}$$

$$\widetilde{K}_{j-1,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\overline{\hat{K}'_{i,a}}) \cap \bigcap_{i=1}^{j-1} \widetilde{L}''_{i,a} = \bigcap_{i=1}^{j-1} P'^{-1}_i(\overline{\hat{K}'_{i,a}}) \tag{3.26}$$

• Employing equations (3.23), (3.24), (3.25), and (3.26), equation (3.20) becomes:

$$\overline{P_j'^{-1}(\hat{K}_{j,a}') \cap P_{j-1}'^{-1}(\hat{K}_{j-1,a}')} \cap \overline{\bigcap_{i=1}^{j-1} P_i'^{-1}(\hat{K}_{i,a}')} =$$

$$P_j'^{-1}(\overline{\hat{K}_{j,a}'}) \cap P_{j-1}'^{-1}(\overline{\hat{K}_{j-1,a}'}) \cap \bigcap_{i=1}^{j-1} P_i'^{-1}(\overline{\hat{K}_{i,a}'}) \tag{3.27}$$

• Now note that $\hat{K}_{j,a}' \subseteq L_{m,j,a}' \subseteq P_j(\hat{K}_{j-1,a}')$. Therefore, $P_j'(P_j'^{-1}(\hat{K}_{j,a}')) \subseteq P_j'(P_{j-1}'^{-1}(\hat{K}_{j-1,a}'))$. Furthermore, $\Sigma(P_j'^{-1}(\hat{K}_{j,a}')) \subseteq \Sigma_j$. Also since P_j' has the observer property with respect to $P_{j-1}'^{-1}(\hat{K}_{j-1,a}')$, we can apply Proposition 3.14 to get the following:

$$P_j'^{-1}(\overline{\hat{K}_{j,a}'}) \cap P_{j-1}'^{-1}(\overline{\hat{K}_{j-1,a}'}) = \overline{P_j'^{-1}(\hat{K}_{j,a}') \cap P_{j-1}'^{-1}(\hat{K}_{j-1,a}')}$$

• Since in general, $P_{j-k+1}'(\cap_{i=j-k+1}^{j} P_i'^{-1}(\hat{K}_{i,a}')) \subseteq P_{j-k+1}'(P_{j-k}^{-1}(\hat{K}_{j-k,a}'))$, $\Sigma(\cap_{i=j-k+1}^{j} P_i'^{-1}(\hat{K}_{i,a}')) \subseteq \Sigma_{j-k+1}$, and P_{j-k+1}' has the observer property with respect to $P_{j-k}'^{-1}(\hat{K}_{j-k,a}')$, we can repeatedly apply Proposition 3.14,

$$\bigcap_{i=1}^{j} P_i'^{-1}(\overline{\hat{K}_{i,a}'}) = \overline{\bigcap_{i=1}^{j} P_i'^{-1}(\hat{K}_{i,a}')} \tag{3.28}$$

• Comparing the result of equation (3.28) to equation (3.27), therefore, shows that $P_j'^{-1}(\hat{K}_{j,a}') \cap \widetilde{L}_{m,j,a}'$ and $\widetilde{K}_{m,j-1,a}$ are nonconflicting. And hence equation (3.20) becomes:

$$\overline{P_j'^{-1}(\hat{K}_{j,a}') \cap \widetilde{L}_{m,j,a}' \cap \widetilde{K}_{m,j-1,a}} = P_j'^{-1}(\overline{\hat{K}_{j,a}'}) \cap \widetilde{L}_{j,a}' \cap \widetilde{K}_{j-1,a}$$

• Then applying Proposition 4.7 once more,

$$\overline{P_j'^{-1}(\overline{\hat{K}_{j,a}'}) \cap \widetilde{L}_{m,j,a}' \cap \widetilde{K}_{m,j-1,a}} = P_j'^{-1}(\overline{\hat{K}_{j,a}'}) \cap \widetilde{L}_{j,a}' \cap \widetilde{K}_{j-1,a} \tag{3.29}$$

Comparing the above to equation (3.19) demonstrates that in general, $\overline{\widetilde{K}_{m,j,a}} = \widetilde{K}_{j,a}$.

• Furthermore, $\widetilde{K}_{m,j,a}$, $\widetilde{K}_{m,j-1,a}$, ..., $\widetilde{K}_{m,1,a}$ are nonconflicting since $\widetilde{K}_{m,j,a} \subseteq \widetilde{K}_{m,j-1,a} \subseteq \ldots \subseteq \widetilde{K}_{m,1,a}$ by definition. Therefore,

$$\overline{\widetilde{K}_{m,1,a} \cap \ldots \cap \widetilde{K}_{m,p,a}} = \widetilde{K}_{1,a} \cap \ldots \cap \widetilde{K}_{p,a} \tag{3.30}$$

• The only step left is to address those events abstracted away by P_1.

• Using the definitions in equation (3.17), we get the following:

$$\begin{aligned}\bigcap_{j=1}^{p} \widetilde{K}_{j,a} &= \bigcap_{j=1}^{p} P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}_a \\ \bigcap_{j=1}^{p} \widetilde{K}_{m,j,a} &= \bigcap_{j=1}^{p} P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}_{m,a} \end{aligned} \tag{3.31}$$

Therefore, $\bigcap_{j=1}^{p} \widetilde{K}_{m,j,a} \subseteq \widetilde{L}_{m,a} = P_1(L_m)$.

• Since it is further known that P_1 possesses the L_m-observer property and $\overline{L_m} = L$, Proposition 4.7 gives us that:

$$\overline{P_1^{-1}(\overline{\bigcap_{j=1}^{p} \widetilde{K}_{m,j,a}}) \cap L_m} = P_1^{-1}(\overline{\bigcap_{j=1}^{p} \widetilde{K}_{m,j,a}}) \cap L \tag{3.32}$$

• Recalling equations (3.17), (3.30), (3.31) and the fact that $L_m \subseteq L \subseteq P_1^{-1}(P_1(L)) = P_1^{-1}(L_a)$, we can show the following:

$$\begin{aligned} P_a^{-1}(\overline{\bigcap_{j=1}^{p} \widetilde{K}_{m,j,a}}) \cap L_m &= P_1^{-1}(\bigcap_{j=1}^{p} \widetilde{K}_{j,a}) \cap L_m \\ &= P_1^{-1}(\bigcap_{j=1}^{p} P_j'^{-1}(\overline{\hat{K}_{j,a}}) \cap L_a) \cap L_m \\ &= \bigcap_{j=1}^{p} P_1^{-1}(P_j'^{-1}(\overline{\hat{K}_{j,a}})) \cap P_1^{-1}(L_a) \cap L_m \\ &= \bigcap_{j=1}^{p} P_j^{-1}(\overline{\hat{K}_{j,a}}) \cap L_m = \bigcap_{j=1}^{p} \widetilde{K}_{m,j} \end{aligned} \tag{3.33}$$

• Similarly, we can show

$$P_1^{-1}(\overline{\bigcap_{j=1}^{p} \widetilde{K}_{m,j,a}}) \cap L = \bigcap_{j=1}^{p} \widetilde{K}_j \tag{3.34}$$

• Examining equations (3.32), (3.33), and (3.34), we have shown our desired result, $\overline{\widetilde{K}_{m,1} \cap \widetilde{K}_{m,2} \cap \ldots \cap \widetilde{K}_{m,p}} = \widetilde{K}_1 \cap \widetilde{K}_2 \cap \ldots \cap \widetilde{K}_p$. □

Lemma 3.16 is the second main result of this chapter; it shows that the conjunction of languages representing the supervised behavior of each of the modules is Σ_u-controllable with respect to L. The proof of this result follows the same type of induction logic used in proving Lemma 3.15.

Lemma 3.16. *The conjunction of modular supervisors constructed by the procedure of Section 3.1, $\widetilde{K}_{m,1} \cap \widetilde{K}_{m,2} \cap \ldots \cap \widetilde{K}_{m,p}$, is Σ_u-controllable with respect to L if such a set of nonempty modular supervisors exists.*

Proof.

• The first step of this proof is to show that $\bigcap \widetilde{K}_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect $L_a = P_1(L)$.

• Beginning on the first level of hierarchy, $\widetilde{L}'_{m,1,a} = \widetilde{L}''_{m,1,a}$. Therefore according to equation (3.18), $\widetilde{K}_{m,1,a} = P_1'^{-1}(\hat{K}'_{1,a}) \cap \widetilde{L}''_{m,1,a}$

• Furthermore, since P_1 is the projection employed on the first level, $P_1'^{-1}(\hat{K}'_{1,a}) = \hat{K}'_{1,a}$ and $\widetilde{L}''_{m,1,a} = L''_{m,1,a}$. Therefore, $\widetilde{K}_{m,1,a} = \hat{K}'_{1,a} \cap \widetilde{L}''_{m,1,a}$

• By construction, $\hat{K}'_{1,a}$ is $\Sigma_{u,1}$-controllable with respect to $L''_{1,a} = \widetilde{L}''_{1,a}$. Furthermore, since $\hat{K}'_{1,a} \subseteq L''_{m,1,a} = \widetilde{L}''_{m,1,a}$, $\hat{K}'_{1,a}$ and $\widetilde{L}''_{m,1,a}$ are nonconflicting. Therefore, Proposition 3.4 provides that $\widetilde{K}_{m,1,a}$ is $\Sigma_{u,1}$-controllable with respect to $\widetilde{L}''_{1,a}$.

• Furthermore, since $L_a \subseteq \widetilde{L}''_{1,a}$, Proposition 3.3 provides that $\widetilde{K}_{m,1}$ is $\Sigma_{u,1}$-controllable with respect to L_a.

• Moving up to higher levels of the hierarchy, each $\hat{K}'_{j,a}$ is $\Sigma_{u,j}$-controllable with respect to $L'_{j,a} = \overline{P'_j(\widetilde{L}'_{j,a})}$ by construction where it is now the case that $\widetilde{L}'_{m,j,a} \neq \widetilde{L}''_{m,j,a}$. Each $P_j'^{-1}(\hat{K}'_{j,a}) \cap \widetilde{L}'_{m,j,a}$ is then $\Sigma_{u,1}$-controllable with respect to $\widetilde{L}'_{j,a}$ by Proposition 3.11. $\widetilde{L}'_{j,a}$ is built to include a plant subsystem as a well as the controlled subplant from the previous level, $\widetilde{L}'_{j,a} = P_{j-1}'^{-1}(\overline{\hat{K}'_{j-1,a}}) \cap \widetilde{L}''_{j,a}$.

• Applying Proposition 3.3, $\overline{P_j'^{-1}(\hat{K}'_{j,a}) \cap \widetilde{L}'_{m,j,a}}$ is $\Sigma_{u,1}$-controllable with respect to $\widetilde{K}_{j-1,a} \cap L_a$. Furthermore, based on the logic of Lemma 3.15, $P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{m,j,a}$ and $\widetilde{K}_{m,j-1,a}$ are nonconflicting. Also from the induction hypothesis, $\widetilde{K}_{m,j-1,a}$ is $\Sigma_{u,1}$-controllable with respect to L_a. Therefore, Proposition 3.5 provides that $P_j'^{-1}(\overline{\hat{K}'_{j,a}}) \cap \widetilde{L}'_{m,j,a} \cap \widetilde{K}_{m,j-1,a}$ is $\Sigma_{u,1}$-controllable with respect to L_a.

• Examining equation (3.19), it can then be seen that each $\widetilde{K}_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect to L_a. Furthermore, since $\widetilde{K}_{m,j,a} \subseteq \widetilde{K}_{m,j-1,a} \subseteq \ldots \widetilde{K}_{m,1,a}$, the set of modular supervisors are nonconflicting. Therefore, Proposition 3.4 provides that the conjunction $\bigcap \widetilde{K}_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect L_a.

• The only step left then is to lift each $\widetilde{K}_{m,j,a}$ to the global alphabet.

• Recall, $L_a = P_1(L)$. Since $\bigcap \widetilde{K}_{m,j,a}$ is $\Sigma_{u,1}$-controllable with respect to $P_1(L)$, Proposition 3.11 provides that $P_1^{-1}(\overline{\bigcap \widetilde{K}_{m,j,a}}) \cap L_m$ is Σ_u-controllable with respect to L. Examination of equation (3.33), therefore, demonstrates that the behavior allowed by the conjunction of modular supervisors $\widetilde{K}_{m,1} \cap \widetilde{K}_{m,2} \cap \ldots \cap \widetilde{K}_{m,p}$ is Σ_u-controllable with respect to L. $\square$

Lemma 3.15 and Lemma 3.16 then provide us with the overall desired result, that the behavior provided by the supervisor languages constructed in Section 3.1 is safe and nonblocking. Specifically, Lemma 3.15 demonstrates that the behavior is nonblocking and Lemma 3.16 demonstrates that the control required by the supervisors is realizable.

Theorem 3.17. *The set of modular supervisors constructed by the procedure of Section 3.1 will meet the given specifications in a nonblocking manner when acting in conjunction if such a set of nonempty nonblocking modular supervisors exists.*

Proof. Follows directly from Lemma 3.15 and Lemma 3.16. $\square$

The result of Theorem 3.17 demonstrates that employing abstractions that possess the respective observer properties provides Σ_u-controllability and nonblocking. However, it does not necessarily lead to an optimal solution. This suboptimality arises at least in part from the fact that if a relevant controllable event is abstracted away, the ability to disable it as a means of preventing an uncontrollable continuation is lost.

In the context of a single system, [80] introduced a property called output-control-consistency (OCC) that addresses this situation. The presence of this OCC property along with the observer property can be employed to demonstrate that the optimal control generated for an abstracted system can be applied to the unabstracted system and still achieve optimal supervision. Maintenance of the OCC property in modular approaches, however, is less well understood. For our approach, we will leave the problem of optimal supervision as an open problem. It is, however, understood that abstracting away fewer controllable events will make the behavior allowed by our approach less restrictive.

3.4 Implementation Examples and Discussion

In this section we will implement our procedure on two moderately large examples. The purpose of this exercise is to further demonstrate the strengths of our approach, and to provide some heuristics for its implementation. Specifically, guidelines will be provided for how to choose the order in which the modular specifications are addressed. As was demonstrated in the example of Section 3.1, the order in which the modular supervisors are built is not unique and can have a large effect on the size of the resulting supervisors.

3.4.1 Ordering algorithm

Examining the partitioning of the FMS example shown in Fig. 3.1, one can see that there are two disjoint modules on the first level of the hierarchy. In constructing the supervisor languages for these modules, very little abstraction will be possible since all of the component specifications are on the current or remaining levels of the hierarchy. In moving up to the next level of the hierarchy, the two existing supervisor languages will have to be composed leading to a multiplicative growth of the state space. Since these two supervisor languages do not share any relevant events, their composition did not increase the number of events that can be considered for abstraction. Therefore, this partition is not very efficient because it unnecessarily grows the size of the state space without helping to increase the amount of reduction that can be achieved by the abstraction.

A more efficient approach is to address only a single specification per level. This allows for more reduction since each time we move up a level of the hierarchy more events become available for abstraction. This approach has been assumed in the procedure of Section 3.1 and the proofs found throughout the paper.

Even with this improvement, we are still left with the question of which order to address the modular specifications in. The guideline we will choose to adhere to here is to maximize the amount of reduction possible at lower levels of the hierarchy. In order to make application of this guideline more rigorous, we will adapt an algorithm from [65].

In our approach, since each level of the hierarchy only addresses one specification, each successive "plant" has the form $L'_{m,j+1,a} = P_{j+1}(\hat{K}'_{j,a})\|L''_{m,j+1,a}$. The decision as to which specification to address next will be made based on the potential characteristics of the resulting subsystem $K'_{j+1,a} = \overline{K_{spec,j+1,a}}\|L'_{m,j+1,a}$. This evaluation will be made employing the following two metrics:

$$\begin{aligned} A_{j+1} &:= \frac{\left|\left|\overline{K_{spec,j+1}} \| \hat{K}'_{j,a} \| L''_{m,j+1}\right|\right|}{\left|\left|\overline{K_{spec,j+1}}\right|\right| \times \left|\left|\hat{K}'_{j,a}\right|\right| \times \left|\left|L''_{m,j+1}\right|\right|} \\ B_{j+1} &:= \frac{\left|\left|P_{j+1}(\overline{K_{spec,j+1}} \| \hat{K}'_{j,a} \| L''_{m,j+1})\right|\right|}{\left|\left|\overline{K_{spec,j+1}} \| \hat{K}'_{j,a} \| L''_{m,j+1}\right|\right|} \end{aligned}$$

In the above, the notation $\left|\left| \cdot \right|\right|$ represents the cardinality of the automaton which generates the associated language, that is, the size of the automaton's state space. In general, we will employ a minimal automaton to generate the given language. Specifically, A_{j+1} is a measure of the potential size reduction due to some states not being reachable in the composition, while B_{j+1} is a measure of the potential for reduction due to abstraction. The projection $P_{j+1} : \Sigma^* \rightarrow \Sigma^*_{j+1}$ erases those events that are not relevant to the current or remaining specifications and that maintain the associated observer properties. Since in choosing the next specification it may not be possible to minimize both A_{j+1} and B_{j+1}, we will rather try and minimize their product. This idea leads to the metric that we will ultimately use:

$$\psi_{j+1} = A_{j+1} \cdot B_{j+1} = \frac{\left|\left|P_{j+1}(\overline{K_{spec,j+1}} \| \hat{K}'_{j,a} \| L''_{m,j+1})\right|\right|}{\left|\left|\overline{K_{spec,j+1}}\right|\right| \times \left|\left|\hat{K}'_{j,a}\right|\right| \times \left|\left|L''_{m,j+1}\right|\right|}$$

Therefore given the closed loop subsystem from the previous level $\hat{K}'_{j,a}$, the next specification will be chosen based on which one minimizes ψ_{j+1}. Since choosing a specification that is disjoint from $\hat{K}'_{j,a}$ will not result in any additional events becoming "local," only those remaining specifications that "neighbor" the current $\hat{K}'_{j,a}$ will be chosen from. This set of neighbors is defined as follows, $N_j = \{k \mid [k > j] \wedge [\text{rel}(\overline{K_{spec,j+1}}) \cap \text{rel}(\hat{K}'_{j,a}) \neq \emptyset]\}$, where it is assumed the indexing is sequential and neighbors are defined as sharing relevant events. One desirable element of the metric ψ_{j+1} is that it does not actually require computing the unabstracted automaton that generates the language $\overline{K_{spec,j+1}} \| \hat{K}'_{j,a} \| L''_{m,j+1}$.

Before the algorithm is begun, there is no $\hat{K}'_{j,a}$ from the previous level and no set of neighbors. Therefore, we will choose the first specification using the same metric, but with the $\hat{K}'_{j,a}$ term missing. Furthermore, since no specifications have been addressed yet, we will use $P_1 : \Sigma^* \rightarrow \Sigma^*_1$ as the initial projection since it erases only events that are not relevant to any specifications. A summary of this algorithm is given below:

Algorithm 3.18. *Specification Ordering Algorithm*

Step 1: Initially, choose a specification such that $\frac{\left|\left|P_1(\overline{K_{spec,1}} \| L''_{m,1})\right|\right|}{\left|\left|\overline{K_{spec,1}}\right|\right| \times \left|\left|L''_{m,1}\right|\right|}$ is minimum.

Step 2: Follow procedure of Section 3.1 for building the first modular supervisor, which generates $\hat{K}'_{1,a}$.

Step 3: Choose next specification from the set of neighbors, N_1, such that ψ_2 is minimum.

Step 4: Follow procedure of Section 3.1 for building the next modular supervisor, which generates $\hat{K}'_{2,a}$

Step 5: Repeat steps 3 and 4 until there are no more specifications left to address.

3.4.2 Flexible Manufacturing System (FMS) example

The first example we will examine will be the FMS example first introduced in Chapter 1 and shown in Fig. 1.3. The basic idea is that parts enter from the left via the conveyor *Con2*. From *Con2* the parts pass through buffer *B2* to a handling robot. This robot then passes parts, through buffer *B4*, to a lathe which can generate two different types of parts. After the lathe has finished an operation and returned a part to the robot, again through buffer *B4*, the robot then passes the part to either buffer *B6* or buffer *B7* depending on the part type. If passed to *B7*, the part is then sent to a painting machine *PM* via conveyor *Con3* and buffer *B8*. Once the painting operation is finished, the part is passed back through the same sequence by which it arrived. From buffers *B6* and *B7*, the two different parts are passed to the machine *AM* for finishing.

The machines *Con2*, *Robot*, *Lathe*, *Con3*, *PM*, and *AM* can be thought of as components of the open-loop plant. The automata models for these machines are given Fig. 3.10.

The buffers *B2*, *B4*, *B6*, *B7*, and *B8* can be thought of as the component specifications for the system where it is desired that the buffers not underflow or overflow. The automata models for these machines are given Fig. 3.11. In these automata odd labels represent controllable events and even labels represent uncontrollable events. Furthermore, all automata have the same event set, though only relevant events are pictured. If a set of traditional modular supervisors are constructed in the sense of [8], it is interesting to note that their conjunction will result in blocking.

Following Algorithm 3.1 of Section 3.1, the first step is to choose an initial spec-

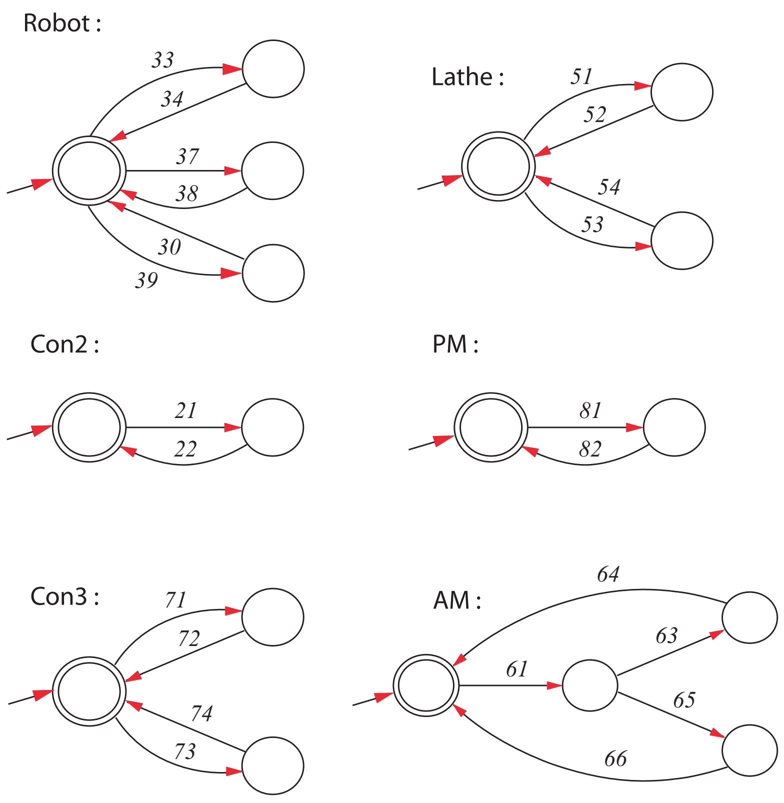

Figure 3.10: Automata models of each machine in the FMS example

ification. Employing the heuristic approach presented in Algorithm 3.18, we choose *B4* to be the first specification since it minimizes the following quantity, where the associated subplant is composed of components *Robot* and *Lathe*.

$$\frac{\left|\left|P_1(\overline{K_{spec,1}} \| L''_{m,1})\right|\right|}{\left|\left|\overline{K_{spec,1}}\right|\right| \times \left|\left|L''_{m,1}\right|\right|} = \frac{\left|\left|B4 \| Robot \| Lathe\right|\right|}{\left|\left|B4\right|\right| \times \left|\left|Robot \| Lathe\right|\right|} = 0.375$$

According to Algorithm 3.1, the next step is to abstract the plant submodules associated with our first specification. Since these plant components do not have any relevant events that are not relevant to the current or remaining specifications, the projection P_1 does not abstract away any relevant events. The resulting supervisor language for this module $\hat{K}'_{1,a}$ is generated by an automaton with 9 states and 10 transitions.

Moving up to the next level of the hierarchy, we again employ the heuristics of Algorithm 3.18 to choose the next specification. Of the remaining specifications, *B2*, *B6*, and *B7* all share relevant events with the first module and hence are considered to be neighbors. Of these, we choose *B7* to be our next specification since it minimizes the following expression. The "plant" for this module consists of the controlled

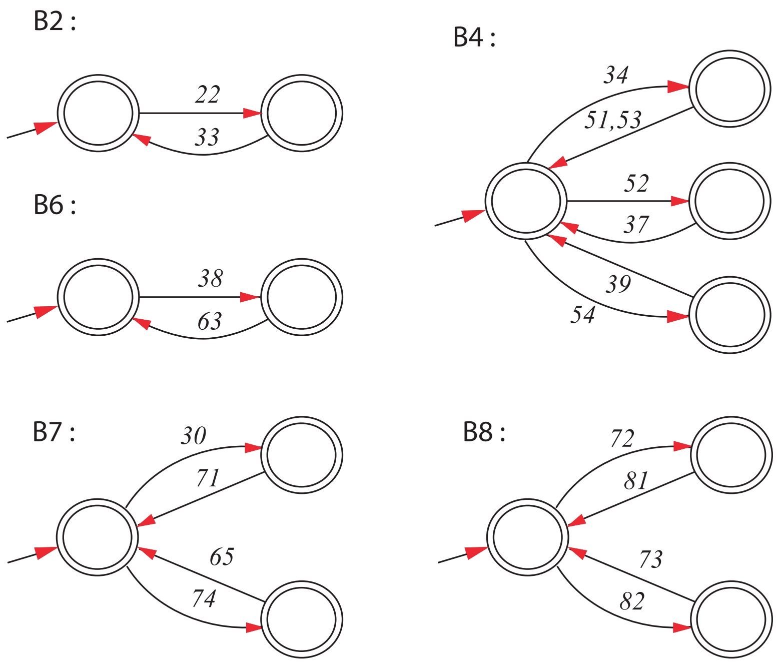

Figure 3.11: Automata models of each buffer in the FMS example

system from the first level $\hat{K}'_{1,a}$, along with subplants *AM* and *Con3*.

$$\psi_2 = \frac{||P_2(\overline{K_{spec,2}}\|\hat{K}'_{1,a}\|L''_{m,2})||}{||\overline{K_{spec,2}}|| \times ||\hat{K}'_{1,a}|| \times ||L''_{m,2}||} = \frac{||B7\|P_2(\hat{K}'_{1,a})\|P_2(AM)\|Con3||}{||B7|| \times ||\hat{K}'_{1,a}|| \times ||AM\|Con3\ ||} \approx 0.296$$

At this level of the hierarchy, the set of events $\{34, 37, 39, 51, 52, 53, 54, 64, 66\}$ are not relevant to any of the remaining specifications and hence can be considered for abstraction. It turns out that the events 34, 52, 54, 64, and 66 can be projected away without causing a violation of the associated L_m-observer properties. This set of events leads to a reduction in state size of the automata used for generating $\hat{K}'_{1,a}$ and *AM*. None of these events, however, are relevant to *B7* or *Con3* and hence P_2 erases only irrelevant events from the languages generated by these automata. The resulting supervisor for this “plant” and specification $\hat{K}_{2,a}$ can be generated by an automaton with 64 states and 189 transitions.

For the next specification, we choose buffer *B8* since it minimizes the quantity ψ_3. The plant for this specification will consist of the controlled subsystem from the previous level $\hat{K}_{2,a}$ and the machine *PM*. Of those events that are not relevant to *B8* or any of the remaining specifications, 30 and 74 can be projected away without causing a violation of the associated L_m-observer properties. The resulting supervisor language for this abstracted plant and specification $\hat{K}_{3,a}$ can be generated by an automaton with 110 states and 315 transitions.

Moving up another level of hierarchy, we choose buffer *B6* to be the next specification since it minimizes the quantity ψ_4. For this specification, the plant will simply consist of $\hat{K}_{3,a}$ since all the subplants relevant to *B6* were already included in constructing earlier supervisor languages. At this point, the only events that are not relevant to *B6* or the other remaining specification *B2* that do not cause a violation of the associated L_m-observer property are 71, 72, 81, and 82. The resulting supervisor language for this module $\hat{K}_{4,a}$ can be generated by an automaton with 71 states and 149 transitions.

Now addressing the final specification *B2*, the plant for this module will consist of the controlled subsystem from the previous level $\hat{K}_{4,a}$ and the only remaining plant component *Con2*. Since there are no further specifications left, all events not relevant to *B2* are eligible to be considered for abstraction. It turns out, however, that only events 37 and 38 can be abstracted without causing a violation of the L_m-observer property. The resulting supervisor language $\hat{K}_{5,a}$ for this module can be generated by an automaton with 165 states and 435 transitions. This is the largest automaton used to generate any of the modular supervisor languages. Also, construction of this supervisor language required building an automaton with 220 states and 609 transitions in an intermediate step. The details of each step of this example are presented in Table 3.1.

For the purposes of comparison, the monolithic supervisor has 2256 states and 7216 transitions, while the composition of all plant and specification components leads to an automaton with 13,248 states and 46,424 transitions. These numbers give some indication of the reduction in complexity offered by this approach to supervisory control. A drawback of this modular approach, however, is that it results in more restrictive control. In this example, the modular solution is more restrictive than the monolithic solution, though both solutions allow the FMS to process a maximum of six pieces at a given time.

The modular solution with the buffers addressed in the order *B4* $\rightarrow$ *B7* $\rightarrow$ *B8* $\rightarrow$ *B6* $\rightarrow$ *B2* presented above will be referred to as FMS Modular 1 in the complexity discussion presented in Section 3.4.4. If the specifications had rather been addressed in numerical order, *B2* $\rightarrow$ *B4* $\rightarrow$ *B6* $\rightarrow$ *B7* $\rightarrow$ *B8*, then a different set of modular supervisors would have been arrived at. This alternate solution, which we will

Table 3.1: Application of Algorithm 3.1 to FMS example

Step	Language Constructed	States (Transitions)	Notes
1	$\mathcal{L}(B4)$	4(7)	choose first specification, $\psi_1 = 0.375$
	$\mathcal{L}(Robot)$	4(6)	associated plant components
	$\mathcal{L}(Lathe)$	3(4)	
2			no relevant events can be projected away
3	$L'_{1,a}$	12(34)	plant: $L'_{1,a} = \mathcal{L}(Robot) \Vert \mathcal{L}(Lathe)$
	$K'_{1,a}$	18(23)	allowable language: $K'_{1,a} = \mathcal{L}(B4) \Vert L'_{1,a}$
4	$\hat{K}'_{1,a}$	9(10)	supervisor language: $\hat{K}'_{1,a} = \sup\mathcal{C}(K'_{1,a}, L'_{1,a})$
5	$P_1^{-1}(\hat{K}'_{1,a})$		does not have to be performed in practice
6	$\mathcal{L}(B7)$	3(4)	move to next level of hierarchy, $\psi_2 \approx 0.296$
	$\mathcal{L}(AM)$	4(5)	associated plant components
	$\mathcal{L}(Con3)$	3(4)	
2	$P_2(\hat{K}'_{1,a})$	6(7)	{34,52,54} projected away
	$P_2(\mathcal{L}(AM))$	2(3)	{64,66} projected away
3	$L''_{2,a}$	6(17)	$L''_{2,a} = P_2(\mathcal{L}(AM)) \Vert \mathcal{L}(Con3)$
	$L'_{2,a}$	36(144)	plant: $L'_{2,a} = P_2(\hat{K}'_{1,a}) \Vert L''_{2,a}$
	$K'_{2,a}$	96(294)	allowable language: $K'_{2,a} = \mathcal{L}(B7) \Vert L'_{2,a}$
4	$\hat{K}'_{2,a}$	64(189)	supervisor language: $\hat{K}'_{2,a} = \sup\mathcal{C}(K'_{2,a}, L'_{2,a})$
5	$P_2^{-1}(\hat{K}'_{2,a})$		does not have to be performed in practice
6	$\mathcal{L}(B8)$	3(4)	move to next level of hierarchy, $\psi_3 \approx 0.547$
	$\mathcal{L}(PM)$	2(2)	associated plant component
2	$P_3(\hat{K}'_{2,a})$	50(149)	{30,74} projected away
3	$L''_{3,a}$	2(2)	$L''_{3,a} = \mathcal{L}(PM)$
	$L'_{3,a}$	100(398)	plant: $L'_{3,a} = P_3(\hat{K}'_{2,a}) \Vert L''_{3,a}$
	$K'_{3,a}$	210(625)	allowable language: $K'_{3,a} = \mathcal{L}(B8) \Vert L'_{3,a}$
4	$\hat{K}'_{3,a}$	110(315)	supervisor language: $\hat{K}'_{3,a} = \sup\mathcal{C}(K'_{3,a}, L'_{3,a})$
5	$P_3^{-1}(\hat{K}'_{3,a})$		does not have to be performed in practice
6	$\mathcal{L}(B6)$	2(2)	move to next level of hierarchy, $\psi_4 \approx 0.242$
2	$P_4(\hat{K}'_{3,a})$	40(99)	{71,72,81,82} projected away
3	$L''_{4,a}$	–	no new subplant components
	$L'_{4,a}$	40(99)	plant: $L'_{4,a} = P_4(\hat{K}'_{3,a})$
	$K'_{4,a}$	80(170)	allowable language: $K'_{4,a} = \mathcal{L}(B6) \Vert L'_{4,a}$
4	$\hat{K}'_{4,a}$	71(149)	supervisor language: $\hat{K}'_{4,a} = \sup\mathcal{C}(K'_{4,a}, L'_{4,a})$
5	$P_4^{-1}(\hat{K}'_{4,a})$		does not have to be performed in practice
6	$\mathcal{L}(B2)$	2(2)	move to next level of hierarchy
	$\mathcal{L}(Con2)$	2(2)	associated plant component
2	$P_5(\hat{K}'_{4,a})$	55(119)	{37,38} projected away
3	$L''_{5,a}$	2(2)	$L''_{5,a} = \mathcal{L}(Con2)$
	$L'_{5,a}$	110(348)	plant: $L'_{5,a} = P_5(\hat{K}'_{4,a}) \Vert L''_{5,a}$
	$K'_{5,a}$	220(609)	allowable language: $K'_{5,a} = \mathcal{L}(B2) \Vert L'_{5,a}$
4	$\hat{K}'_{5,a}$	165(435)	supervisor language: $\hat{K}'_{5,a} = \sup\mathcal{C}(K'_{5,a}, L'_{5,a})$
5	$P_5^{-1}(\hat{K}'_{5,a})$		does not have to be performed in practice
6			no further subsystems left
7			procedure is finished

refer to as FMS Modular 2, actually results in smaller automata being constructed. Specifically, the largest automaton that generates a modular supervisor language has

106 states and 270 transitions, and the largest intermediate automaton that is constructed in the overall process has 210 states and 516 transitions. It turns out these two modular solutions produce the same behavior, though this will not be the case in general. These examples give some indication of how difficult it is to determine a "good" ordering with which to address the specifications.

3.4.3 Automated Guided Vehicle (AGV) example

We will now apply the supervisor construction approach of this chapter to a system of automated guided vehicles (AGVs). The details of the component specification and plant modules employed here can be found in [77]. Specifically, there are five plant modules $\{AGV_1, AGV_2, AGV_3, AGV_4, AGV_5\}$ representing each of five AGVs. Also, there are a total of eight component specifications. Five of these $\{Z_1, Z_2, Z_3, Z_4, IPS\}$ represent areas of the factory that can only be occupied by a single AGV at a time, thereby avoiding the collision of AGVs. The remaining three specifications $\{WS_1, WS_2, WS_3\}$ represent workstations and dictate the order in which parts are unloaded from and loaded to the AGVs. The example presented in [77] is modified from its original form in [29]. The monolithic supervisor built for this system is represented by an automaton that has 4406 states and 11,338 transitions. Furthermore, when all component specifications and plant modules are composed into a single automaton, that automaton has 22,784 states and 67,520 transitions. Note that traditionally constructed modular supervisors are pairwise nonconflicting, but block when all are acting in conjunction.

One possible solution employing our approach addresses the specifications in the order $Z_1 \rightarrow IPS \rightarrow Z_2 \rightarrow WS_2 \rightarrow Z_3 \rightarrow WS_3 \rightarrow Z_4 \rightarrow WS_1$. This ordering was chosen using Algorithm 3.18 introduced in Section 3.4.1. The largest resulting modular supervisor constructed by the IHSC approach is represented by an automaton that has 96 states and 250 transitions. The largest intermediate automaton built throughout this example has 144 states and 302 transitions.

The modular solution developed here turns out to be suboptimal. Specifically, it is more restrictive in terms of which areas of the factory can by occupied by AGVs at a given time than the monolithic solution which is maximally permissive. However, both the modular and monolithic solutions allow all three workstations to contain workpieces simultaneously. In the summary of numerical results presented in Table 3.2, this solution is referred to as AGV Modular 1. If the procedure of this paper uses the same ordering of specifications, but chooses to not abstract away some of the controllable events that could otherwise be erased, then it too provides optimal control. The largest modular supervisor in this case is significantly larger in that it is represented by an automaton with 576 states and 1700 transitions. Furthermore, the largest intermediate automaton built in this process has 864 states and 2100 transitions. In Table 3.2, this solution is referred to as AGV Modular 2.

3.4.4 Complexity discussion

Assessment of the complexity advantage provided by the approach of this chapter is difficult because in the worst case no abstraction is possible and the size of the state space grows at the same exponential rate as the monolithic approach. As a result, we cannot show a bound on complexity that is guaranteed to give an improvement as compared to the monolithic solution. However, in most cases significant abstraction is possible through the steps of this procedure resulting in a slowing of the exponential growth of the state space. The examples presented in this section specifically give some indication of the level of reduction achievable in terms of the size of the automata that are built. A table summarizing the results of each of the examples of this section is given below.

Table 3.2: Summary of Results for IHSC Approach

case	♯ states (♯ transitions) in largest supervisor	♯ states (♯ transitions) in largest intermediate automaton
FMS monolithic (opt)	*2256 (7216)*	*13,248 (46,424)*
FMS modular 1 (subopt)	165 (435)	220 (609)
FMS modular 2 (subopt)	106 (270)	210 (516)
AGV Monolithic (opt)	*4406 (11,338)*	*22,784 (67,520)*
AGV Modular 1 (subopt)	96 (250)	144 (302)
AGV Modular 2 (opt)	576 (1700)	864 (2100)

Apart from the size of the resulting automata, it is also necessary to consider the complexity of the additional operations required of the procedure of this paper, including the process of applying a natural projection and the process of finding a natural projection with the observer property. It is known that in the worst case the projection of a DES can lead to an exponential growth of the state space, thereby indicating the time complexity of the operation is at worst exponential. However, it has been demonstrated in [72] that a projection with the observer property is guaranteed to result in an abstracted system that is no larger than the original system. Furthermore, [72] demonstrates that under these conditions the complexity of generating the projected model is at worst polynomial in time. As far as finding a projection that possesses the observer property, [15] presents a polynomial time algorithm for finding an extension of the set of observable events for a projection such that the projection is an observer. Since the process of building a supervisory controller also has polynomial complexity, the complexity of the monolithic and modular approaches will both be dominated by a polynomial complexity operation applied to the largest intermediate automaton that has to be built. Therefore, the size of the largest intermediate automaton is a reasonable metric for the complexity of generating supervisory control for a given system. Furthermore, the size of the largest resulting supervisor provides a reasonable metric for the complexity of the implementation of the control scheme.

It should be noted that the algorithm of [15] in many cases finds a reasonably small extension of the observable event set, though it is not guaranteed to be minimal. This fact, along with the problem of ordering, indicates that it is not possible to find the minimal reduction of the state space. We can, however, use intuition and experience to find reasonably good reductions.

One way to further improve the complexity of our approach is to combine it with other techniques that exist in the literature. For example, the work of [18] provides results for certain commonly encountered types of systems. For these classes of systems, it is shown that conditions on certain modules indicate that they will be nonconflicting with the rest of the system. As such, these modules do not have to be considered when trying to guarantee nonblocking of the global system. Reduction in the number of components that must be considered directly results in less complexity since the size of the state space grows exponentially in the number of components. It is also possible that the approach of this chapter can be employed with more computationally efficient data structures than automata, like binary decision diagrams [3] and state tree structures [46]. In [46] and [62] different data structures are combined with variations of the supervisory control framework to improve complexity and understandability.

3.5 Chapter Summary

This chapter puts forth an incremental procedure for generating a set of modular supervisors that are nonconflicting by construction. The conjunction of the modular supervisors was shown to satisfy given specifications without blocking when natural projections with the L_m-observer property are employed. It is also shown that in many cases the set of modular supervisors provides suboptimal control. It is argued that this potential loss of optimality is worth the reduction in complexity this approach provides.

Examples were presented in Section 3.4, with our modular solution showing significant savings in size as compared to the monolithic solution. The approach to supervisor design presented here is most often useful for systems where the coupling between elements is well-distributed. If every specification of a system addresses every plant module, then this approach will likely provide little improvement.

Overall this approach to supervisor construction shares similarities with the work of [21] and [54], which were developed at approximately the same time and that verify nonconflict by incrementally constructing the global system using abstraction. The advantage of the IHSC approach is that it goes beyond just detecting nonconflict to actually construct modular supervisors that are nonconflicting.

The IHSC approach also shares similarities with another work that was developed around the same time [17]. This work similarly achieves safe, nonblocking modular control using an abstraction with the observer property to improve complexity. In [17], however, the authors construct an additional level of control to resolve conflict among the modular supervisors. The modular supervisors constructed by the IHSC approach are built so that they are nonconflicting by construction, hence an extra level of control is not necessary. This makes the resulting controllers marginally less complex to implement than those developed in [17]. Another disadvantage of the approach of [17] is that their process leads to more interaction between the modules, thereby limiting the amount of achievable reduction. For example, consider the partitioning of the FMS example shown in Fig. 1.4. Since the *Robot* subplant is shared between multiple modules, the approach of [17] cannot abstract away any of the structure of the *Robot* subplant until the conflict has been resolved between all the modules that share that component. With the IHSC approach, only those aspects of the structure of the *Robot* subplant that are relevant to the remaining specifications must be preserved. The IHSC approach is also able to achieve a greater reduction than [17] since it does not require that the abstraction be output-control-consistent. By foregoing this property, however, the optimality of control that the approach of [17] is able to achieve is sacrificed.

One way to improve the work of this chapter would be to relax some of the requirements of the IHSC approach. Namely, it could be worthwhile to explore whether or not the requirement that the specification languages be prefix-closed could be relaxed, or if reduced supervisors in the sense of [66] could somehow be employed. It would also be useful if the approach of this chapter could be applied to systems that do not have the structure of a product system. Looking in the opposite direction, it would be useful if this approach to supervisor construction could synthesize control that is maximally permissive by placing additional constraints on the abstraction we employ. A specific abstraction property that could be investigated would be the output-control-consistency property required by [17]. A final direction for this work would be to further explore heuristics for choosing the order in which specifications are addressed and for determining which events to abstract away.

CHAPTER 4

Equivalence-Based Conflict Resolution

Modular supervisory control is one approach for addressing the DES complexity problem that is limited in application since modular supervisors can have conflicting goals and hence can lead to blocking when working in conjunction. The Equivalence-Based Conflict Resolution (EBCR) approach of this chapter addresses this problem by proposing a methodology for constructing coordinating filters to resolve conflict among modular supervisors when it is present. This approach incrementally composes modular supervisors applying abstraction each time a new module is added. At each step if the resulting composition is blocking, then a filter is constructed to resolve the conflict.

This type of modular architecture with an additional level of coordinating control has been employed in previous work, but our approach is unique in that it is the first to employ a conflict-equivalent abstraction in the construction of the conflict-resolving filters. Most existing works on supervisor construction that employ abstraction use an observer-type abstraction. Employing a conflict-equivalent abstraction is advantageous because it is able to achieve a greater reduction in model size than an observer-type abstraction [49]. A drawback of a conflict-equivalent abstraction is that it can introduce nondeterminism into the model.

The work of this chapter, therefore, is developed for nondeterministic automata. We specifically propose a set of novel requirements on the conflict-resolving filter laws that, if met, guarantee safe nonblocking control when acting in conjunction with traditionally built modular supervisors. We then present one possible approach for constructing filters that meet these proposed requirements. The filter construction algorithm itself represents a significant contribution in that it generates a covering-based feedback law in the presence of nondeterminism that results in less restrictive control than can achieved by existing state-feedback approaches.

The outline of the rest of this chapter is as follows. Section 4.1 introduces some technical preliminaries. Section 4.2 provides a procedure for resolving conflict assuming that filters exist, while Section 4.3 proves that deterministic filters meeting the given language-based requirements provide safe nonblocking control. Section 4.4 then proposes a set of analogous state-based requirements that allow the deterministic filter laws to be represented by nondeterministic automata. These state-based requirements are then shown to be satisfied by construction for the covering-based approach to control introduced in Section 4.5. Section 4.6 applies these results to the FMS example and Section 4.7 summarizes the contributions of this chapter.

4.1 Supervisor Construction and Abstraction

In the EBCR approach of this chapter, supervisors will be built in the sense of [8] as shown in Definition 4.1. Let H_i be an automaton realization of a closed-loop subsystem $\mathcal{S}_i/G'_i$ consisting of a plant G'_i under the control of the supervisor $\mathcal{S}_i$. The supervisor synthesis of Definition 4.1 provides that the automaton H_i represents both the closed-loop behavior of the i^{th} module and its associated supervisor since $\mathcal{S}_i/G'_i = H_i \| G'_i = H_i$.

Definition 4.1.

$$\begin{aligned}
G'_i &= \quad \|_{j \in J_i} G_j, \text{ where:} \\
J_i &= \{j \in \{1, \ldots, n\} \mid \Sigma_{rel}(G_j) \cap \Sigma_{rel}(E_i) \neq \emptyset\} \\
\mathcal{L}_m(H_i) &= \sup \mathcal{C}(\mathcal{L}(E_i) \cap \mathcal{L}_m(G'_i), \mathcal{L}(G'_i)) \\
\mathcal{L}(H_i) &= \overline{\mathcal{L}_m(H_i)} \ \diamond
\end{aligned}$$

In the above, the definition of the allowable language as $\mathcal{L}(E_i) \cap \mathcal{L}_m(G'_i)$ results in a *nonmarking* supervisor. That is, the supervisor does not affect the marking of the uncontrolled plant. Additionally, given that the uncontrolled plant is nonblocking, $\overline{\mathcal{L}_m(G'_i)} = \mathcal{L}(G'_i)$, this formulation will result in a nonblocking closed-loop subsystem. The results of this paper will be stated in terms of the closed-loop modules, H_i. Since the results do not depend on the supervisor employed, supervisors can be constructed in a manner different from Definition 4.1. If there are plant modules G_j that do not share relevant events with any of the specifications E_i, they are treated as closed-loop modules H_i on their own, without any additional supervision. Let $\{H_1, H_2, \ldots, H_q\}$ be the resulting set of automata representing the closed-loop modules.

As the modular supervisors are composed, those events that are not relevant to any of the remaining supervisors can be "hidden," that is, they can be replaced by the silent event τ. Hiding these events assists in achieving a greater reduction in model size. Specifically, the abstraction that will be employed preserves conflict properties. The notion of conflict-equivalence was introduced earlier in Definition 2.6. Conflict-equivalent abstraction in general provides a greater reduction in the state size of a model than either an observation-equivalent abstraction or a projection with the observer property [49]. A drawback of a conflict-equivalent abstraction is that it is not as straightforward to implement; it is implemented via heuristics and a select set of rules [19] [21]. Also, a unique minimal reduction does not exist in general.

In this paper we will employ the notation G_a to represent a conflict-equivalent abstraction of the automaton G. The abstracted automaton will specifically be generated in the following manner:

Algorithm 4.2. *Conflict-Equivalent Abstraction*
Step 1: Given an automaton G, "hide" those events in the set Σ_h. One approach for constructing the set Σ_h in the context of the approach of this paper is presented within Algorithm 4.6 of Section 4.2. These events are hidden by replacing their occurrences in the automaton G by the silent event τ resulting in an intermediate automaton G'.
Step 2: Apply the conflict equivalence preserving rules that will be identified in Section 4.3.2 and are taken from [19] [21] to G'. The result of these rules is the reduced automaton G_a. $\diamond$

Remark 4.3. By construction, the intermediate automaton G' and the abstraction G_a are conflict equivalent per Definition 2.6. However, the original automaton G and the reduced automaton G_a are only guaranteed to be conflict equivalent with respect to automata that do not have any relevant events that were hidden in the process of generating G'. In the remainder of this paper, we will only hide events in a manner consistent with this fact, that is, events are only hidden if they are not relevant to any remaining automata. In a slight abuse of notation, we will still write that $G \simeq_{conf} G_a$. Note also that G and G_a are consistent with respect to the property of blocking. $\diamond$

The software tool `Supremica` can be employed for generating conflict-equivalent abstractions [1]. Since each automaton in this dissertation has the same event set Σ_τ, it is implied that any hidden events are self-looped at every state of the resulting automaton. However, we will not in general picture all of these self-looped transitions. Example 4.4 demonstrates a conflict-equivalent abstraction.

Example 4.4. Consider automaton G in Fig. 4.1 where event f is not relevant to any other automata. Since f is "local" to G, we can hide it by replacing all occurrences of f by the silent event τ resulting in a new automaton G'. In G', states 1 and 2 are not observation equivalent because state 1 has the observed continuation bc while state 2 does not. States 1 and 2, however, can be merged to achieve the conflict-equivalent automaton G_a. We will consider the abstraction G_a to have the same alphabet as G, namely Σ_τ, but will not picture an event (except τ) if it is not relevant to the automaton. Therefore, one can imagine that G_a has the event f self-looped at every state. A consequence of this abstraction is that G_a is nondeterministic. ⋄

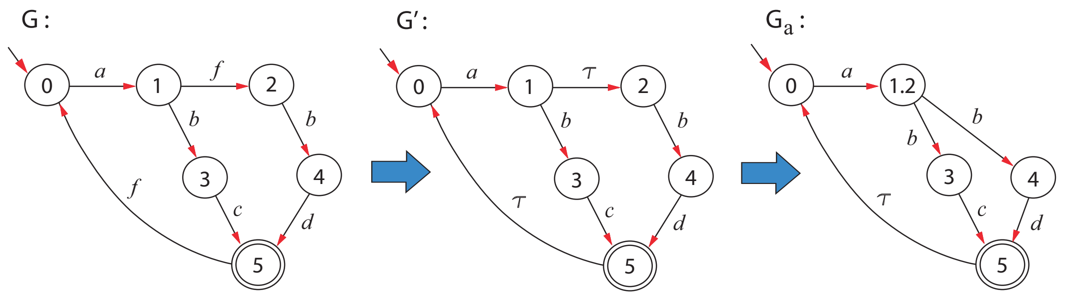

Figure 4.1: Illustrative example of a conflict-equivalent abstraction

In order to make the conflict-equivalent abstraction useful, we need to show that it is preserved under the synchronous composition operation $\|$. This result follows from Proposition 4.5 which is a reformulation of a result from [50].

Proposition 4.5. *Let G, G_a, and H be automata. Also assume that any events hidden in the process of generating G_a (Algorithm 4.2) are not relevant to H, that is, $\Sigma(H) \cap \Sigma_h = \emptyset$. If $G \simeq_{conf} G_a$ then $G\|H \simeq_{conf} G_a\|H$. See Remark 4.3.*

The above proposition can be used to show that if no relevant events shared between G_1 and G_2 are hidden, then $G_1\|G_2 \simeq_{conf} G_{1,a}\|G_{2,a}$. Analogous results can also be shown for conflict-equivalent languages.

4.2 Incremental Conflict Resolution Using Filters

The overall goal of this chapter is to generate a set of modular supervisors and conflict-resolving filters that control the behavior of a given plant so that it satisfies a set of specifications in a nonblocking manner. A further goal is to limit the computational complexity of constructing the supervisors and filters. In this section we describe a procedure by which conflict is incrementally detected and resolved among modular supervisors. Requirements on the conflict-resolving filters are presented in Section 4.3 and Section 4.4, while an algorithm for constructing the filters is presented in Section 4.5.

In the following procedure, it will be assumed without loss of generality that the modules H_i are addressed sequentially, that is, each pass through the procedure the next module added to the composition has index $i+1$. Additionally, the composition for which the last module added was H_i will be denoted H_i' where the index i, therefore, indicates both the current composition of components as well as the most recently added module. Since a filter is constructed to resolve conflict in a given composition H_i' only when the composition is blocking, the index j on filters $H_{\mathit{filt},j}$ increments independently.

Algorithm 4.6. *Conflict Resolution*
Step 1: Build modular supervisors according to Definition 4.1. Note that the supervisors may be constructed in other ways as long as the closed-loop automaton H_i is employed in the subsequent steps. Any plant components that are not addressed by a specification are treated as additional closed-loop modules. Let $\{H_1, H_2, \ldots, H_q\}$ represent the resulting set of closed-loop modules. Section 4.3.4 identifies a special case where the full closed-loop automaton H_i need not be employed.
Step 2: For each supervised subsystem H_i, generate a conflict-equivalent abstraction $H_{i,a}$ according to Algorithm 4.2. The set of hidden events in this step corresponds to those events relevant to only a single H_i, that is, $\Sigma_h = \Sigma - \bigcup_{i \neq j}(\Sigma_{rel}(H_i) \cap \Sigma_{rel}(H_j))$.
Step 3: Choose an abstracted subsystem $H_{1,a}$ with which to begin the procedure. Let $H_{i,a}' = H_{1,a}$ where $i = 1$ is the index for the individual closed-loop modules. Also initialize the index for the filters, $j = 1$.
Step 4: Choose one of the remaining abstracted subsystems, $H_{i+1,a}$, to compose with $H_{i,a}'$. This operation is performed via synchronous composition, $H_{i+1}' = H_{i,a}' \| H_{i+1,a}$.

Step 5: If the composition H'_{i+1} is nonblocking, skip to Step 7, otherwise proceed to Step 6.

Step 6: At this point a coordinating filter law $\mathcal{H}_{\mathit{filt},j} : \mathcal{L}(H'_{i+1}) \longrightarrow 2^{\Sigma}$ must be generated to resolve the detected conflict in the preceding blocking composition. Otherwise stated, $\mathcal{H}_{\mathit{filt},j}$ is built so that the controlled system $\mathcal{H}_{\mathit{filt},j}/H'_{i+1}$ is nonblocking. Specific requirements for this filter will be presented in Section 4.3 and Section 4.4, and an approach for its construction will be proposed in Section 4.5. After the filter is constructed, increment the filter index j.

Step 7: If all controlled subsystems have been addressed, then $i = q$ and the procedure is finished. Otherwise, more abstraction is performed according to Algorithm 4.2 and this overall procedure is repeated beginning at Step 4. The abstraction is performed in order to take advantage of the fact that some events are no longer relevant to any remaining abstracted subsystems and hence can now be hidden. More precisely, the set of hidden events becomes

$$\Sigma_h \leftarrow \Sigma_h \cup (\Sigma - \bigcup_{k>i+1} \Sigma_{rel}(H_{k,a})) \tag{4.1}$$

and $H'_{i+1} = H'_{i,a} \| H_{i+1,a}$ is abstracted to generate $H'_{i+1,a}$. The index i is then incremented before returning to Step 4. $\diamond$

The process in Step 4 to Step 7 of abstracting and composing subsystems and adding filters as necessary to prevent blocking is repeated until there are no more subsystems remaining. The work of [21] offers a sizable survey of heuristics for determining the ordering with which subsystems are addressed. The end result of this procedure is a set of filters that act in conjunction with the set of modular supervisors. If the controlled subsystems are nonconflicting on their own, no filters are needed. If a filter is generated that is the empty automaton, it is possible that a nonempty filter can be found by abstracting away fewer details of the controlled subsystems, that is, by making Σ_h smaller. A nonempty solution could also be found by addressing the modules in a different order. This approach to conflict resolution is unique in that it is the first to employ conflict-equivalent abstraction in the construction of coordinating filters for conflict resolution. The details of this coordinating level of control will be discussed in the following three sections. Section 4.3 provides a set of language-based requirements that are sufficient to guarantee safe nonblocking control when the filters are represented by deterministic automata. Section 4.4 then provides an analogous set of state-based requirements that allow the deterministic filter laws to be represented by possibly nondeterministic automata, while Section 4.5 introduces a methodology for constructing filters that satisfy these state-based requirements.

4.3 Language-Based Filter Requirements

In the preceding section, Algorithm 4.6 was presented for incrementally resolving conflict among a set of supervised subsystems. In this section we will provide a set of language-based conditions on these deterministic filter laws and prove they are sufficient to provide safe nonblocking control when acting in conjunction with traditionally built modular supervisors. Within this process, we will examine the details of how a conflict-equivalent abstraction is generated. At the end of this section, we will also discuss how the closed-loop modules constructed as part of Algorithm 4.6 can be reduced to further improve the overall complexity of our approach.

In Algorithm 4.6, each filter law $\mathcal{H}_{\mathit{filt},j}$ is built with respect to a blocking composition of abstracted automata that have preceded it. Here we will denote the associated blocking composition $B_{j,a}$. In the proofs that follow, we will assume that the control required by each filter law $\mathcal{H}_{\mathit{filt},j}$ is realized by a deterministic, nonblocking automaton $H_{\mathit{filt},j}$ and applied via synchronous composition, that is, $\mathcal{H}_{\mathit{filt},j}/B_{j,a} = H_{\mathit{filt},j} \| B_{j,a}$, therefore,

$$B_{j,a} = (H_{\mathit{filt},j-1} \| \ldots (H_{\mathit{filt},1} \| H_{1,a} \| H_{2,a})_a \ldots)_a \| H_{i_j,a}$$

Furthermore, we will prove that deterministic filter automata meeting the following three requirements ($R1$-$R3$) will provide safe nonblocking control when acting in conjunction with the modular supervisors.

Language-based filter requirements

$R1$) $H_{\mathit{filt},j} \| B_{j,a}$ is nonblocking

$R2$) $\mathcal{L}(H_{\mathit{filt},j})$ is language controllable w.r.t $\mathcal{L}(B_{j,a})$

$R3$) $\Sigma_{rel}(H_{\mathit{filt},j}) \cap \Sigma_h = \emptyset$

In the above, requirement $R3$ is meant to prevent a given filter law from trying to affect the occurrence of events that have been hidden. Since the set Σ_h changes at each iteration, it is implicit in $R3$ that Σ_h be the set taken at the time the filter $H_{\mathit{filt},j}$ is constructed. We must now prove that these requirements are sufficient for guaranteeing safe nonblocking control can be realized.

In Section 4.4 we will present new state-based requirements that will allow our deterministic filter laws $\mathcal{H}_{\mathit{filt},j}$ to be realized by possibly nondeterministic automata. These state-based requirements are also shown to be satisfied by filters constructed according to the algorithm of Section 4.5.

4.3.1 Nonblocking

Recall that the conjunction of modular supervisors satisfies the global specification E. Since the addition of filters only serves to further restrict the behavior of the system, the conjunction of filters and modular supervisors also provides safety. We will now demonstrate global nonblocking. In the following we will assume sequential ordering of the automata without loss of generality.

Theorem 4.7. *Let H_i be the automaton representing the behavior of the i^{th} controlled subsystem where $i \in \{1, \ldots, q\}$. Also let there be filter automata $H_{\mathit{filt},j}$, $j \in \{1, \ldots, k\}$, constructed according to Algorithm 4.6 and satisfying requirements $R1$ and $R3$. The conjunction of supervised subplants and filters $H_{\mathit{filt},1} \| \ldots \| H_{\mathit{filt},k} \| H_1 \| \ldots \| H_q$ is then nonblocking.*

Proof.

• By the procedure of Section 4.2, automata are incrementally abstracted and composed. Assume the first two abstracted automata do not conflict. Therefore, $H_{1,a} \| H_{2,a}$ is nonblocking. Since $\Sigma_{rel}(H_1) \cap \Sigma_{rel}(H_2) \subseteq (\Sigma - \Sigma_h)$, $H_{1,a} \| H_{2,a} \simeq_{conf} H_1 \| H_2$ by Proposition 4.5. Therefore, $H_1 \| H_2$ is also nonblocking since conflict equivalence preserves blocking properties.

• Assume the addition of a third automaton also does not cause conflict, then $(H_{1,a} \| H_{2,a})_a \| H_{3,a}$ is nonblocking. Noting again that conflict equivalence holds across synchronous composition when shared relevant events are not abstracted away, $(H_{1,a} \| H_{2,a})_a \| H_{3,a}$ is conflict equivalent to $H_{1,a} \| H_{2,a} \| H_3$. Since those events made silent in the generation of $H_{1,a}$ and $H_{2,a}$ are not relevant to any of the remaining subsystems, Proposition 4.5 provides that $H_{1,a} \| H_{2,a} \| H_3 \simeq_{conf} H_1 \| H_2 \| H_3$. Furthermore, since equivalence relations are transitive, $(H_{1,a} \| H_{2,a})_a \| H_{3,a} \simeq_{conf} H_1 \| H_2 \| H_3$. Therefore, $H_1 \| H_2 \| H_3$ is also nonblocking.

• Assume for the first i_1 automata addressed, where $1 \leq i_1 \leq q$, no conflict is detected. Therefore, the resulting nested composition given below is nonblocking.

$$H'_{i_1} = ((\ldots((H_{1,a} \| H_{2,a})_a \| H_{3,a})_a \| \ldots)_a \| H_{i_1-1,a})_a \| H_{i_1,a} \tag{4.2}$$

Following the logic above, the expression in equation (4.2) is conflict equivalent to $H_1 \| H_2 \| \ldots \| H_{i_1}$. Therefore, $H_1 \| H_2 \| \ldots \| H_{i_1}$ is nonblocking since the expression in equation (4.2) is.

• If $i_1 = q$, then there are no filters and we are done. Otherwise, the filter $H_{\mathit{filt,1}}$ is needed to resolve the conflict in $H'_{i_1,a} \| H_{i_1+1,a}$, where $H'_{i_1,a}$ is the further abstraction of the expression in equation (4.2). By $R1$, $H_{\mathit{filt,1}} \| H'_{i_1,a} \| H_{i_1+1,a}$ is nonblocking. By Proposition 4.5 and the above, $H'_{i_1,a} \| H_{i_1+1,a}$ is conflict equivalent to $H_1 \| \ldots \| H_{i_1+1}$. Therefore, $H_{\mathit{filt,1}} \| H'_{i_1,a} \| H_{i_1+1,a}$ is conflict equivalent to $H_{\mathit{filt,1}} \| H_1 \| \ldots \| H_{i_1+1}$ by Proposition 4.5 since no events in Σ_h at this point are relevant to $H_{\mathit{filt,1}}$ by $R3$. Therefore, $H_{\mathit{filt,1}} \| H_1 \| \ldots \| H_{i_1+1}$ is nonblocking since $H_{\mathit{filt,1}} \| H'_{i_1,a} \| H_{i_1+1,a}$ is.

• Let automata $H_{i_1+2,a}$ through $H_{i_2,a}$ be added such that the following expression is nonblocking, where $i_1 + 2 \leq i_2 \leq q$.

$$H'_{i_2} = (\ldots((H_{\mathit{filt,1}} \| H'_{i_1,a} \| H_{i_1+1,a})_a \| H_{i_1+2,a})_a \| \ldots)_a \| H_{i_2,a} \tag{4.3}$$

Following the logic employed above, it can then be shown that the expression in equation (4.3) is conflict equivalent to $H_{\mathit{filt,1}} \| H_1 \| \ldots \| H_{i_1+1} \| H_{i_1+2} \| \ldots \| H_{i_2}$, which is in turn nonblocking also.

• If $i_2 = q$, then there are no more filters and we are done. Otherwise, the filter $H_{\mathit{filt,2}}$ is needed to resolve the conflict in the composition $H'_{i_2,a} \| H_{i_2+1,a}$, where $H'_{i_2,a}$ is the further abstraction of the expression in equation (4.3). By $R1$, $H_{\mathit{filt,2}} \| H'_{i_2,a} \| H_{i_2+1,a}$ is nonblocking. By Proposition 4.5 and the above, $H'_{i_2,a} \| H_{i_2+1,a}$ is conflict equivalent to $H_{\mathit{filt,1}} \| H_1 \| \ldots \| H_{i_2}$. Therefore, $H_{\mathit{filt,2}} \| H'_{i_2,a} \| H_{i_2+1,a}$ is conflict equivalent to $H_{\mathit{filt,2}} \| H_{\mathit{filt,1}} \| H_1 \| \ldots \| H_{i_2}$ by Proposition 4.5 since no events in Σ_h at this point are relevant to $H_{\mathit{filt,2}}$ by $R3$. Therefore, $H_{\mathit{filt,2}} \| H_{\mathit{filt,1}} \| H_1 \| \ldots \| H_{i_2}$ is nonblocking since $H_{\mathit{filt,2}} \| H'_{i_2,a} \| H_{i_1+2,a}$ is.

• Repeating this process, supervised subsystems and filters are added to the composition until they have all been addressed. The resulting composition $H_{\mathit{filt,1}} \| \ldots \| H_{\mathit{filt,k}} \| H_1 \| \ldots \| H_q$ is, therefore, shown to be nonblocking. $\square$

4.3.2 Conflict-equivalence preserving rules

We now need to demonstrate that the control required of these filters is realizable. For a deterministic control law, this corresponds to demonstrating the language controllability condition of equation (2.1). In order to demonstrate this, we will require that our conflict-equivalent abstraction satisfies the following property, where $P_h : \Sigma^* \to (\Sigma - \Sigma_h)^*$ is the natural projection that erases those events that have been hidden.

$$P_h(\mathcal{L}(H)) = P_h(\mathcal{L}(H_a)) \tag{4.4}$$

In words, we need that the original and reduced automata generate the same projected languages. We will specifically demonstrate which rules of [19] [21] applied in Step 2 of Algorithm 4.2 achieve the property required by equation (4.4). We must first, however, introduce the following equivalence relation from [21]. This relation is employed in some of the reduction rules to follow.

Definition 4.8. [21] Let $G = (Q, \Sigma_\tau, \delta, q_0, Q_m)$ be an automaton. The binary relation $\sim_{inc} \subseteq Q \times Q$ is defined such that $q \sim_{inc} q'$ if:

$$q_0 \overset{\varepsilon}{\Rightarrow} q \iff q_0 \overset{\varepsilon}{\Rightarrow} q';$$
$$\forall p \in Q \text{ and } \forall \sigma \in \Sigma \ : \ p \overset{\sigma}{\Rightarrow} q \iff p \overset{\sigma}{\Rightarrow} q'. \ \diamond$$

The relation $\sim_{inc}$ defines two states as being equivalent if they are reached in the same observed manner. In a sense, this relation is dual to observation equivalence where states with the same observed future are equated. The following two rules from [21] employ the relation $\sim_{inc}$ to identify *conflict-equivalent states.* Two states are defined to be conflict equivalent if they have future behaviors that cannot be distinguished by conflict equivalence. The reduction of the automaton model is then achieved by merging conflict-equivalent states.

1) Active Events Rule: Two states that are equivalent with respect to $\sim_{inc}$ and have the same set of *active events* are conflict equivalent. The active event set of a state q is defined here to be those events $\sigma \in \Sigma$ for which there exists a string $t \in \Sigma_\tau^*$ such that $\delta(q, t)!$ and $P_\tau(t) = \sigma$.

2) Silent Continuation Rule: Two states that are equivalent with respect to $\sim_{inc}$ and from which states without outgoing τ transitions can be reached via a *nonempty* sequence of τ transitions are conflict equivalent.

Observation-equivalence provides another rule for identifying conflict-equivalent states since observation equivalence implies conflict equivalence [21].

3) Observation Equivalence Rule: Two states that are observation equivalent are also conflict equivalent.

The requirement presented in equation (4.4) can now be demonstrated for automata abstracted by applying the *Active Events Rule* and the *Silent Continuation Rule* based on their reliance on the binary relation $\sim_{inc}$.

Proposition 4.9. *Let there be two (possibly nondeterministic) automata H and H_a, where $H = (Q, \Sigma_\tau, \delta_h, q_0, Q_m)$ and H_a is an abstraction generated by Algorithm 4.2. If only the* Active Events Rule *and the* Silent Continuation Rule *are applied in the process of abstraction, then $P_h(\mathcal{L}(H_a)) = P_h(\mathcal{L}(H))$, where $P_h : \Sigma^* \to (\Sigma - \Sigma_h)^*$.*

Proof.
• Following the first step of Algorithm 4.2, let $H' = (Q, \Sigma_\tau, \delta'_h, q_0, Q_m)$ be the automaton generated by replacing those transitions of H that are in Σ_h by the silent event τ. It is then apparent that:

$$P_h(\mathcal{L}(H')) = P_h(\mathcal{L}(H)) \tag{4.5}$$

• Next, assume that H_a is generated from H' by first merging the single pair of distinct states $q, q' \in Q$, where $q \in \delta'_h(q_0, s)$, $q' \in \delta'_h(q_0, s')$, and $s, s' \in \Sigma_\tau^*$. See Fig. 4.2. Since it is assumed that equivalent states are identified by only the *Active Events Rule* and the *Silent Continuation Rule*, we then have that $q \sim_{inc} q'$.
• Let $l \in \Sigma_\tau^*$ be a string accepted by H'. We now must show that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Leftrightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$.

(Case 1)

• Let $l \in \Sigma_\tau^*$ be a string accepted by H' such that l does not pass through either of the states to be merged, that is, $s, s' \nleq l$. It logically follows that $P_\tau(l) \in \mathcal{L}(H') \Leftrightarrow P_\tau(l) \in \mathcal{L}(H_a)$ and further that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Leftrightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$.

(Case 2)

• We will now examine all strings $l, l' \in \Sigma_\tau^*$ accepted by H' that do pass through q and q' respectively. If l passes through the states q and q' more than once, then let $s \leq l$ be the prefix of l that last reaches q. Likewise, let $s' \leq l'$ be the prefix of l' that last reaches q'.
• Therefore, let $l = st$ where $q \in \delta(q_0, s)$. Furthermore, $\nexists v \in \Sigma_\tau^* - \{\varepsilon\}$ such that $sv \leq l$ and $q \in \delta'_h(q_0, sv)$ or $q' \in \delta'_h(q_0, sv)$. Likewise, let $l' = s't'$ where $q' \in \delta(q_0, s')$. Also, $\nexists v \in \Sigma_\tau^* - \{\varepsilon\}$ such that $s'v \leq l'$ and $q \in \delta'_h(q_0, s'v)$ or $q' \in \delta'_h(q_0, s'v)$.
• In the following we will examine the string l. To examine l', the logic is the same just with relabeling.

• ($\subseteq$) Merging q and q' means that if the strings st and $s't'$ are accepted by H', then st, st', $s't$, and $s't'$ are accepted by H_a (see Fig. 4.2). Furthermore, recall that $q \sim_{inc} q'$. Referring to Definition 4.8, this means that either $P_\tau(s) = P_\tau(s') = \varepsilon$, or that $s = ru$ and $s' = ru'$ where $p = \delta'_h(q_0, r)$ and $P_\tau(u) = P_\tau(u') = \sigma$. In either case, $P_\tau(s) = P_\tau(s')$. It then follows that:

$$P_\tau(st) = P_\tau(s't) \text{ and } P_\tau(s't') = P_\tau(st') \tag{4.6}$$

Therefore, $P_\tau(st), P_\tau(s't') \in \mathcal{L}(H') \Rightarrow P_\tau(st) = P_\tau(s't), P_\tau(s't') = P_\tau(st') \in \mathcal{L}(H_a)$. Since $l = st$, this also means that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Rightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$.

• ($\supseteq$) Again by merging q and q', we have that the strings st, st', $s't$, and $s't'$ are accepted by H_a. This, therefore, implies that at least two elements of the set $\{st, st', s't, s't'\}$ are accepted by H' where each of the strings s, s', t, and t' is used in at least one of the strings accepted from the set. Employing equation (4.6) again, we then have that $P_\tau(st), P_\tau(st'), P_\tau(s't), P_\tau(s't') \in \mathcal{L}(H_a) \Rightarrow P_\tau(st) = P_\tau(s't), P_\tau(s't') = P_\tau(st') \in \mathcal{L}(H')$. Also since $l = st$, $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a)) \Rightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H))$.

• Taking Case 1 and Case 2 together, since it is true $\forall l$ accepted by H' that $P_h(P_\tau(l)) \in P_h(\mathcal{L}(H')) \Leftrightarrow P_h(P_\tau(l)) \in P_h(\mathcal{L}(H_a))$, we have that $P_h(\mathcal{L}(H')) = P(\mathcal{L}(H_a))$ if a single pair of states have been merged.

• If we further abstract H_a by merging another pair of states that satisfy the binary relationship $\sim_{inc}$, we can repeat the logic above. Therefore, we can show that $P_h(\mathcal{L}(H')) = P_h(\mathcal{L}(H_a))$ in general. This in conjunction with equation (4.5), therefore, proves our ultimate desired result. $\square$

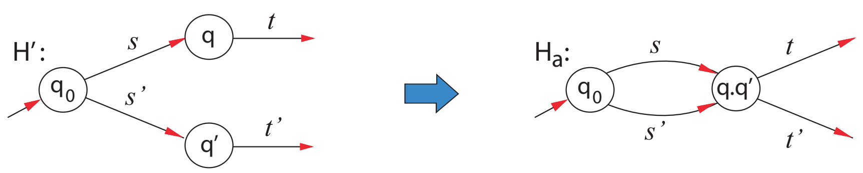

Figure 4.2: Example of an abstraction using an equivalence relation

Similar logic to the above proposition can be employed to show that for two observation equivalent automata H and H_a, equation (4.4) also holds. This fact is noted by [64]. Other rules found in [19] can be derived based on *Rules 1-3* mentioned previously, therefore, they will also satisfy equation (4.4). In the EBCR approach, we will only apply rules which derive from these three rules. If other rules that meet equation (4.4) can be identified, then they can be employed in our application as well.

Now recall that a reduced automaton H_a has replaced all hidden events with the τ event then added self-loops at every state for each hidden event, therefore, none of the events that have been made silent are relevant to H_a. This in turn means that $P_h^{-1}(P_h(\mathcal{L}(H_a))) = \mathcal{L}(H_a)$. Here P_h^{-1} is an inverse projection which expands the alphabet from $(\Sigma - \Sigma_h)$ to Σ. In terms of automata, P_h^{-1} adds self loops at every state for all events in Σ_h. This logic along with equation (4.4) then provides that:

$$P_h^{-1}(P_h(\mathcal{L}(H))) = P_h^{-1}(P_h(\mathcal{L}(H_a))) = \mathcal{L}(H_a) \supseteq \mathcal{L}(H)$$

Defining the languages marked by these automata as $K = \mathcal{L}_m(H)$ and $K_a = \mathcal{L}_m(H_a)$ and assuming the automata are nonblocking, we then have that:

$$\overline{K_a} \supseteq \overline{K} \tag{4.7}$$

A final reduction rule of [21], the *certain conflicts rule*, is not guaranteed to satisfy the containment of equation (4.7). However, this rule is only relevant to blocking automata and hence is not employed in our approach. The certain conflicts rule could be modified to add self-loops for events that have not been hidden in order to provide the containment of equation (4.7) without affecting conflict equivalence.

Repeated application of equation (4.7) can be used to generate the expression in equation (4.8) where each K_i is either the marked language of a closed-loop module or a coordinating filter. Below we will show a few steps of the logic that leads us to equation (4.8).

Beginning with the expression $\overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k}$, equation (4.7) can be used to show that $\overline{K_{1,a}} \supseteq \overline{K_1}$ and that $\overline{K_{2,a}} \supseteq \overline{K_2}$. Therefore,

$$\overline{K_{1,a}} \cap \overline{K_{2,a}} \cap \overline{K_3} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k} \supseteq \overline{K_1} \cap \overline{K_2} \cap \overline{K_3} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k}$$

Applying equation (4.7) again, we have that $(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \supseteq \overline{K_{1,a}} \cap \overline{K_{2,a}}$ and that $\overline{K_{3,a}} \supseteq \overline{K_3}$. Combining these results with the above expression, we then have that:

$$(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \overline{K_{3,a}} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k} \supseteq \overline{K_1} \cap \overline{K_2} \cap \overline{K_3} \cap \ldots \cap \overline{K_{k-1}} \cap \overline{K_k}$$

Continued repetition of the above logic then leads us to equation (4.8) that will be employed in the proofs of the next section.

$$(((\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \overline{K_{3,a}})_a \cap \ldots)_a \cap \overline{K_{k-1,a}})_a \cap \overline{K_{k,a}} \supseteq \overline{K_1} \cap \ldots \cap \overline{K_k} \tag{4.8}$$

4.3.3 Controllability

Equation (4.8) and Propositions 3.3 and 3.4 defined earlier will help to demonstrate that our filters acting in conjunction with the modular supervisors will be language controllable with respect to the global uncontrolled plant language L. First though, we need the following lemma that demonstrates language controllability for the conjunction of a single filter and its associated blocking composition. This result will then be applied repeatedly to show controllability of the conjunction of all modular supervisors and filters. We will denote the languages marked and generated by the filters $H_{\mathit{filt},j}$ respectively as $K_{\mathit{filt},j} = \mathcal{L}_m(H_{\mathit{filt},j})$ and $\overline{K_{\mathit{filt},j}} = \mathcal{L}(H_{\mathit{filt},j})$.

Lemma 4.10. *Let K_{filt}, K_1, K_2, …, K_k, and $L \subseteq \Sigma^*$ be languages and L be prefix-closed. Let the subscript 'a' represent an abstraction satisfying $\overline{K_a} \supseteq \overline{K}$. Also let K_{filt}, K_1, K_2, …, K_k be a nonconflicting set. Let $\Sigma_u \subseteq \Sigma$ be the set of uncontrollable events. If K_{filt} is language controllable with respect to $L'_a = (\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \ldots)_a \cap \overline{K_{k,a}}$ and K_1, K_2, …, K_k are each language controllable with respect to L, then $K_{\mathit{filt}} \cap K_1 \cap \ldots \cap K_k$ is language controllable with respect to L.*

Proof.

• It is given that K_{filt} is language controllable w.r.t. L'_a:

$$\overline{K_{\mathit{filt}}}\Sigma_u \cap L'_a \subseteq \overline{K_{\mathit{filt}}}$$

• Noting equation (4.8), intersection of both sides of the above with $L' = \overline{K_1} \cap \overline{K_2} \cap \ldots \cap \overline{K_k}$ gives us that:

$$\overline{K_{\mathit{filt}}}\Sigma_u \cap L' \subseteq \overline{K_{\mathit{filt}}} \cap L' \tag{4.9}$$

• It is further given that K_1, K_2, …, K_k are each language controllable w.r.t. L. Hence, $L'\Sigma_u \cap L \subseteq L'$. This fact combined with equation (4.9) gives us that:

$$\overline{K_{\mathit{filt}}}\Sigma_u \cap (L'\Sigma_u \cap L) \subseteq \overline{K_{\mathit{filt}}}\Sigma_u \cap L' \subseteq \overline{K_{\mathit{filt}}} \cap L'$$

and substituting the expression for L' we get

$$(\overline{K_{\mathit{filt}}} \cap \overline{K_1} \cap \ldots \cap \overline{K_k})\Sigma_u \cap L \subseteq \overline{K_{\mathit{filt}}} \cap \overline{K_1} \cap \ldots \cap \overline{K_k}$$

• Also recalling that it is given that the set K_{filt}, K_1, K_2, …, K_k is nonconflicting, we have our desired result:

$$(\overline{K_{\mathit{filt}} \cap K_1 \cap \ldots \cap K_k})\Sigma_u \cap L \subseteq \overline{K_{\mathit{filt}} \cap K_1 \cap \ldots \cap K_k}$$

$\square$

The following theorem provides the language controllability result for the global system that we require.

Theorem 4.11. *Let $K_i = \mathcal{L}_m(H_i)$ be the language representing the behavior of the i^{th} subplant $L'_i = \mathcal{L}(G'_i)$ under the supervision of the i^{th} modular supervisor where $i \in \{1, \ldots, q\}$. Furthermore, let there be filters $K_{\mathit{filt},j}$, $j \in \{1, \ldots, k\}$, constructed as part of Algorithm 4.6 and satisfying requirements $R1$ and $R2$. The conjunction of supervised languages and filters $K_{\mathit{filt},1} \cap \ldots \cap K_{\mathit{filt},k} \cap K_1 \cap \ldots \cap K_q$ is then language controllable with respect to the global uncontrolled plant $L = \mathcal{L}(G) = L'_1 \cap \ldots \cap L'_q$.*

Proof.

• Each supervised language K_i is language controllable with respect to its associated subplant L'_i by construction. Since $L \subseteq L'_i$ for each local subplant, each closed-loop language is also language controllable with respect to the global plant L by Proposition 3.3.

• Let the set $K_1, \ldots, K_{i_1}$ be nonconflicting, where $1 \leq i_1 \leq q$. Since each K_i is language controllable with respect to L, $K_1 \cap \ldots \cap K_{i_1}$ is also language controllable with respect to L by Proposition 3.4.

• If $i_1 = q$, then there are no filters and we are done. Otherwise, the filter $K_{\mathit{filt},1}$ is needed to resolve the conflict in the composition $K'_{i_1,a} \cap K_{i_1+1,a}$, where $K'_{i_1,a} = ((\ldots(K_{1,a} \cap K_{2,a})_a \cap \ldots)_a \cap K_{i_1,a})_a$. By $R1$ and Theorem 4.7, the set $K_{\mathit{filt},1}, K_1, \ldots, K_{i_1+1}$ is nonconflicting. Also by $R2$, $K_{\mathit{filt},1}$ is language controllable with respect to $\overline{K'_{i_1,a}} \cap \overline{K_{i_1+1,a}}$, where $\overline{K'_{i_1,a}} = ((\ldots(\overline{K_{1,a}} \cap \overline{K_{2,a}})_a \cap \ldots)_a \cap \overline{K_{i_1,a}})_a$. Therefore, $K_{\mathit{filt},1} \cap K_1 \cap \ldots \cap K_{i_1+1}$ is language controllable with respect to L by Lemma 4.10.

• Let $K_{i_1+2}, \ldots, K_{i_2}$ be chosen such that the set $K_{\mathit{filt},1}, K_1, \ldots, K_{i_1+1}, K_{i_1+2}, \ldots, K_{i_2}$ is nonconflicting, where $i_1 + 2 \leq i_2 \leq q$. Also, since $K_{\mathit{filt},1} \cap K_1 \cap \ldots \cap K_{i_1+1}$ and each K_i is language controllable with respect to L, Proposition 3.4 provides that $K_{\mathit{filt},1} \cap K_1 \cap \ldots \cap K_{i_2}$ is language controllable with respect to L.

• If $i_2 = q$, then there are no more filters and we are done. Otherwise, the filter $K_{\mathit{filt},2}$ is needed to resolve the conflict in the composition $K'_{i_2,a} \cap K_{i_2+1,a}$, where

$K'_{i_2,a} = ((\ldots(K_{\mathit{filt,1},a} \cap K'_{i_1,a})_a \cap K_{i_1+2,a} \ldots)_a \cap K_{i_2,a})_a$. By $R1$ and Theorem 4.7, the set $K_{\mathit{filt,2}}, K_{\mathit{filt,1}}, K_1, \ldots, K_{i_2+1}$ is nonconflicting. Also by $R2$, $K_{\mathit{filt,2}}$ is language controllable with respect to $\overline{K'_{i_2,a}} \cap \overline{K_{i_2+1,a}}$, where $\overline{K'_{i_2,a}} = ((\ldots(\overline{K_{\mathit{filt,1},a}} \cap \overline{K'_{i_1,a}})_a \cap \overline{K_{i_1+2,a}} \ldots)_a \cap \overline{K_{i_2,a}})_a$. Hence, $K_{\mathit{filt,1}} \cap K_{\mathit{filt,2}} \cap K_1 \cap \ldots \cap K_{i_2+1}$ is language controllable with respect to L by Lemma 4.10.

• Repeating this logic, supervised modules and filters are added to the composition until they have all been addressed. The resulting composition $K_{\mathit{filt,1}} \cap \ldots \cap K_{\mathit{filt,k}} \cap K_1 \cap \ldots \cap K_q$ is, therefore, shown to be language controllable with respect to L. □

Theorem 4.7 and Theorem 4.11 therefore provide the desired result that deterministic filter laws built to satisfy requirements $R1$, $R2$, and $R3$ provide safe, nonblocking control when acting in conjunction with the modular supervisors.

4.3.4 Supervisor reduction

A further improvement over Algorithm 4.6 presented in Section 4.2 is that in some instances a closed-loop module H_i can be replaced by a reduction in the sense of [66], prior to the module being abstracted using a conflict-equivalent abstraction. For a plant G'_i and supervisor automaton S_i, [66] shows how to generate a reduced supervisor C_i that provides the same behavior as S_i when acting on the given plant, that is, $C_i \| G'_i = S_i \| G'_i$.

In our procedure, we propose that a closed-loop module $H_i = S_i \| G'_i$ can be replaced by the reduced supervisor C_i if the components making up the associated plant G'_i are included in other plant modules $\{G'_j | j < i\}$ that have already been addressed; here it is assumed the modules are addressed in numerical order. Otherwise stated, $J_i \subseteq \cup_{j<i} J_j$, where J_k is the set of indices of subplants in the composition that produces G'_k. The following proposition formalizes this idea for a situation involving three closed-loop modules and a single filter. Figure 4.3 can be referenced to help visualize the result. In the following, each of the closed-loop modules $H_i = S_i \| G'_i$ are supervised such that they satisfy the corresponding specification E_i. The corresponding subplants are defined as $G'_1 = G_1 \| G_2$, $G'_2 = G_3 \| G_4$, and $G'_3 = G_2 \| G_3$.

Proposition 4.12. *Let H_1, H_2, and H_3 be deterministic closed-loop modules. Let C_3 be the reduction constructed in the manner of [66] corresponding to the module $H_3 = S_3 \| G'_3$ where S_3 and G'_3 are respectively the deterministic supervisor and plant. Also, let $H_{\mathit{filt,1}}$ be a filter automaton constructed to satisfy requirements $R1$ and $R3$*

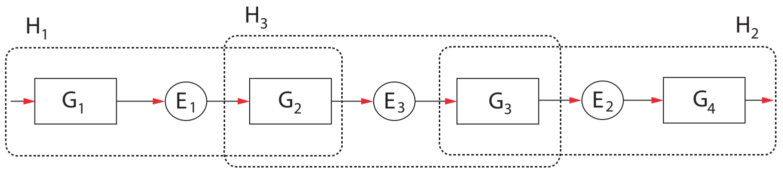

Figure 4.3: Three specification example for demonstrating supervisor reduction

with respect to the blocking composition $(H_{1,a} \| H_{2,a})_a \| C_{3,a}$. *If* $\mathcal{L}(H_1 \| H_2) \subseteq \mathcal{L}(G_3')$, *then* $H_{\mathit{filt},1} \| (H_{1,a} \| H_{2,a})_a \| C_{3,a} \simeq_{\mathit{conf}} H_{\mathit{filt},1} \| H_1 \| H_2 \| H_3$.

Proof.

• Since requirements $R1$ and $R3$ are given, the logic of Theorem 4.7 can be employed to show that

$$H_{\mathit{filt},1} \| (H_{1,a} \| H_{2,a})_a \| C_{3,a} \simeq_{\mathit{conf}} H_{\mathit{filt},1} \| H_1 \| H_2 \| C_3 \tag{4.10}$$

• Since it is given that $\mathcal{L}(H_1 \| H_2) \subseteq \mathcal{L}(G_3')$ and the involved automata models are deterministic, we have that $H_1 \| H_2 = H_1 \| H_2 \| G_3'$.

• Therefore,

$$H_{\mathit{filt},1} \| H_1 \| H_2 \| C_3 = H_{\mathit{filt},1} \| H_1 \| H_2 \| G_3' \| C_3 \tag{4.11}$$

• Since C_3 is a reduced supervisor of S_3, we further have that $H_3 = S_3 \| G_3' = C_3 \| G_3'$. Therefore,

$$H_{\mathit{filt},1} \| H_1 \| H_2 \| G_3' \| C_3 = H_{\mathit{filt},1} \| H_1 \| H_2 \| H_3 \tag{4.12}$$

• Since equality implies conflict equivalence, by equations (4.10), (4.11), and (4.12) we have the desired result that

$$H_{\mathit{filt},1} \| (H_{1,a} \| H_{2,a})_a \| C_{3,a} \simeq_{\mathit{conf}} H_{\mathit{filt},1} \| H_1 \| H_2 \| H_3$$

□

The idea of employing supervisor reduction in conflict resolution was first employed by [16] in constructing a different sort of conflict-resolving coordinator. The real advantage of this result is that since the relevant event set of C_i is smaller than the relevant event set of H_i, more reduction can take place in generating the conflict-equivalent abstractions of the preceding H_j.

4.4 State-Based Filter Requirements

In this section we will propose a new set of state-based requirements that are analogous to the language-based requirements of Section 4.3 ($R1$-$R3$). These new requirements are necessary because of the nondeterminism introduced into our models by the process of abstraction. The language-based requirements are not sufficient to guarantee safe, nonblocking control when applied to nondeterministic filter automata.

4.4.1 Supervisory control in the presence of nondeterminism

A difficulty that arises in considering how to construct filters is that each blocking composition $B_{j,a}$ is possibly nondeterministic because of the abstraction employed. Determinization is not appropriate in this instance because it can change the blocking properties of the automaton model. In addition, it can result in a new model with a state space that is exponentially larger than the original nondeterministic model. To avoid the determinization process, we need to specify a set of requirements and a filter construction algorithm that addresses nondeterminism.

An alternate way to think about our problem is that each blocking composition $B_{j,a}$ is like our uncontrolled plant and we are trying to build a supervisor (the filter $H_{\mathit{filt},j}$) to achieve a specification in a nonblocking manner. If we consider our specification to be the language Σ^*, then we have a situation where the "plant" is nondeterministic and the "specification" is deterministic. Of the existing research on supervisory control in the presence of nondeterminism, some address the situation where either only the plant is nondeterministic [36] [53] or only the specification is nondeterministic [13]. Still other research allows the supervisors to be nondeterministic but only in application to partially observed deterministic plants and deterministic specifications [31] [35]. The works applicable to our situation [36] [53] demonstrate conditions for supervisor existence, but do not provide a supervisor construction algorithm.

Another, perhaps more intuitive, way to think about our situation is to consider our specification to be the trim of $B_{j,a}$. Therefore, we have a situation where our "plant" and "specification" are both nondeterministic. Research that addresses this situation is presented in [24] [52] [81]. The work of [52] only addresses deadlock avoidance and its construction algorithm for building supervisors has exponential complexity. In the work of [81], conditions are presented under which a supervisor exists that can achieve behavior that is bisimilar to the given specification. A limitation of [81] is that supervisor synthesis is not addressed other than to mention that a search can be performed over the cartesian product of the plant and specification state spaces. The work of [24] handles the situation of a nondeterministic plant and specification by converting the models to partially observed deterministic ones. At this point, traditional techniques for control under partial observation can be applied. This approach could be applied to our situation, but we hope to avoid the conversion process and the exponential complexity of the traditional techniques.

As existing works do not provide a methodology for constructing the filters required by Algorithm 4.6 with less than exponential complexity, we will propose our own approach for constructing deterministic filter laws that meet the requirements $R1$, $R2$, and $R3$. We will represent these deterministic control laws by possibly nondeterministic automata in order to keep the representation compact and in order to avoid determinizing the model. One problem that arises is that language controllability is insufficient to assess the realizability of a control law in regards to nondeterministic automata, as demonstrated by the following example.

Example 4.13. Consider the automata in Fig. 4.4 where G is the plant and H is the specification and event b is uncontrollable. Since the string ab is in $\mathcal{L}(G)$ as well as in $\mathcal{L}(H)$, $\mathcal{L}(H)$ is language controllable with respect to $\mathcal{L}(G)$. However, the automaton H still requires that the uncontrollable event b be disabled at state 2. $\diamond$

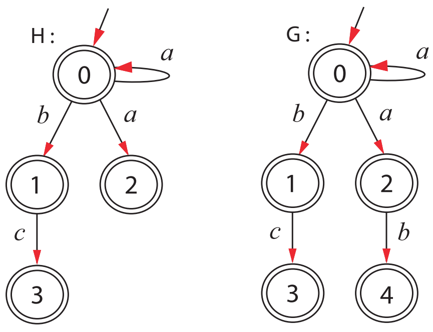

Figure 4.4: State controllability example

4.4.2 State controllability and observability

One solution to address the limitation of language controllability with respect to nondeterministic automata is to require a *state controllability* property similar to what was done in [13] and [81]. Language controllability requires that following an observed string s, if there is an uncontrollable continuation σ allowed in the plant automaton, then at least one instance of σ must be allowed following a string with the same observation s. With the state controllability property of [13] and [81], it is rather required that the continuation σ be allowed following every string with the observation s. In the case of subautomata as defined below, we can apply a slightly weaker notion of state controllability.

Definition 4.14. $H = (Q_h, \Sigma_\tau, \delta_h, q_{0h}, Q_{mh})$ is a *subautomaton* of $G = (Q_g, \Sigma_\tau, \delta_g, q_{0g}, Q_{mg})$ denoted $H \sqsubseteq G$ if and only if

$$Q_h \subseteq Q_g,\ q_{0h} = q_{0g},\ Q_{mh} = Q_{mg} \cap Q_h \text{ and}$$
$$p \in \delta_h(q,\sigma) \Rightarrow p \in \delta_g(q,\sigma). \quad \diamond$$

The idea of this weaker notion is that following a string with an observation s, we will require that an instance of an uncontrollable event σ must be allowed only if the event σ is possible in that particular state of the plant automaton. That is, if there is a string with an observation s that leads to a state in the plant automaton where σ is not possible, then σ does not have to be enabled at that state. Both state controllability properties imply language controllability. We will now formally define our state controllability property for a subautomaton.

Definition 4.15. Let $\Sigma_u \subseteq \Sigma_\tau$ with $\tau \in \Sigma_u$. Subautomaton H of G is *state controllable in G* if

$\forall s$ for which $\delta_h(q_0, s)!$ and $\forall q \in \delta_h(q_0, s)$ and $\forall \sigma \in \Sigma_u$, $p \in \delta_g(q,\sigma) \Rightarrow p \in \delta_h(q,\sigma)$ $\diamond$

State controllability as a property, however, is not sufficient to provide that the subautomaton H represents a deterministic control law with respect to G. If the same observed string leads to two different states, those two states could require conflicting control actions. As such, we need a new observability-type requirement.

Definition 4.16. Let $\Sigma_c \subseteq \Sigma$. Subautomaton H of G is *state observable in G* with respect to the event set Σ_c if

$$\forall s \text{ for which } \delta_h(q_0, s)! \text{ and } \forall q \in \delta_h(q_0, s) \text{ and } \forall \sigma \in \Sigma_c,$$
$$P_\tau(s)\sigma \in \mathcal{L}(H) \text{ and } p \in \delta_g(q,\sigma) \Rightarrow p \in \delta_h(q,\sigma) \quad \diamond$$

Taken together, state controllability and state observability provide that H represents a deterministic control law with respect to G. This is demonstrated formally by the following theorem that shows that a deterministic automaton H_{obs} that generates and marks the same languages as H will produce a result that is bisimulation equivalent to H when it is composed with G. In essence, H_{obs} can be considered a deterministic supervisor that achieves the specification represented by the nondeterministic automaton H for the nondeterministic plant model G. Similar results for generating control for bisimulation equivalence can be found in [47] [55] [67], but none demonstrate the following specific result. Here we implicitly assume the states of the automata are reachable.

Theorem 4.17. *Let $H = (Q_h, \Sigma_\tau, \delta_h, q_0, Q_{mh})$ and $G = (Q_g, \Sigma_\tau, \delta_g, q_0, Q_{mg})$ be (possibly nondeterministic) automata such that $H \sqsubseteq G$ and H is state controllable and state observable in G. Also let $\Sigma_\tau = \Sigma_c \dot{\cup} \Sigma_u$ where $\tau \in \Sigma_u$. If $H_{obs} = (Q_{ho}, \Sigma_\tau, \delta_{ho}, p_0, Q_{mho})$ is a deterministic automaton for which $\mathcal{L}(H_{obs}) = \mathcal{L}(H)$ and $\mathcal{L}_m(H_{obs}) = \mathcal{L}_m(H)$, then the synchronous composition $H_{obs}\|G =$ $(Q_\|, \Sigma_\tau, \delta_\|, (p_0, q_0), Q_{m\|})$ is bisimulation equivalent to H.*

Proof.
• Since $H \sqsubseteq G$, Definition 4.14 implies that $\mathcal{L}(H) \subseteq \mathcal{L}(G)$. Also, since $\mathcal{L}(H) = \mathcal{L}(H_{obs})$, $\mathcal{L}(H_{obs}\|G) = \mathcal{L}(H_{obs}) \cap \mathcal{L}(G) = \mathcal{L}(H)$.
• Therefore, $H_{obs}\|G$ and H are equivalent in terms of the languages they generate. We will show in the following, however, that they are also bisimulation equivalent.
• If H is the empty automaton, then so is $H_{obs}\|G$ since their generated languages are both empty. Otherwise, $H_{obs}\|G$ and H can be shown to be bisimulation equivalent by demonstrating that their initial states are bisimulation equivalent. That is, $(p_0, q_0) \sim q_0$ in the sense of Definition 2.7.
• Specifically, we need to demonstrate Points $(i)-(iii)$ of Definition 2.7 for the initial states. In words, Point (i) means that if an event takes the state (p_0, q_0) to the state $(p_1, q_1) \in Q_\|$, then the same event must take q_0 to a state $q_1 \in Q_h$ that is bisimulation equivalent to (p_1, q_1), that is, $(p_1, q_1) \sim q_1$. Point (ii) likewise means that if an event takes the state q_0 to the state $q_1 \in Q_h$, then the same event must take (p_0, q_0) to a state $(p_1, q_1) \in Q_\|$ that is bisimulation equivalent to q_1, $(p_1, q_1) \sim q_1$. Point (iii) further requires that (p_0, q_0) and q_0 have the same marking.
• Since the bisimulation equivalence of (p_0, q_0) and q_0 depends on the bisimulation

equivalence of states subsequently reached by the same strings, we will perform this proof iteratively.

• (Point *i*) Let $\sigma_1 \in \Sigma_\tau$. We need to show that $(p_1, q_1) \in \delta_{\|}((p_0, q_0), \sigma_1)$ implies that $q_1 \in \delta_h(q_0, \sigma_1)$. Assuming $(p_1, q_1) \in \delta_{\|}((p_0, q_0), \sigma_1)$, $p_1 = \delta_{ho}(p_0, \sigma_1)$ and $q_1 \in \delta_g(q_0, \sigma_1)$ by Definition 2.1. The following then shows that $q_1 \in \delta_h(q_0, \sigma_1)$:

- If $\sigma_1 \in \Sigma_u$, then $q_1 \in \delta_h(q_0, \sigma_1)$ since $q_1 \in \delta_g(q_0, \sigma_1)$ and H is state controllable in G.
- If $\sigma_1 \in \Sigma_c$, then $\sigma_1 \in \mathcal{L}(H)$ since $\sigma_1 \in \mathcal{L}(H_{obs} \| G) = \mathcal{L}(H_{obs})$. Since it is also known that $q_1 \in \delta_g(q_0, \sigma_1)$ and H is state observable in G, we then have that $q_1 \in \delta_h(q_0, \sigma_1)$.

• (Point *ii*) Now we need to show that $q_1 \in \delta_h(q_0, \sigma_1)$ implies $(p_1, q_1) \in \delta_{\|}((p_0, q_0), \sigma_1)$. Since $H \sqsubseteq G$, $q_1 \in \delta_h(q_0, \sigma_1)$ implies $q_1 \in \delta_g(q_0, \sigma_1)$ by Definition 4.14. If $\sigma_1 = \tau$, then Definition 2.1 provides that $(p_0, q_1) \in \delta_{\|}((p_0, q_0), \sigma_1)$ and we can let $(p_1, q_1) = (p_0, q_1)$. If $\sigma_1 \in \Sigma_\tau - \{\tau\}$, then $\sigma_1 \in \mathcal{L}(H) = \mathcal{L}(H_{obs})$. Since H_{obs} is deterministic, we then have that $\delta_{ho}(p_0, \sigma_1)$ has a single element that we will call p_1. Therefore, $(p_1, q_1) \in \delta_{\|}((p_0, q_0), \sigma_1)$ again by Definition 2.1.

• (Point *iii*) We now need to show that $q_0 \in Q_{mh}$ if and only if $(p_0, q_0) \in Q_{m\|}$.

($\Rightarrow$) Let $q_0 \in Q_{mh}$. Definition 4.14 then implies that $q_0 \in Q_{mg}$. Additionally, since $\mathcal{L}_m(H) = \mathcal{L}_m(H_{obs})$, $\varepsilon \in \mathcal{L}_m(H)$ implies that $\varepsilon \in \mathcal{L}_m(H_{obs})$ and thus $p_0 \in Q_{mho}$. Therefore, by Definition 2.1 $(p_0, q_0) \in Q_{m\|}$.

($\Leftarrow$) Let $(p_0, q_0) \in Q_{m\|}$. This implies that $q_0 \in Q_{mg}$ by Definition 2.1 which in turn implies that $q_0 \in Q_{mh}$ by Definition 4.14 since we already have that $q_0 \in Q_h$.

• The above logic holds for any $\sigma_1 \in \Sigma_\tau$ for which $\delta_{\|}((p_0, q_0), \sigma_1)$ or $\delta_h(q_0, \sigma_1)$ is nonempty.

• In order to complete the proof that $(p_0, q_0) \sim q_0$, we then need that $(p_1, q_1) \sim q_1$. This can be demonstrated by following logic similar to that given above.

• (Point *i*) Let $\sigma_2 \in \Sigma_\tau$. We need to show that $(p_2, q_2) \in \delta_{\|}((p_1, q_1), \sigma_2)$ implies that $q_2 \in \delta_h(q_1, \sigma_2)$. Assuming $(p_2, q_2) \in \delta_{\|}((p_1, q_1), \sigma_2)$, $p_2 = \delta_{ho}(p_1, \sigma_2)$ and $q_2 \in \delta_g(q_1, \sigma_2)$ by Definition 2.1. The following then shows that $q_2 \in \delta_h(q_1, \sigma_2)$:

- If $\sigma_2 \in \Sigma_u$, then $q_2 \in \delta_h(q_1, \sigma_2)$ since $q_2 \in \delta_g(q_1, \sigma_2)$ and H is state controllable in G.
- If $\sigma_2 \in \Sigma_c$, then $P_\tau(\sigma_1)\sigma_2 \in \mathcal{L}(H)$ since $P_\tau(\sigma_1)\sigma_2 \in \mathcal{L}(H_{obs} \| G) = \mathcal{L}(H_{obs})$.

Since it is also known that $q_2 \in \delta_g(q_1, \sigma_2)$ and H is state observable in G, we then have that $q_2 \in \delta_h(q_1, \sigma_2)$.

• (Point *ii*) Now we need to show that $q_2 \in \delta_h(q_1, \sigma_2)$ implies $(p_2, q_2) \in \delta_{\|}((p_1, q_1), \sigma_2)$. Since $H \sqsubseteq G$, $q_2 \in \delta_h(q_1, \sigma_2)$ implies $q_2 \in \delta_g(q_1, \sigma_2)$ by Definition 4.14. If $\sigma_2 = \tau$, then Definition 2.1 provides that $(p_1, q_2) \in \delta_{\|}((p_1, q_1), \sigma_2)$ and we can let $(p_2, q_2) = (p_1, q_2)$. If $\sigma_2 \in \Sigma_\tau - \{\tau\}$, then $P_\tau(\sigma_1)\sigma_2 \in \mathcal{L}(H) = \mathcal{L}(H_{obs})$. Since H_{obs} is deterministic, we then have that $\delta_{ho}(p_1, \sigma_2)$ has a single element that we will call p_2. Therefore, $(p_2, q_2) \in \delta_{\|}((p_1, q_1), \sigma_2)$ again by Definition 2.1.

• (Point *iii*) We now need to show that $q_1 \in Q_{mh}$ if and only if $(p_1, q_1) \in Q_{m\|}$.

($\Rightarrow$) Let $q_1 \in Q_{mh}$. Definition 4.14 then implies that $q_1 \in Q_{mg}$. Additionally, since $\mathcal{L}_m(H) = \mathcal{L}_m(H_{obs})$, $P_\tau(\sigma_1) \in \mathcal{L}_m(H)$ implies that $P_\tau(\sigma_1) \in \mathcal{L}_m(H_{obs})$ and thus $p_1 \in Q_{mho}$. Therefore, by Definition 2.1 $(p_1, q_1) \in Q_{m\|}$.

($\Leftarrow$) Let $(p_1, q_1) \in Q_{m\|}$. This implies that $q_1 \in Q_{mg}$ by Definition 2.1 which in turn implies that $q_1 \in Q_{mh}$ by Definition 4.14 since we already have that $q_1 \in Q_h$.

• The above logic holds for any $\sigma_2 \in \Sigma_\tau$ for which $\delta_{\|}((p_1, q_1), \sigma_2)$ or $\delta_h(q_1, \sigma_2)$ is nonempty.

• In order to complete the proof that $(p_1, q_1) \sim q_1$, we then need that $(p_2, q_2) \sim q_2$. This can be demonstrated by following logic similar to that given above. In general, this logic can be repeated for all reachable states of $H_{obs}\|G$ or H.

• Therefore, $(p_0, q_0) \sim q_0$ and hence, $H_{obs}\|G$ is bisimulation equivalent to H.

$\square$

4.4.3 State-based requirements

The above theorem allows our filters to now be represented by possibly nondeterministic automata models, $H_{\mathit{filt},j}$. More specifically, it allows us to replace the language-based requirements of Section 4.3 with the following set of state-based requirements:

State-based filter requirements

$R1'$) $H_{\mathit{filt},j}$ is a nonblocking subautomaton of $B_{j,a}$

$R2'$) $H_{\mathit{filt},j}$ is state controllable and state observable in $B_{j,a}$

$R3'$) $\Sigma(H_{\mathit{filt},j}) \cap \Sigma_h = \emptyset$

These results imply that a determinized version of the automaton $H_{\mathit{filt},j}$, which we will denote $H_{\mathit{filt},j,obs}$, will meet the previously established requirements $R1$, $R2$, and $R3$. Specifically, conditions $R1'$ and $R2'$ together with Theorem 4.17 imply that $H_{\mathit{filt},j,obs} \| B_{j,a}$ is nonblocking since it is bisimulation equivalent to $H_{\mathit{filt},j}$. Therefore, requirement $R1$ is satisfied. Furthermore, since $H_{\mathit{filt},j}$ is state controllable, the generated language $\mathcal{L}(H_{\mathit{filt},j})$ is language controllable. This then implies that $\mathcal{L}(H_{\mathit{filt},j,obs})$ is also language controllable, thereby satisfying requirement $R2$. Also, $R3'$ are $R3$ are equivalent.

Thus far we have demonstrated that determinized versions of the filter automata $H_{\mathit{filt},j}$ meeting requirements $R1'$, $R2'$, and $R3'$ will provide safe, nonblocking control when acting in conjunction with traditionally built modular supervisors. However, we would like to avoid the determinization process. Since the nondeterministic filter automata $H_{\mathit{filt},j}$ possesses all the information that $H_{\mathit{filt},j,obs}$ does, it turns out that $H_{\mathit{filt},j,obs}$ never actually has to be constructed. However, the control required by the automaton $H_{\mathit{filt},j}$ cannot be implemented via the synchronous composition operation. Rather, following the observation of a string $s \in \Sigma^*$, all continuations active at all states reached by strings with the same observation must be allowed. In essence, we are using $H_{\mathit{filt},j}$ and its transition function δ_{filt} to generate an online implementation of $H_{\mathit{filt},j,obs}$. Consider the following mathematical definition of the filter law $\mathcal{H}_{\mathit{filt},j} : \mathcal{L}(B_{j,a}) \to 2^{\Sigma}$ that determines which events are to be enabled.

$$\mathcal{H}_{\mathit{filt},j}(s) := \bigcup_{q \in T(s)} \Sigma_{H_{\mathit{filt}}}(q), \text{ where } T(s) = \{q \mid q \in \delta_{\mathit{filt}}(q_0, t) \text{ and } t \in P_{\tau}^{-1}(s)\} \quad (4.13)$$

In order to make this more clear, consider the automaton G_a in Fig. 4.1. If we consider G_a to be a nondeterministic representation of a deterministic control law, then following an observation of the string ab we do not know if we are in state 3 or state 4, therefore, we have to allow both event c and event d to occur. The result of Theorem 4.17 also allows the composition $H_{\mathit{filt},j,obs} \| B_{j,a}$ to be replaced by the subautomaton $H_{\mathit{filt},j}$ in the course of Algorithm 4.6.

4.5 Filter Law Construction

Having established that we can employ filter laws represented by nondeterministic automata, the final question that remains is how to construct $H_{\mathit{filt},j}$ so that requirements $R1'$, $R2'$, and $R3'$ are satisfied. Since we are ultimately trying to find a subautomaton, we are in essence trying to solve a state avoidance problem. This type of problem can be solved by a state-feedback approach to control. In other words, the control applied depends only on the state the system is in, not on the path taken to get there. It is well-established that a static control law of this type is potentially more restrictive than a dynamic control law in the case of partial observation [34]. However, we are willing to make this sacrifice in order to avoid exponential complexity. In this section we introduce some existing results on state-based approaches to control that can be employed to generate the conflict-resolving filters required by the approach of this paper. Additionally, we develop an improved covering-based approach to control that is less restrictive than existing state-feedback approaches.

4.5.1 State-based supervisory control

A *state-feedback supervisory controller* is a function $f : Q_g \to 2^{\Sigma}$ that determines the set of events to be enabled based on the current state of the system under control $G = (Q_g, \Sigma_\tau, \delta_g, q_0, Q_{mg})$. In the context of our larger approach to conflict resolution, the "plant" G represents a given blocking composition $B_{j,a}$. The closed-loop system f/G then represents the allowable set of states, that is, the subautomaton representing the coordinating filter $H_{\mathit{filt},j}$. If G is equal to the empty automaton, then so is f/G. Otherwise, f/G is defined iteratively as that portion of G that is reachable via transitions that are allowed by f:

Definition 4.18.

$$f/G = (Q_f, \Sigma_\tau, \delta_f, q_0, Q_{mf}) \tag{4.14}$$

Iterative Definition of f/G:
Step 1: $q_0 \in Q_f$.
Step 2: If $q \in Q_f$ and $\delta_g(q, \sigma)!$ for some $\sigma \in f(q)$, then $q' \in Q_f$, $\forall q' \in \delta_g(q, \sigma)$. Also, $\delta_f(q, \sigma) = \delta_g(q, \sigma)$. Otherwise, $\delta_f(q, \sigma)$ is empty.
Step 3: Every state in Q_f and every transition for which δ_f is nonempty is obtained as in Step 1 and Step 2. Also, $Q_{mf} = Q_f \cap Q_{mg}$. $\diamond$

Note from the above definition that all of the states of f/G are reachable and inherit their marking from G. The existence of a state-feedback controller that can keep the behavior of G within a set of "good" states represented by the subset $Q_h \subseteq Q_g$ requires a property called Σ_u-invariance [56]. In terms of nondeterministic automata and the notation of this paper, Σ_u-invariance of a state set $Q_h \subseteq Q_g$ is equivalent to the state controllability of a subautomaton of G, $H = (Q_h, \Sigma_\tau, \delta_h, q_0, Q_{mh})$. If the state space of G is not fully observable, then additional considerations must be addressed.

In existing work on state-feedback control under partial observation, the concept of a "mask" is employed. A mask M is defined as a function $M : Q_g \rightarrow Y$ that maps elements from the state space Q_g to the observation space Y. Under partial observation two states q and q' might not be distinguishable, that is, they could have the same observation $M(q) = M(q') = y$. It is then necessary that the state-feedback control $f(q)$ be determined based on $M(q)$. Specifically, it is required that:

$$\text{For any } q, q' \in Q_g, M(q) = M(q') \Rightarrow f(q) = f(q') \tag{4.15}$$

In existing state-feedback work [43] [69] it is assumed the mask M is given. In this paper we will assume the mask M is constructed to satisfy the following constraint.

$$\begin{aligned}&\text{The mask } M : Q_g \rightarrow Y \text{ is defined such that if } \exists s, s' \in \Sigma_\tau^* \text{ with}\\ &q \in \delta_g(q_0, s),\ q' \in \delta_g(q_0, s') \text{ and } P_\tau(s) = P_\tau(s'), \text{ then } M(q) = M(q').\end{aligned} \tag{4.16}$$

In the above, states $q \in \delta_g(q_0, s)$ and $q' \in \delta_g(q_0, s')$ for which $P_\tau(s) = P_\tau(s')$ are defined to be *indistinguishable*. In other words, two states that are reached by strings that have the same projection are indistinguishable and the mask M is constructed such that all indistinguishable states map to the same observation under M.

Of the existing work for generating state-feedback control under partial observation, the least restrictive control strategy is proposed in [69] and builds off the prior results of [68] and [70]. We will now outline their strategy using notation consistent with this paper and extensions we have added to handle nondeterminism. Specifically, [69] presents an algorithm for constructing a state-feedback controller that satisfies equation (4.15). This algorithm is based on the following sets $A_H(q) \subseteq \Sigma_c$ that define which events must be disabled at state q for a set of allowable states represented by the subautomaton $H \sqsubseteq G$. In essence, the sets $A_H(q)$ capture which events at a given state will cause a violation of the observability-type property captured by equation (4.15).

$$A_H(q) = \{\sigma \in \Sigma_c \mid (\exists q' \in Q_h) : [M(q) = M(q')] \wedge [\exists p \in \delta_g(q', \sigma) \text{ such that } p \notin Q_h]\} \tag{4.17}$$

In the above equation, $A_H(q)$ is also defined for $q' = q$ since a state is always considered indistinguishable from itself, that is, $M(q) = M(q)$. The resulting state-feedback control law is thus defined:

$$f(q) = \Sigma_\tau - A_H(q) \tag{4.18}$$

In order to guarantee that the control law defined by equation (4.18) is able to achieve the specification required by the subautomaton H, [69] requires the following property in addition to Σ_u-invariance:

$$Q_h \subseteq R(Q_h) \tag{4.19}$$

In the above, R is a transformation that represents which states of H are reachable by permissable transitions, that is, those transitions that are not prohibited by the sets $A_H(q)$. Recall, the sets $A_H(q)$ enforce the observability-type requirement prescribed in equation (4.15). If $Q_h = \emptyset$, then $R(Q_h) = \emptyset$. Otherwise, the set of states represented by $R(Q_h)$ can be constructed iteratively in a similar manner to [69]:

Algorithm 4.19. $R(Q_h)$ *Construction*
Step 1: $q_0 \in R(Q_h)$.
Step 2: If $q \in R(Q_h)$ and $\delta_g(q,\sigma) \subseteq Q_h$ for some $\sigma \in \Sigma_\tau - A_H(q)$, then $q' \in R(Q_h)\ \forall q' \in \delta_g(q,\sigma)$.
Step 3: Every state satisfying $R(Q_h)$ is obtained as in Step 1 and Step 2. $\diamond$

A specification represented by the set of states Q_h that is Σ_u-invariant and satisfies equation (4.19) is defined to be *M-controllable* in [70]. The work of [70] further demonstrates that a state-feedback controller that can achieve the behavior prescribed by the state set Q_h exists if and only if the state set is M-controllable. In particular, the control law given by equation (4.18) will achieve the M-controllable state set Q_h and is further the supremal state-feedback control law that satisfies equation (4.15).

If a given state set is not M-controllable, then [69] prescribes an approach for finding an M-controllable subset, $R(Q_h^\uparrow)$. Here the $\uparrow$ operation generates the supremal Σ_u-invariant subset of states constructed according to [56]. While the resulting $R(Q_h^\uparrow)$ is not necessarily maximal or supremal, it does represent a larger state set than can be achieved by other existing state-feedback approaches [43] [68].

The subautomaton that results from the state-feedback controller of [69] can be shown to be state controllable and state observable in G. Therefore, the results given above could be directly applied to the construction of filters required by our approach to conflict resolution. In the next section, however, we will propose an improved covering-based approach that generates a more permissive control law than [69].

4.5.2 Covering-based supervisory control

Our improvement over [69] derives from the fact that the requirement of equation (4.15) is stronger than necessary for the achievement of state observability. Therefore, we can apply a new covering-based approach that will result in a less restrictive control law. In this section we will additionally address blocking.

The observability-type requirement of equation (4.15) is too strong based on the character of the mask M. The fact that M is a function implies that when the state space is observed through this mask it is effectively partitioned into disjoint sets of states that have the same observation. For example, if q and q' have the same observation $M(q) = M(q')$, and q' and q'' have the same observation $M(q') = M(q'')$, it then follows that q and q'' must have the same observation $M(q) = M(q'')$. Therefore, all three states q, q', and q'' must be in the same partition of the state space. For achievement of state observability, however, it may not be necessary that the same control action be applied at q and q'' if they are not indistinguishable. In other words, if the observed string that reaches q and q' is different than the observed string that reaches q' and q'', then the control applied at q and q'' may be allowed to be different. Figure 4.5 illustrates this situation where σ is disabled at q'', but need not be disabled at q since these states are not both reached by the same observed string. In essence, we would like to base our control on a covering of the state space rather than a partition like that imposed by the mask M. If the event σ was possible at the state q', then σ would need to be disabled at all three states for our covering-based approach too.

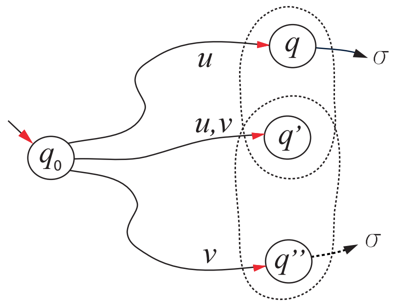

Figure 4.5: Example of a covering for indistinguishable states

In order to present out covering-based approach, we will employ the mapping I_H defined as follows:

Definition 4.20. Let $I_H : Q_h \rightarrow 2^{Q_h}$ be a mapping defined $\forall q, q' \in Q_h$ as follows: $q' \in I_H(q)$ if q and q' are indistinguishable, that is, if $\exists s, s' \in \Sigma_\tau$ such that $q \in \delta_h(q_0, s)$, $q' \in \delta_h(q_0, s')$, and $P_\tau(s) = P_\tau(s')$. $\diamond$

In the above, it is always the case that a state q is considered indistinguishable from itself, that is, $q \in I_H(q)$.

We can then define new sets of prohibited events, $A'_H(q) \subseteq \Sigma_c$. At a state q in the uncontrolled plant G, a controllable transition σ that is defined in the uncontrolled plant G is prohibited if it leads to a state outside of Q_h or if it is prohibited at a state q' that is indistinguishable from q in H.

Since the definition of prohibited events at a state q, $A'_H(q)$, depends on the prohibited events of other states, each set $A'_H(q)$ is constructed iteratively. In words, if there is a string of indistinguishable states defined:

$$q \in I_H(q'),\ q' \in I_H(q''), \ldots,\ q^{(m-1)} \in I_H(q^{(m)})$$

each with σ possible in G and such that σ is not possible in H at $q^{(m)}$, then σ is again added to $A'_H(q)$. This construction indicates a transitivity similar to that imposed by M, *except* that here the transitivity is limited to indistinguishable states where σ is possible in G. Assuming the mapping I_H is given, $A'_H(q)$ can be constructed according to Algorithm 4.21 given below. The mapping I_H can be constructed with polynomial complexity using results from [71].

Algorithm 4.21. *Prohibited Events Determination*

```
Input: automaton $G$, subautomaton $H \sqsubseteq G$ and mapping $I_H : Q_h \rightarrow 2^{Q_h}$
For each $q \in Q_h$
    For each transition $\sigma \in \Sigma_G(q) \cap \Sigma_c$
        If $\exists p \in \delta_g(q, \sigma)$ such that $p \notin \delta_h(q, \sigma)$ then
            add $\sigma$ to the set of prohibited events at $q$, $A'_H(q) \leftarrow \{\sigma\} \cup A'_H(q)$.
        End if
        If $\sigma \in A'_H(q)$ then
```

```
            define a set of states $T$ that is initialized with the state $q$, $T \leftarrow \{q\}$.
            Also let $\mathcal{M} : Q_h \rightarrow \{0, 1\}$ be a partial function marking whether
            or not states in the set $T$ have been addressed yet. Set $\mathcal{M}(q) = 0$.
            For each $q' \in T$ with $\mathcal{M}(q') = 0$
                For each $q'' \in I_H(q')$ that is not in $T$
                    If $\delta_g(q'', \sigma)!$ then
                        add state $q''$ to the set $T$, $T \leftarrow \{q''\} \cup T$, and add event $\sigma$
                        to the set of prohibited events at $q''$, $A'_H(q'') \leftarrow \{\sigma\} \cup A'_H(q'')$.
                        Set $\mathcal{M}(q'') = 0$.
                    End if
                End for
                Set $\mathcal{M}(q') = 1$.
            End for
            Clear $T$ and $\mathcal{M}$.
        End if
    End for
End for
Output: the sets $A'_H(q)$   $\diamond$
```

Algorithm 4.21 has complexity $\mathcal{O}(mn^2)$ where m is the cardinality of the event set and n is the cardinality of the state space. Assuming the subautomaton H has a finite number of transitions, this algorithm will terminate in finite time. The sets $A'_H(q)$ defined by Algorithm 4.21 satisfy the following equation by construction:

$$A'_H(q) = \{\sigma \in \Sigma_c \mid (\delta_g(q, \sigma)! \wedge \exists q' \in Q_h) : [q' \in I_H(q)] \wedge \{[\exists p \in \delta_g(q', \sigma) \text{ such that } p \notin Q_h] \vee [\sigma \in A'_H(q')]\}\} \quad (4.20)$$

In order to explicitly compare the sets $A_H(q)$ and $A'_H(q)$, we now define the following mapping J_H that reflects the partition implicitly imposed by a given mask

M on the state set Q_h:

Definition 4.22. Let $J_H : Q_h \to 2^{Q_h}$ be a mapping defined $\forall q, q' \in Q_h$ as follows: $q' \in J_H(q)$ if $M(q) = M(q')$. $\diamond$

The definition of $A_H(q)$ given in equation (4.17) can then be rewritten in terms of the mapping J_H as follows:

$$A_H(q) = \{\sigma \in \Sigma_c \mid (\exists q' \in Q_h) : [q' \in J_H(q)] \wedge [\exists p \in \delta_g(q', \sigma) \text{ such that } p \notin Q_h]\} \tag{4.21}$$

Examining equations (4.20) and (4.21), σ can be an element of $A_H(q)$ when $\delta_g(q, \sigma)$ is empty, while it cannot be an element of $A'_H(q)$. Furthermore, since the mapping J_H imparts a partition on the state space Q_h, $A_H(q) = A_H(q')$ if $q' \in J_H(q)$. These observations along with the fact that $I_H(q) \subseteq J_H(q)$ then implies that $A'_H(q) \subseteq A_H(q)$.

This new $A'_H(q)$ can then be employed to generate a new transformation R'. R' is defined in the same manner as Algorithm 4.19 that constructs R except for the different definition of prohibited events that is employed. R' is again a transformation that retains those states of H that are reachable by permissible transitions. As we construct the state set $R'(Q_h)$ below, we will additionally construct an associated subautomaton of H, $R'(H)$, with a state set $R'(Q_h)$ and a transition function $\delta_{R'}$. Based on our definition of a subautomaton given in Definition 4.14, the marking of $R'(H)$ will be consistent with the marking of H. The use of $A'_H(q)$ in the construction process will result in the transformed subautomaton $R'(H)$ being state observable in G.

If $Q_h = \emptyset$, then $R'(Q_h) = \emptyset$ and $R'(H)$ is the empty automaton. Otherwise, $R'(Q_h)$ and $R'(H)$ are constructed as follows:

Algorithm 4.23. *Construction of $R'(Q_h)$ and $R'(H)$*
Step 1: $q_0 \in R'(Q_h)$.
Step 2: If $q \in R'(Q_h)$ and $\delta_g(q, \sigma) \subseteq Q_h$ for some $\sigma \in \Sigma_\tau - A'_H(q)$, then $q' \in R'(Q_h)$ $\forall q' \in \delta_g(q, \sigma)$ and $\delta_{R'}(q, \sigma) = \delta_g(q, \sigma)$. Otherwise, $\delta_{R'}(q, \sigma)$ is empty.
Step 3: Every state satisfying $R'(Q_h)$ and every transition for which $\delta_{R'}$ is nonempty is obtained as in Step 1 and Step 2. $\diamond$

From the above algorithm and employing logic from [70], for any $q \in R'(Q_h) - q_0$, there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying the following conditions:

$C1)$ $\delta_g(q_i, \sigma_i) = q_{i+1}$ for $i = 0, 1, \ldots, m-1$

$C2)$ $q_i \in Q_h$ for $i = 0, 1, \ldots, m$

$C3)$ $\sigma_i \in \Sigma_\tau - A'_H(q_i)$ for $i = 0, 1, \ldots, m-1$

$C4)$ $q_m = q$

The following result that employs the logic of Lemma 1 in [68] can now be presented.

Proposition 4.24. *For any subautomaton $H \sqsubseteq G$*

$$R(Q_h) \subseteq R'(Q_h) \tag{4.22}$$

Proof. If $q_0 \notin Q_h$, then $R(Q_h) = R'(Q_h) = \emptyset$. Consider the case that $q_0 \in Q_h$. Since $A'_H(q) \subseteq A_H(q)$ for any $q \in Q_h$, we have $R(Q_h) \subseteq R'(Q_h)$. □

Although the result of $R(Q_h^\uparrow)$ represents a larger state set than can be achieved by any prior state-feedback work, Proposition 4.24 demonstrates that we can generate a potentially larger state set $R'(Q_h^\uparrow) \supseteq R(Q_h^\uparrow)$. This together with the example of Section 4.5.4 shows that our covering-based approach is less restrictive than existing state-feedback approaches.

4.5.3 Covering-based filter construction

In this section we will specify how the subautomaton $R'(H^\uparrow)$ is constructed. Here $R'(H^\uparrow)$ is defined according to Algorithm 4.23, where $H^\uparrow \subseteq H$ is the subautomaton that possesses the state set $Q_h^\uparrow$. Specifically, the transition structure of $H^\uparrow$ is defined such that it includes all transitions $q \xrightarrow{\sigma} q'$ of H for which the source state q and destination state $q' \in \delta_h(q, \sigma)$ are in $Q_h^\uparrow$.

Since a subautomaton $R'(H^\uparrow)$ will be employed to represent each coordinating filter, we must first demonstrate that $R'(H^\uparrow)$ is state observable and state controllable in its associated G. We will specifically show that state observability holds directly as a result of the R' transformation. We will then demonstrate that the R' operation did not destroy the state controllability achieved by the $\uparrow$ operation. In order to accomplish this goal, we first propose the following hypothetical state-feedback control law f' that achieves the specification $R'(Q_h^\uparrow)$; though, our covering-based law will ultimately be implemented according to equation (4.13).

$$f'(q) = \Sigma_\tau - A'_{H\uparrow}(q) \tag{4.23}$$

The subautomaton $f'/G = (Q_{f'}, \Sigma_\tau, \delta_{f'}, q_0, Q_{mf'})$ is defined in the same manner as Definition 4.18. Also, for any state $q \in Q_g - q_0$ that is also in $Q_{f'}$, there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying the following conditions [44]:

$C5)$ $q_{i+1} \in \delta_g(q_i, \sigma_i)$ for $i = 0, 1, \ldots, m-1$

$C6)$ $\sigma_i \in f'(q_i)$ for $i = 0, 1, \ldots, m-1$

$C7)$ $q_m = q$

Based on the manner in which the sets $A'_{H\uparrow}(q)$ are constructed, it can also be seen that the following relation is implied where $I_{f'}$ is defined for the automaton f'/G.

$$\text{For any } q,\ q' \in Q_{f'} \text{ with } q' \in I_{f'}(q),\ \sigma \in f'(q) \cap \Sigma_G(q) \Rightarrow \sigma \in f'(q') \tag{4.24}$$

The above then leads to a result that is similar to equation (4.15):

$$\text{For any } q, q' \in Q_{f'}, q' \in I_{f'}(q) \Rightarrow f'(q) \cap \Sigma_G(q) \cap \Sigma_G(q') = f'(q') \cap \Sigma_G(q) \cap \Sigma_G(q') \tag{4.25}$$

The above expression captures that the control applied by f' is consistent between states that are indistinguishable for those feasible events that are shared between the states. The limitation of consistency to those feasible events shared between states demonstrates the limited transitivity that provides the improvement of our covering-based approach. With the above property, we can now demonstrate that state observability is achieved.

Proposition 4.25. *The state-feedback law f' given by equation (4.23) generates a subautomaton f'/G that is state observable in G.*

Proof.

• Let $q \in \delta_{f'}(q_0, s)$, $\sigma \in \Sigma_c$, $P_\tau(s)\sigma \in \mathcal{L}(f'/G)$ and $p \in \delta_g(q, \sigma)$.

• $P_\tau(s)\sigma \in \mathcal{L}(f'/G)$ implies there exists an $s' \in \Sigma_\tau^*$ and $q' \in \delta_{f'}(q_0, s')$ for which $P_\tau(s') = P_\tau(s)$ and $\sigma \in f'(q')$.

• Since $P_\tau(s') = P_\tau(s)$, $q' \in I_{f'}(q)$. Therefore, we have that $f'(q) \cap \Sigma_G(q) \cap \Sigma_G(q') = f'(q') \cap \Sigma_G(q) \cap \Sigma_G(q')$ by equation (4.25).

• Since $\sigma \in f'(q') \cap \Sigma_G(q) \cap \Sigma_G(q')$, $\sigma \in f'(q) \cap \Sigma_G(q) \cap \Sigma_G(q')$ also.

• Since $\sigma \in f'(q)$ and $p \in \delta_g(q, \sigma)$, $p \in \delta_{f'}(q, \sigma)$ by definition of $\delta_{f'}$. Thus we have shown that f'/G is state observable in G. □

This control law f' achieves the set of states $R'(Q_h^{\uparrow})$. This fact is mathematically represented $Q_{f'} = R'(Q_h^{\uparrow})$ and is proven in the following proposition employing logic from Theorem 1 of [70]:

Proposition 4.26. *If for the subautomaton $H \sqsubseteq G$, $H^{\uparrow}$ is nonempty and f' is given by equation (4.23), then $Q_{f'} = R'(Q_h^{\uparrow})$.*

Proof. See proof in Appendix. □

Based on the above proposition, we have that $q_0 \in R'(Q_h^{\uparrow})$ and that $Q_{f'} = R'(Q_h^{\uparrow})$, therefore, $R'(Q_h^{\uparrow})$ is Σ_u-invariant by Theorem 6 of [44]. This in turn implies that the associated subautomaton $f'/G = R'(H^{\uparrow})$ is state controllable in G. Therefore, we have demonstrated that $R'(H^{\uparrow})$ is state controllable and state observable in G. As such we can employ $R'(H^{\uparrow})$ as our filter automaton $H_{\mathit{filt},j}$ and can implement our filter law $\mathcal{H}_{\mathit{filt},j}$ according to equation (4.13). Note, the hypothetical state-feedback law f' working under full observation achieves the same behavior as the covering-based law of equation (4.13) under partial observation.

The only property that has not been addressed yet is blocking. By construction, the marking of $R'(H^{\uparrow})$ is consistent with the marking of G. Taking the trim of $R'(H^{\uparrow})$ makes the subautomaton nonblocking. In this instance, the trim operation will simply remove those states of $R'(H^{\uparrow})$ that are blocking. This, however, can destroy state controllability and/or state observability. Therefore, following the trim operation, it may be necessary to repeat the $\uparrow$ and R' operations again. A summary of this algorithm is given below.

Algorithm 4.27. *Filter Law Construction*
Step 1: Given a blocking automaton $G = B_{j,a}$, let the subautomaton $H = \mathit{trim}(G)$ be our "specification."
Step 2: Find $Q_h^{\uparrow}$, the supremal Σ_u-invariant subset of Q_h. The algorithm of [56] can be employed. Let $H^{\uparrow}$ be the subautomaton with the state set corresponding to $Q_h^{\uparrow}$.
Step 3: Construct the mapping $I_{H^{\uparrow}}$ of indistinguishable states of the subautomaton $H^{\uparrow}$. The algorithm of [71] can be used for this purpose and has polynomial complexity in the number of events and states.
Step 4: Construct the sets of prohibited transitions $A'_{H^{\uparrow}}(q)$ to satisfy equation (4.20). Algorithm 4.21 can be employed.

Step 5: Follow Algorithm 4.23 to construct the state set $R'(Q_h^{\uparrow})$ and subautomaton $R'(H^{\uparrow})$.
Step 6: If the subautomaton $R'(H^{\uparrow})$ is nonblocking, then this represents our filter automaton $H_{\mathit{filt},j}$ and we are done. Otherwise, redefine $H = trim(R'(H^{\uparrow}))$ and return to Step 2. $\diamond$

In the above, Step 3 and Step 4 could be addressed simultaneously by a single algorithm. The end result of this procedure is a (possibly nondeterministic) subautomaton $H_{\mathit{filt},j}$ that satisfies requirements $R1'$, $R2'$, and $R3'$ with respect to the blocking automaton $G = B_{j,a}$.

Each step in the above procedure has polynomial complexity in the number of states and transitions of the initial automaton, therefore, each iteration of the algorithm will also have polynomial complexity. In addition, each pass through the algorithm either removes a state from the subautomaton or reaches a fixpoint. Assuming the initial automaton has a finite number of states, at most n iterations must be performed where n is the number of states in the initial automaton. Therefore, the overall complexity of the algorithm is polynomial. The resulting coordinating filter law produces more permissive control than existing state-feedback approaches, but it is not necessarily maximal. This fact is demonstrated by Remark 4.31 following the example of the next section.

4.5.4 Filter construction example

The following example helps to illustrate our filter construction procedure introduced in Algorithm 4.27 of the previous section.

Example 4.28. Consider the blocking automaton G pictured on the left of Fig. 4.6 with event set partitioned into controllable and uncontrollable events as follows, $\Sigma_c = \{a, b, c, d, f\}$ and $\Sigma_u = \{e, \tau\}$. By Step 1 of Algorithm 4.27, our specification $H_0 = trim(G)$ is a subautomaton of G where the blocking state 9 has been removed. Since state 8 of H_0 then requires that the uncontrollable event e be disabled, the $\uparrow$ operation of Step 2 will remove state 8 resulting in the subautomaton $H_0^{\uparrow}$.

Following Step 3 of Algorithm 4.27, the mapping $I_{H_0^{\uparrow}}$ representing which states are indistinguishable is then constructed. We will represent $I_{H_0^{\uparrow}}$ as Table 4.1 that was constructed using the algorithm from [71]. Specifically, the left-hand column

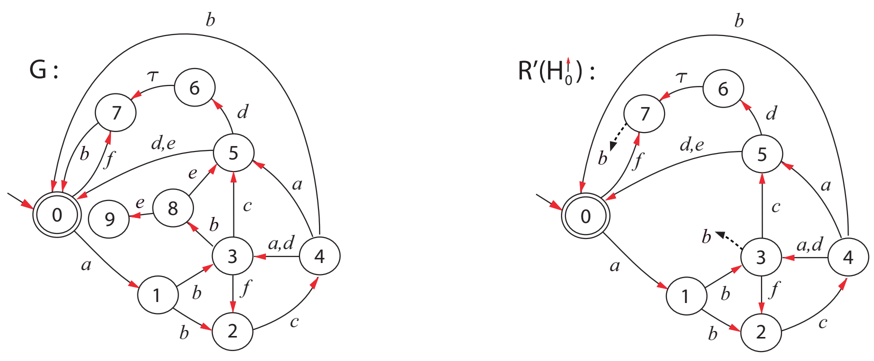

Figure 4.6: Filter construction example

enumerates each state q in the state space of $H_0^{\uparrow}$ and the center column lists the corresponding set of indistinguishable states $I_{H_0^{\uparrow}}(q)$.

Table 4.1: Table representing the map $I_{H_0^{\uparrow}}$

q	$I_{H_0^{\uparrow}}(q)$	$A'_{H_0^{\uparrow}}(q)$
0	0, 3, 6, 7	
1	1	
2	2, 3, 7	
3	3, 2, 5, 6, 7, 0	b
4	4, 5	
5	5, 4, 3	
6	6, 7, 3, 0	
7	7, 6, 3, 0, 2	b

Step 4 of Algorithm 4.27 then constructs the sets $A'_{H_0^{\uparrow}}(q)$. Examining state 3, $\delta_g(3,b) = 8 \in Q_g$, but $8 \notin Q^{\uparrow}_{H_0}$, therefore, $b \in A'_{H_0^{\uparrow}}(3)$. It then follows that b is also in the set $A'_{H_0^{\uparrow}}(7)$ since b is defined at state 7 and $7 \in I_{H_0^{\uparrow}}(3)$. Since there are no other states in $I_{H_0^{\uparrow}}(3)$ or $I_{H_0^{\uparrow}}(7)$ for which a b event is defined, and since there are no other events actively disabled by $H_0^{\uparrow}$, all the sets $A'_{H_0^{\uparrow}}(q)$ are now completely defined. Step 5 of the algorithm then applies the transformation R'. Since $\delta_g(7,b) = 0 \in Q^{\uparrow}_{H_0}$ and $b \in A'_{H_0^{\uparrow}}(7)$, event b must be disabled at state 7. The resulting subautomaton $R'(H_0^{\uparrow})$ is displayed on the right-hand side of Fig. 4.6.

According to Step 6, since $R'(H_0^{\uparrow})$ is blocking, we must then take the trim and start over at Step 2 of the algorithm. Let $H_1 = \mathit{trim}(R'(H_0^{\uparrow}))$ and refer to the left-hand side of Fig. 4.7 for an illustration.

Since the only actively disabled events b, d, and f are controllable, the $\uparrow$ operation does not remove any states, $H_1^{\uparrow} = H_1$. We now construct the map $I_{H_1^{\uparrow}}$; the result is shown below in Table 4.2.

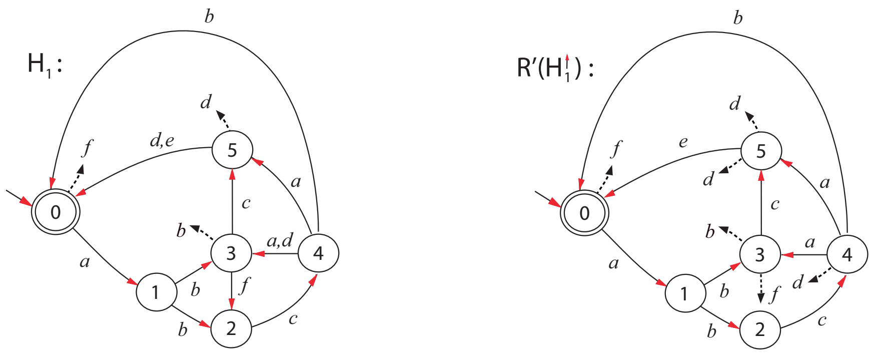

Figure 4.7: Filter construction example

Table 4.2: Table representing the map $I_{H_1^{\uparrow}}$

q	$I_{H_1^{\uparrow}}(q)$	$A'_{H_1^{\uparrow}}(q)$
0	0, 3	f
1	1	
2	2, 3	
3	3, 2, 5, 0	b, f
4	4, 5	d
5	5, 4, 3	d

Next we build the sets $A'_{H_1^{\uparrow}}(q)$. Noting which transitions of G are not included in $H_1^{\uparrow}$ allows us to determine that $f \in A'_{H_1^{\uparrow}}(0)$, $b \in A'_{H_1^{\uparrow}}(3)$, and $d \in A'_{H_1^{\uparrow}}(5)$. Examining the table describing the mapping $I_{H_1^{\uparrow}}$, we then also have that $f \in A'_{H_1^{\uparrow}}(3)$ since $3 \in I_{H_1^{\uparrow}}(0)$ and $\delta_g(3, f)!$. Likewise, $d \in A'_{H_1^{\uparrow}}(4)$ since $4 \in I_{H_1^{\uparrow}}(5)$ and $\delta_g(4, d)!$.

Now according to Step 5 we construct $R'(H_1^{\uparrow})$. First, note that one instance of a d event is disabled at state 5 according to $H_1^{\uparrow}$, therefore, the remaining d transition at state 5 must also be disabled. Since $\delta_g(3, f) = 2 \in Q_{H_1}^{\uparrow}$ and $f \in A'_{H_1^{\uparrow}}(3)$, the f event at state 3 must also be disabled. Likewise, the d event at state 4 must be disabled. The resulting $R'(H_1^{\uparrow})$ is shown on the right-hand side of Fig. 4.7. Since this subautomaton is nonblocking, we are done. Therefore, our deterministic filter law $\mathcal{H}_{filt}$ is represented by the nondeterministic automaton $H_{filt} = R'(H_1^{\uparrow})$ that satisfies requirements $R1'$, $R2'$, and $R3'$. $\diamond$

In view of the above example, we make the following observations regarding the new results presented in this section.

Remark 4.29. In traditional state-feedback control employing a mask M, states 3 and 4 would be in the same partition since state 3 is indistinguishable from state 5 and state 5 is indistinguishable from state 4. Therefore, traditional approaches would have disabled the b event at state 4. This example along with equation (4.22) demonstrates the advantage of our covering-based approach over the state-feedback control approaches of [43] [69]. $\diamond$

Remark 4.30. Our approach, however, still produces a static control law with respect to the those events feasible at a given state. For example, if for some reason the b event at state 3 needed to be disabled following the string ab, but not following the string $abca$, our covering-based control law would not be able to make that distinction. This more restrictive control law is again chosen to avoid the exponential complexity that would come with implementing an event-feedback law. $\diamond$

Remark 4.31. If we had recalculated the mapping I after the event d at state 5 was disabled, then states 0 and 3 would no longer be indistinguishable. This new I mapping would then not have required that the f event at state 3 be disabled. If we allowed the I mapping to change within the calculation of the transformation R', then the resulting subautomaton would be dependent on the order in which the states were addressed. This dependence is an issue common also to event-feedback approaches to control under partial observation. $\diamond$

Remark 4.32. We have shown that our covering-based approach is an improvement over the state-feedback approach proposed by [69]. The approach of [69] in turn has been shown to provide more permissive control than the construction of the supremal controllable and normal subset of states presented in [43]. Namely, if for all q in the allowed state set the set $M^{-1}(M(q))$ is a subset of the allowed state set, where

$$M^{-1}(M(q)) = \{q' \mid M(q) = M(q')\},$$

then the state set is normal. The example of this subsection, therefore, shows how the approach of [43] is more restrictive than our approach. Since in our example states 0 and 6 are reached by the same string $abcd$, they both have the same observation under M, but state 0 is in the state set $R'(Q^{\uparrow}_{H_1})$ while state 6 is not. Therefore, the state set $R'(Q^{\uparrow}_{H_1})$ violates normality. Construction of the supremal normal and controllable subset of states would then require removal of state 0 leading to the empty automaton. $\diamond$

4.6 Flexible Manufacturing System (FMS) Example

In this section we will demonstrate the EBCR approach for generating nonblocking modular supervisory control through the FMS example employed throughout this dissertation. The details of this manufacturing system including the component automata models were presented in Section 3.4.2.

Following Algorithm 4.6 of Section 4.2, the first step is to generate a set of local modular supervisory controllers. Specifically, H_2 is the automaton representation of the supervised module corresponding to specification *B2* and subplant $G'_2 = \mathit{Con2} \| \mathit{Robot}$. Likewise, H_4 corresponds to specification *B4* and subplant $G'_4 = \mathit{Robot} \| \mathit{Lathe}$, H_6 to specification *B6* and subplant $G'_6 = \mathit{Robot} \| AM$, H_7 to specification *B7* and subplant $G'_7 = G'_6 \| \mathit{Con}3$, and H_8 to specification *B8* and subplant $G'_8 = \mathit{Con3} \| PM$.

We will ultimately choose to address the supervisors in the order $H_7 \rightarrow H_6 \rightarrow H_4 \rightarrow H_8 \rightarrow H_2$. Recognizing that the plant components making up G'_6 are a subset of the plant components making up G'_7, we have that $\mathcal{L}(H_7) \subseteq \mathcal{L}(G'_6)$ and can employ a reduced supervisor for H_6 based on the logic of Proposition 4.12. We will denote this reduction C_6. While the original supervisor H_6 has 28 states and 71 transitions, its reduction C_6 can be represented by an automaton with 2 states and 3 transitions.

Step 2 then instructs us to generate conflict-equivalent abstractions for each supervised subsystem employing Algorithm 4.2. Initially module H_2 is represented by an automaton with 12 states and 24 transitions that we will write 12(24). If a transition is self-looped at every state, then we will not count it in the total number of transitions. Since events 21 and 22 are relevant to only the current subsystem H_2, we will "hide" them, that is, we will replace their occurrence in H_2 by the silent event τ. As such, we can apply the rules from Section 4.3.2 to generate the abstraction $H_{2,a}$ that has size 4(6). Similarly, events 51, 52, 53, and 54 are relevant to only subsystem $H_4 : 9(10)$, leading to the abstraction $H_{4,a} : 6(7)$. The relevant event set of $C_6 : 2(3)$ is contained in the relevant event set of $H_7 : 80(259)$ and hence can be reduced no further. The reduced size of the relevant event set of C_6 as compared to H_6, however, does allow us to hide events 61, 64, 65, and 66 in H_7. The resulting conflict-equivalent abstraction is then $H_{7,a} : 31(115)$. Also, $H_8 : 6(6)$ has events 81 and 82 that can be hidden, leading to $H_{8,a} : 4(4)$.

The next step is to pick an initial subsystem. Some considerations for how to pick a "good" ordering of subsystems will be discussed at the end of this section, but for now, we will choose $H_{7,a}$ as our starting point. Following Step 4 of the procedure, we will then choose the next subsystem to be C_6 and will generate the composition

$H_{7,a} \| C_6 : 136(405)$. The next step is to check for blocking. Since it turns out the $H_{7,a} \| C_6$ is nonblocking, we skip to Step 7. At this point, event 63 is not relevant to any of the remaining subsystems and hence can now be hidden. This leads to the abstraction $(H_{7,a} \| C_6)_a : 27(79)$.

Since other subsystems still have not yet been addressed, we return to Step 4 and add $H_{4,a}$ to the composition, $(H_{7,a} \| C_6)_a \| H_{4,a} : 45(120)$. Note, the process of abstraction has led $H_{4,a}$ and the resulting composition to be nondeterministic. At this point we again check for blocking. Since the composition is nonblocking, we skip to Step 7. All the relevant events of the composition $(H_{7,a} \| C_6)_a \| H_{4,a}$ are still relevant to the remaining subsystems, therefore, no more events can be hidden at this point. However, the composition can still be reduced further by applying the conflict equivalence preserving rules introduced earlier, $((H_{7,a} \| C_6)_a \| H_{4,a})_a : 42(115)$.

Returning to Step 4 again, $H_{8,a}$ is added to the composition. The result $((H_{7,a} \| C_6)_a \| H_{4,a})_a \| H_{8,a} : 61(52)$ turns out to be blocking. Therefore, according to Step 6 of the procedure, a filter must be built to resolve the conflict. The blocking composition $B_{1,a} = ((H_{7,a} \| C_6)_a \| H_{4,a})_a \| H_{8,a}$ is in essence the uncontrolled "plant" and we can apply Algorithm 4.27 to construct the subautomaton of $B_{1,a}$ that will serve as the filter which supervises the system and prevents the blocking. Taking the trim of $B_{1,a}$ removes two blocking states, but leaves the resulting subautomaton not state controllable with respect to $B_{1,a}$. Applying Steps 2 through 5 of Algorithm 4.27 leaves a state controllable and state observable automaton, but it is again blocking. Taking the trim and performing another iteration of the algorithm leaves us a nonblocking subautomaton that is state controllable and state observable with respect to $B_{1,a}$. This subautomaton has 41 states and 120 transitions and serves as our coordinating filter law $H_{\mathit{filt,1}}$.

Since a determinized version of $H_{\mathit{filt,1}}$ composed with $(H_6 \| H_7)_a \| H_{4,a} \| H_{8,a}$ is bisimulation equivalent to $H_{\mathit{filt,1}}$, we will replace the composition $(H_6 \| H_7)_a \| H_{4,a} \| H_{8,a}$ by $H_{\mathit{filt,1}}$ as we proceed to Step 7. At this point, the events 71, 72, 73, and 74 have become local and can thus be hidden leading to the abstraction $(H_{\mathit{filt,1}})_a : 9(17)$.

Returning to Step 4 once more, the last subsystem $H_{2,a}$ is added to the composition, $(H_{\mathit{filt,1}})_a \| H_{2,a} : 9(17)$. Since the composition is nonblocking and no further subsystems remain, we are done. The resulting modular control achieved by the five original modular supervisors along with the conflict-resolving law $\mathcal{H}_{\mathit{filt,1}}$ satisfies the given specifications in a nonblocking manner and is nonempty. Table 4.3 summarizes the details of the procedure applied in this example.

Table 4.3: Application of Algorithm 4.6 to FMS example

Step	Automaton Built	States (Transitions)	Notes
1	H_2	12(24)	modular supervisors are built
	H_4	9(10)	
	H_6	28(74)	
	H_7	80(259)	note $G_7' \| B_7 : 128(420)$
	H_8	6(6)	
	$H_6 \rightarrow C_6$	2(3)	supervisor reduction
2	$H_2 \rightarrow H_{2,a}$	4(6)	{21,22} hidden
	$H_4 \rightarrow H_{4,a}$	6(7)	{51,52,53,54} hidden
			$H_{4,a}$ is nondeterministic
	$H_7 \rightarrow H_{7,a}$	31(115)	{61,64,65,66} hidden
	$H_8 \rightarrow H_{8,a}$	4(4)	{81,82} hidden
3			$H_{7,a}$ chosen as the initial subsystem
4	$H_{7,a} \| C_6$	53(174)	C_6 chosen from neighboring subsystems
5			composition is nonblocking
6			this step is skipped
7	$H_{7,a} \| C_6 \rightarrow (H_{7,a} \| C_6)_a$	27(79)	{63} hidden
4	$(H_{7,a} \| C_6)_a \| H_{4,a}$	45(120)	$H_{4,a}$ chosen from neighboring subsystems
5			composition is nonblocking
6			this step is skipped
7	$(H_{7,a} \| C_6)_a \| H_{4,a} \rightarrow$ $((H_{7,a} \| C_6)_a \| H_{4,a})_a$	42(115)	no further events hidden at this point
4	$((H_{7,a} \| C_6)_a \| H_{4,a})_a \| H_{8,a}$	61(52)	$H_{8,a}$ chosen from neighboring subsystems
5			composition is blocking
6	$H_{filt,1}$	41(120)	employ Algorithm 4.27 $H_{filt,1}$ replaces $((H_{7,a} \| C_6)_a \| H_{4,a})_a \| H_{8,a}$
7	$H_{filt,1} \rightarrow (H_{filt,1})_a$	9(17)	{71,72,73,74} hidden
4	$(H_{filt,1})_a \| H_{2,a}$	9(17)	$H_{2,a}$ chosen from neighboring subsystems
5			composition is nonblocking
6			this step is skipped
7			no further subsystems left, done

It turns out that the resulting modular solution is more restrictive than the monolithic solution in that it allows only five pieces to be operated on by the FMS at a given time, while the monolithic solution allows six pieces to be active at once. The loss of optimality of our approach arises in two ways due to the hiding of events. Namely, hiding an event means that we lose the ability to disable it. Also, the hiding of events causes us to lose information about what state the underlying plant is in, and as such forces us to employ a more conservative control law. This loss of optimality, however, is often worth the reduction in complexity the modular approach provides. Specifically, a measure of the complexity of the modular solution in the above example is that the largest automaton that had to be built had 128 states and 420 transitions. This automaton was constructed in the process of generating the modular supervisor H_7. In the monolithic approach, the composition of all the machines and buffers leads to an automaton with 13,248 states and 46,424 transitions. While the size of the resulting automata does not account for the complexity of the algorithms involved in generating the control laws and the abstractions, it does give some indication of the benefits of this approach. Specifically, all algorithms employed in generating the monolithic and EBCR solutions are known to have polynomial complexity, except for the generation of the conflict-equivalent abstraction. This is an area that needs to be investigated further.

Noting that the largest automaton constructed in the modular approach was the result of building a single modular supervisor, the overall complexity could be reduced further by employing abstraction in the construction of the modular supervisors, in addition to using abstraction in the construction of the conflict-resolving filters. Specifically, results for the construction of individual supervisors could be borrowed from [17] [26] [75].

Additionally, an improved modular solution can often be arrived at by changing the order in which subsystems are addressed or by changing the set of events that are considered silent along the way. For instance, if in the above example we had chosen not to hide the events 61, 63, and 65, the resulting modular solution would have allowed six pieces to be operated on by the FMS at a given time, just like the monolithic solution. This solution would have resulted in slightly larger automata, with the largest automaton constructed having 150 states and 352 transitions.

One limitation of this approach is that there is not a single approach to ordering subsystems that will result in the "best" overall solution. Some ordering heuristics that can help keep the overall complexity of the procedure down include first choosing subsystems that are either small or that offer the possibility of larger reduction. The work of [21] offers a sizable survey of ordering heuristics applied to a variety of examples. Implementation of a conflict-equivalent abstraction also relies on an incomplete set of heuristic rules which do not in general provide a unique result.

Some heuristics for improving the optimality of this approach include "hiding" fewer events. In this way, reduction is traded for optimality. It is also possible to change the outcome by not building a filter immediately following the detection of blocking. The idea here is that sometimes conflict is resolved by composition with other subsystems and ultimately a filter is not needed. The advantage of waiting is that a filter cannot disable uncontrollable events and hence sometimes must remove states from an automaton, while interaction with other subsystems can prevent an uncontrollable event from happening in the first place so that it does not need to be actively disabled. The drawback of waiting to build the filter is that often the process of abstraction hides transitions that could be used to prevent blocking or violations of controllability. Ordering heuristics remain an open area for investigation.

This approach in general is well-suited to systems that are loosely coupled, as are other modular approaches to control. If a component specification shares relevant events with all plant components, then the achievable reduction will likely be modest, though in most cases it will still result in smaller automata being built than with the monolithic solution.

4.7 Chapter Summary

This chapter has proposed a new approach for resolving conflict among traditionally-built modular supervisors. Requirements are presented for conflict-resolving filter laws that guarantee safe nonblocking control. A methodology for building covering-based filter laws that meet the prescribed requirements and avoids exponential complexity is also proposed. Additionally, a manufacturing example is presented showing the overall potential of the EBCR approach.

The modular architecture of the EBCR approach with its additional level of coordinating control is similar to previous works [17] [74] [76]. The approach of this chapter is unique, however, in that it employs conflict-equivalent abstractions in generating the coordinating control. Conflict-equivalent abstractions offer the potential for a greater reduction in state-size than observer-type abstractions that are employed in most prior work. Drawbacks of a conflict-equivalent abstraction are that it can introduce nondeterminism and it is not as straightforward to implement.

Conflict-equivalent abstractions were employed by [21] to reduce the complexity of verifying nonconflict. The work of [21] incrementally constructs the global system applying abstraction each step along the way. The EBCR approach builds off this work, where we go beyond detecting conflict to actually resolving it if it is present. Conflict-equivalent abstractions were also employed in [48] for constructing nonconflicting modular supervisors. The work of [48] proposes a methodology similar to the IHSC approach of Chapter 3, but does not specify how to construct the supervisors based on the abstracted models.

The covering-based approach employed in building the filters of this chapter is also a contribution in that it generates a less restrictive control law than is achieved by existing state-feedback methodologies for partially observed systems. Existing state-feedback approaches to control under partial observation require that the control be consistently applied at states in the same observation partition [43] [69]. In the approach of this chapter, we rather generate a covering of the state space that allows for the application of a less restrictive control law.

One drawback of the architecture of the EBCR approach, and of work like [17], is that sharing is increased between the modules, thereby limiting which events can be hidden. Referring to the FMS example as shown in Fig. 1.4, one can see those events associated with the *Robot* subplant are shared between four of the five modular supervisors. Therefore, those events that are relevant to *Robot* cannot be hidden until the conflict has been resolved among the four modular supervisors that control it. This problem has been addressed somewhat by the logic demonstrated in Section 4.3.4 that shows that a reduced supervisor can be employed when the associated plant has been addressed by previously examined modular supervisors. This logic has not been proven, however, when only a portion of the associated plant has been addressed by preceding modular supervisors. Therefore, a direction for future work would be to develop the theory for generating reduced supervisors based on only a portion of a plant.

Another important direction for future work is to develop a better understanding of how conflict-equivalent abstractions can be generated. Specifically, it would be useful to investigate the complexity associated with generating a conflict-equivalent abstraction based on the heuristic rules of [19] [21]. It could also be interesting to explore the possibility that other rules could be developed for generating the abstraction. Other directions for future work include investigating different ordering heuristics. Work could also be done with regard to finding new ways to construct the conflict-resolving filters. Finally, the results of this work could be combined with other modular and hierarchical approaches to supervisory control to achieve even greater reduction in complexity.

CHAPTER 5

Multi-Level Interface-Based Control

In this chapter an approach to supervisory control we will refer to as Multiple-Level Interface-Based Control (MLIBC) is proposed to mitigate the complexity problems associated with the analysis and design of DES. This approach partitions the global system into modules and adds interfaces between neighboring modules. The additional structure provided by these interfaces allows global system properties such as nonblocking and controllability to be guaranteed solely based on the satisfaction of local requirements. The ability to verify properties and to design supervisors without having to construct the monolithic system helps to mitigate the state-space explosion problem. This type of architecture also helps to improve reconfigurability since, if a single module is modified, the global system does not need to be reanalyzed. In this case, only the modified module must be reanalyzed with respect to its interfaces. The results of this chapter generalize the work of Leduc [38] [39] [41] that was developed for a two-level interface-based architecture.

An interface-based approach to control, however, does have its drawbacks. Namely, the increased restrictiveness of the interfaces can result in suboptimal control. In many cases, this exchange of optimality for a reduction in computational complexity and improved reconfigurability may be acceptable.

This chapter specifically provides local conditions that guarantee global nonblocking and language controllability of a multiple-level system with interfaces like the one pictured in Fig. 5.1. This generalized architecture allows the system to be partitioned into smaller modules than the two-level interface-based architecture first introduced in [38] [41], thereby further limiting the complexity of analysis and design. We also demonstrate that the interface consistency requirements of [38] [41] can be relaxed. This change makes the necessary requirements easier to satisfy, especially in the multiple-level case.

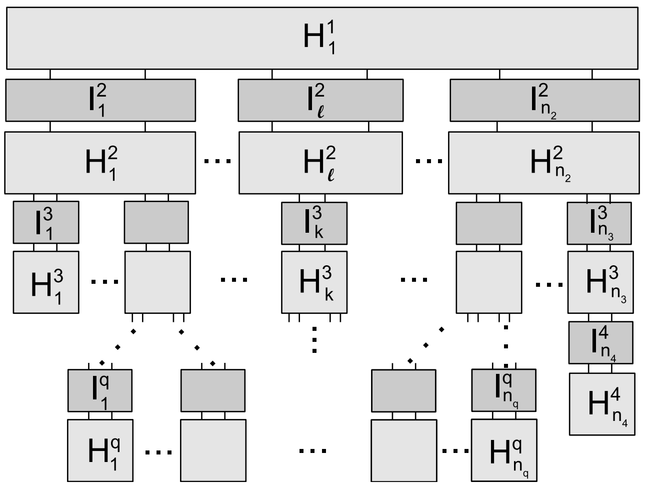

Figure 5.1: Illustration of the multiple-level architecture

In this chapter, we also present results for synthesizing modular supervisors in the multiple-level architecture that are maximally permissive with respect to a given specification and set of interfaces. These results directly follow the supervisor synthesis approach developed in [39] for the two-level case. A final contribution of this chapter is to propose an approach to interface synthesis, something that has not yet been addressed in the literature. While computationally expensive, the approach provides insight into this challenging problem. In existing works, interfaces have been constructed heuristically based on the designer's understanding of the system.

The results of this chapter assume that the DES are modeled by deterministic automata. It is also assumed that the automata may have different event sets, though the languages employed in the chapter have all been lifted to the global alphabet Σ.

The organization of the remainder of this chapter is as follows. Section 5.1 introduces notation and definitions necessary for a hierarchical interface-based approach to supervisory control. Section 5.2 demonstrates that the global properties of nonblocking and language controllability can be verified through local analysis. Sections 5.3 and 5.4 outline our approaches for supervisor and interface synthesis respectively. Section 5.5 demonstrates the application of this architecture to the FMS example, while Section 5.6 concludes the chapter with a summary of its contributions.

5.1 Hierarchical Interface-Based Supervisory Control

We will now define the notation and definitions necessary for proving results with regard to a multiple-level application of hierarchical interface-based supervisory control. We will specifically assume a connected tree architecture with a single root node. Figure 5.1 illustrates this situation. Our component-wise specified system is split up into modules, each consisting of a plant G_k^i and a supervisor S_k^i constructed with respect to a local specification E_k^i resulting in the closed-loop subsystem $H_k^i = G_k^i \| S_k^i$. The superscript i reflects the level of the hierarchy and takes values $\{1, \ldots, q\}$. The subscript k indicates the index within a given level and takes the values $\{1, \ldots, n_i\}$, where this set represents all modules and interfaces on a given level i, including modules and interfaces that have different corresponding higher-level neighbors.

All interaction between modules takes place through corresponding interfaces I_k^i. These interfaces restrict the behaviour of the overall system in such a way that global properties can be guaranteed by local analysis. In a sense, these interfaces may apply additional control. Figure 5.2 shows a detail of the multiple-level architecture. We will refer to the global system defined in terms of these modules and interfaces as the system Φ.

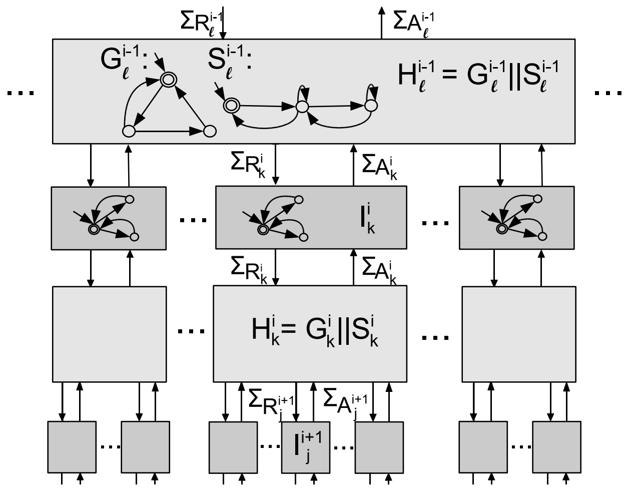

Figure 5.2: Detail of the multiple-level architecture

In this architecture, all events shared between a given module H_k^i and its higher-level neighbor are classified as either request events $\rho \in \Sigma_{R_k^i}$ or answer events $\alpha \in \Sigma_{A_k^i}$. The occurrence of each of these events must then be accepted by the corresponding interface I_k^i. Conceptually, request events are thought of as being under the control of the higher-level module and answer events as being under the control of the lower-level module. For the purposes of this chapter, we will assume the interfaces take the form of a *command-pair interface* defined below in the manner of [38].

Definition 5.1. A DES $I_k^i = (X_k^i, \Sigma_{R_k^i} \dot{\cup} \Sigma_{A_k^i}, \xi_k^i, x_{0_k}^i, X_{m_k}^i)$ is a *command-pair interface* if the following are true:

A) $\mathcal{L}(I_k^i) \subseteq \overline{(\Sigma_{R_k^i}.\Sigma_{A_k^i})^*}$

B) $\mathcal{L}_m(I_k^i) = (\Sigma_{R_k^i}.\Sigma_{A_k^i})^* \cap \mathcal{L}(I_k^i)$ $\diamond$

From the above definition it can be deduced that the relevant event set for the interface I_k^i is given as follows:

$$\Sigma_{I_k^i} := \Sigma_{A_k^i} \dot{\cup} \Sigma_{R_k^i}$$

We will also define the relevant event set of a given module H_k^i to be equal to the union of the relevant event sets of the associated plant component and specification.

$$\Sigma_{H_k^i} := \Sigma(G_k^i) \cup \Sigma(E_k^i)$$

In turn, we will assume the event sets of G_k^i, E_k^i, and S_k^i are equal to $\Sigma_{H_k^i}$, though their relevant event sets will not necessarily be the same. We will also assume that the global alphabet is partitioned as shown in equation (5.1), where the set Σ_k^i represents those events relevant to H_k^i but no other modules. The following also assumes there is only a single module on level 1, the top level.

$$\Sigma := \Sigma_1^1 \dot{\cup} \dot{\bigcup_{i=2,\ldots,q}} \left(\dot{\bigcup}_{k=1,\ldots,n_i} \left(\Sigma_k^i \dot{\cup} \Sigma_{A_k^i} \dot{\cup} \Sigma_{R_k^i} \right) \right) \tag{5.1}$$

A consequence of equation (5.1) is that each interface is completely disjoint from all other interfaces, that is, $\Sigma_{I_k^i} \cap \Sigma_{I_{k'}^{i'}} = \emptyset, \forall ((i \neq i') \vee (k \neq k'))$. We will further assume that the event set of each module H_k^i is constrained to have the partitioning given in equation (5.2). The following is consistent with the connected tree architecture of our approach. Since there are no interfaces with superscript 1 or $q+1$, we will consider $\mathcal{L}(I_k^1) = \mathcal{L}(I_j^{q+1}) = \Sigma^*$. This therefore implies that these interfaces have no relevant events and hence $\Sigma_{I_k^1} = \Sigma_{I_j^{q+1}} = \emptyset$.

$$\begin{aligned} \Sigma_{H_k^i} &= \Sigma_k^i \dot{\cup} \Sigma_{I_k^i} \dot{\cup} \dot{\bigcup}_{j \in J_k^i} \Sigma_{I_j^{i+1}} \\ \text{where} \qquad & J_k^i := \{ j \mid \Sigma_{H_k^i} \cap \Sigma_{I_j^{i+1}} \neq \emptyset \} \end{aligned} \tag{5.2}$$

In the above, the index sets J_k^i for modules on the i^{th} level partition the set $\{1, \ldots, n_{i+1}\}$ into disjoint subsets. An implication of equation (5.2) is that each module H_k^i may share relevant events only with modules from the $i+1$ level and a single module from the $i-1$ level. We will employ script letters to represent the languages generated by the corresponding automata lifted to the global alphabet. This convention is employed in the following definitions.

$$\begin{aligned}
P_{H_k^i} &: \Sigma^* \to \Sigma_{H_k^i}^* & P_{I_k^i} &: \Sigma^* \to \Sigma_{I_k^i}^* \\
\mathcal{H}_k^i &:= P_{H_k^i}^{-1}(\mathcal{L}(H_k^i)) & \mathcal{H}_{m_k}^i &:= P_{H_k^i}^{-1}(\mathcal{L}_m(H_k^i)) \\
\mathcal{G}_k^i &:= P_{H_k^i}^{-1}(\mathcal{L}(G_k^i)) & \mathcal{G}_{m_k}^i &:= P_{H_k^i}^{-1}(\mathcal{L}_m(G_k^i)) \\
\mathcal{E}_k^i &:= P_{H_k^i}^{-1}(\mathcal{L}(E_k^i)) & \mathcal{E}_{m_k}^i &:= P_{H_k^i}^{-1}(\mathcal{L}_m(E_k^i)) \\
\mathcal{S}_k^i &:= P_{H_k^i}^{-1}(\mathcal{L}(S_k^i)) & \mathcal{S}_{m_k}^i &:= P_{H_k^i}^{-1}(\mathcal{L}_m(S_k^i)) \\
\mathcal{I}_k^i &:= P_{I_k^i}^{-1}(\mathcal{L}(I_k^i)) & \mathcal{I}_{m_k}^i &:= P_{I_k^i}^{-1}(\mathcal{L}_m(I_k^i))
\end{aligned}$$

The following requirements modified from [40] will be employed to guarantee global properties through local analysis for a given set of DES. Specifically, each property will be checked with respect to the *i-k*th module and those interfaces with which it shares relevant events. Refer again to Fig. 5.2 to help visualize the structure of a single module. In the following, we will define the event set $\Sigma_{L_k^i} = \Sigma_{H_k^i} - \Sigma_{I_k^i}$ to be those events relevant to the module H_k^i that are not relevant to the module on the next higher level of the hierarchy.

Definition 5.2. The multiple-level interface system Φ is said to be *multi-level nonblocking* if for all $i \in \{1, \ldots, q\}$ and for all $k \in \{1, \ldots, n_i\}$ corresponding to each i, the following condition is satisfied:

$$\overline{\mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}} = \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \quad \diamond$$

Definition 5.3. The multiple-level interface system Φ is said to be *multi-level controllable* with respect to the alphabet partitions given by (5.1) and (5.2), if for all $i \in \{1, \ldots, q\}$ and for all $k \in \{1, \ldots, n_i\}$ corresponding to each i, the following conditions are satisfied:

i) The event set of G_k^i and S_k^i is $\Sigma_{H_k^i}$ and the event set of I_k^i is $\Sigma_{I_k^i}$.

ii) $(\forall s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{S}_k^i)$

$$\mathrm{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_k^i \cap \mathcal{I}_k^i}(s) \quad \diamond$$

Definition 5.4. The multiple-level interface system Φ is said to be *multi-level consistent* with respect to the alphabet partitions given by (5.1) and (5.2), if for all $i \in \{1, \ldots, q\}$ and for all $k \in \{1, \ldots, n_i\}$ corresponding to each i, the following conditions are satisfied:

Multi-level Properties

1) The event set of H_k^i is $\Sigma_{H_k^i}$.

2) I_k^i is a command-pair interface.

Upper-level Property

3) $(\forall s \in \mathcal{H}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1})(\forall j \in J_k^i)$,
$\mathrm{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \subseteq \mathrm{Elig}_{\mathcal{H}_k^i}(s)$

Lower-level Properties

4) $(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.\Sigma_{L_k^i}^* \cap \mathcal{H}_k^i \cap \mathcal{I}_k^i)(\forall \rho \in \Sigma_{R_k^i})$,
$s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}$

5) $(\forall s \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1})(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i})$,
$s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}$

6) $(\forall s \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1})$,
$s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ \ sl \in \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}$ $\diamond$

In words, Point 3 of Definition 5.4 requires that the i-k^{th} module be $\Sigma_{A_j^{i+1}}$-controllable with respect to each of its lower-level interfaces I_j^{i+1}. Points 4 and 5 require that request and answer events, respectively, be reachable in a module by events not shared with the corresponding upper-level module. Point 6 requires that if a string is marked and accepted by an interface, then it can be extended to a marked string in the corresponding lower-level module by events that again are not shared with the upper-level module. Some of these requirements are discussed further in Section 5.2.1.

5.2 Global Nonblocking and Controllability

In this section we will present the main results of this chapter. Specifically, we will show that if a set of local conditions based on the definitions of Section 5.1 are satisfied, then the global multiple-level system is nonblocking and the conjunction of modular supervisors and interfaces is language controllable in the sense of equation (2.1) with respect to the global plant. These results are presented in Theorem 5.5 and

Theorem 5.6 given below. Their proofs are presented after some important special cases are discussed.

Theorem 5.5. *If the multiple-level interface system* Φ *is* multi-level nonblocking *and* multi-level consistent *with respect to the alphabet partitions given by (5.1) and (5.2), then the complete system is nonblocking:*

$$\overline{\mathcal{H}_m^1 \cap \mathcal{H}_m^2 \cap \mathcal{I}_m^2 \cap \ldots \cap \mathcal{H}_m^q \cap \mathcal{I}_m^q} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \ldots \cap \mathcal{H}^q \cap \mathcal{I}^q$$

where

$$\mathcal{H}_m^i = \mathcal{H}_{m_1}^i \cap \ldots \cap \mathcal{H}_{m_{n_i}}^i, \quad \mathcal{I}_m^i = \mathcal{I}_{m_1}^i \cap \ldots \cap \mathcal{I}_{m_{n_i}}^i$$
$$\mathcal{H}^i = \mathcal{H}_1^i \cap \ldots \cap \mathcal{H}_{n_i}^i, \qquad \mathcal{I}^i = \mathcal{I}_1^i \cap \ldots \cap \mathcal{I}_{n_i}^i$$

Theorem 5.6. *If the multiple-level interface system* Φ *is* multi-level controllable *with respect to the alphabet partitions given by (5.1) and (5.2), then the supervisor language* $\mathcal{S} = \mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2 \cap \ldots \cap \mathcal{S}^q \cap \mathcal{I}^q$ *is* Σ_u*-controllable with respect to the plant language* $\mathcal{G} = \mathcal{G}^1 \cap \ldots \cap \mathcal{G}^q$.

Where: $\mathcal{S}^i = \mathcal{S}_1^i \cap \ldots \cap \mathcal{S}_{n_i}^i$, $\mathcal{I}^i = \mathcal{I}_1^i \cap \ldots \cap \mathcal{I}_{n_i}^i$, *and* $\mathcal{G}^i = \mathcal{G}_1^i \cap \ldots \cap \mathcal{G}_{n_i}^i$

5.2.1 Two-level case

In this subsection we present the following results which are special cases of Theorem 5.5 and Theorem 5.6 for a two-level system. Specifically, Theorem 5.10 is a result modified from [40] for the new interface consistency definition of Section 5.1, while Theorem 5.11 is taken directly from [40].

Consider a two-level system consisting of a single high-level module H^1, and a series of low-level modules $H_1^2, \ldots, H_n^2$, and interfaces $I_1^2, \ldots, I_n^2$. Examining the notion of multi-level nonblocking introduced in Definition 5.2 for each of the two levels, one can see this definition reduces to the *level-wise nonblocking* definition of the two-level case presented in [40] and repeated below. In examination of Definition 5.2, we assume $\mathcal{I}^1 = \Sigma^*$ and $\mathcal{I}_j^3 = \Sigma^*$ for all j, since there are no interfaces above the first level of the hierarchy or below the second level. This convention for interfaces is used in examination of multi-level controllability also.

Definition 5.7. [40] A two-level interface system composed of DES $H^1, H_1^2, I_1^2, \ldots, H_n^2, I_n^2$, is said to be *level-wise nonblocking* if the following conditions are satisfied:

i) $\overline{\mathcal{H}^1_m \cap \bigcap_{j=1,\ldots,n} \mathcal{I}^2_{m_j}} = \mathcal{H}^1 \cap \bigcap_{j=1,\ldots,n} \mathcal{I}^2_j$

ii) $\overline{\mathcal{H}^2_{m_k} \cap \mathcal{I}^2_{m_k}} = \mathcal{H}^2_k \cap \mathcal{I}^2_k, \ \forall k \in \{1, \ldots, n\}$ $\diamond$

Similarly, we can examine the notion of multi-level controllability introduced in Definition 5.3. Applying Point *ii)* of the definition to the two levels of the interface system, Definition 5.3 reduces to the *level-wise controllability* definition of the two-level case presented in [40] and repeated below.

Definition 5.8. [40] A two-level interface system composed of plant components $G^1, G^2_1, \ldots, G^2_n$, supervisors $S^1, S^2_1, \ldots, S^2_n$, and interfaces $I^2_1, \ldots, I^2_n$, is said to be *level-wise controllable* with respect to the alphabet partition given by (5.1), if for all $k \in \{1, \ldots, n\}$, the following conditions are satisfied:

i) The alphabet of G^1 and S^1 is Σ_{H^1}, of G^2_k and S^2_k is $\Sigma_{H^2_k}$, and of I^2_k is $\Sigma_{I^2_k}$.

ii) $(\forall s \in \mathcal{G}^2_k \cap \mathcal{I}^2_k \cap \mathcal{S}^2_k)$,

$$\mathrm{Elig}_{\mathcal{G}^2_k}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}^2_k \cap \mathcal{I}^2_k}(s)$$

iii) $(\forall s \in \mathcal{G}^1 \cap \mathcal{I} \cap \mathcal{S}^1)$,

$$\mathrm{Elig}_{\mathcal{G}^1 \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}^1}(s)$$

where $\mathcal{I} = \mathcal{I}^2_1 \cap \ldots \cap \mathcal{I}^2_n$ $\diamond$

For the two-level case, Definition 5.4 reduces to Definition 5.9 which is analogous to the *interface consistency* definition presented in [40]. The difference between the definition presented here and the one presented in [40] is the modified Point 4.

Since the high-level module H^1 has no corresponding interface I^1 above it in the hierarchy, Points 4-6 do not need to be verified for level 1 since $\Sigma_{A^1} = \Sigma_{R^1} = \emptyset$. Likewise, since the low-level modules H^2_k do not have any interfaces I^3_j below them in the hierarchy, Point 3 of Definition 5.4 does not need to be verified for level 2 since $\Sigma_{A^3_j} = \Sigma_{R^3_j} = \emptyset$ for all j. Combining Definition 5.4 for each of the two levels leads to the following modified version of interface consistency.

Definition 5.9. A two-level interface system composed of DES $H^1, H^2_1, I^2_1, \ldots, H^2_n, I^2_n$, is said to be *interface consistent* with respect to the alphabet partition given by (5.1) if for all $k \in \{1, \ldots, n\}$, the following conditions are satisfied:

1) The event set of H^1 is Σ_{H^1} and the event set of H^2_k is $\Sigma_{H^2_k}$.

2) I^2_k is a command-pair interface.

3) $(\forall s \in \mathcal{H}^1 \cap \bigcap_{j=1,\ldots,n} \mathcal{I}^2_j)$,

$\mathrm{Elig}_{\mathcal{I}^2_k}(s) \cap \Sigma_{A^2_k} \subseteq \mathrm{Elig}_{\mathcal{H}^1}(s)$

4) $(\forall s \in (\Sigma^*.\Sigma_{A_k^2})^*.\Sigma_{L_k^2}^* \cap \mathcal{H}_k^2 \cap \mathcal{I}_k^2)(\forall \rho \in \Sigma_{R_k^2})$,
$s\rho \in \mathcal{I}_k^2 \Rightarrow (\exists l \in \Sigma_{L_k^2}^*)\ sl\rho \in \mathcal{H}_k^2 \cap \mathcal{I}_k^2$
5) $(\forall s \in \mathcal{H}_k^2 \cap \mathcal{I}_k^2)(\forall \rho \in \Sigma_{R_k^2})(\forall \alpha \in \Sigma_{A_k^2})$,
$s\rho\alpha \in \mathcal{I}_k^2 \Rightarrow (\exists l \in \Sigma_{L_k^2}^*)\ s\rho l\alpha \in \mathcal{H}_k^2 \cap \mathcal{I}_k^2$
6) $(\forall s \in \mathcal{H}_k^2 \cap \mathcal{I}_k^2)$,
$s \in \mathcal{I}_{m_k}^2 \Rightarrow (\exists l \in \Sigma_{L_k^2}^*)\ sl \in \mathcal{H}_{m_k}^2 \cap \mathcal{I}_{m_k}^2 \quad \diamond$

We can now use these definitions to present the following results that are special cases of Theorem 5.5 and Theorem 5.6 for a two-level system.

Theorem 5.10. *If the two-level interface system composed of DES $H^1, H_1^2, I_1^2, \ldots, H_n^2, I_n^2$, is* level-wise nonblocking *and* interface consistent *with respect to the alphabet partition given by (5.1), then the global system is nonblocking:*

$$\overline{\mathcal{H}_m^1 \cap \bigcap_{j=1,\ldots,n} (\mathcal{H}_{m_j}^2 \cap \mathcal{I}_{m_j}^2)} = \mathcal{H}^1 \cap \bigcap_{j=1,\ldots,n} (\mathcal{H}_j^2 \cap \mathcal{I}_j^2)$$

Proof. See proof in Appendix. □

Theorem 5.11. *[40] If the two-level interface system composed of plant components $G^1, G_1^2, \ldots, G_n^2$, supervisors $S^1, S_1^2, \ldots, S_n^2$, and interfaces $I_1^2, \ldots, I_n^2$, is* level-wise controllable *with respect to the alphabet partition given by (5.1), then the supervisor language $\mathcal{S} = \mathcal{S}^1 \cap \bigcap_{j=1,\ldots,n}(\mathcal{S}_j^2 \cap \mathcal{I}_j^2)$ is Σ_u-controllable with respect to the plant language $\mathcal{G} = \mathcal{G}^1 \cap \bigcap_{j=1,\ldots,n}(\mathcal{G}_j^2)$.*

The proof of Theorem 5.10 can be found in the Appendix and follows closely the logic presented in [37], where the only difference is that the interface consistency requirement has been relaxed. Specifically, Point 4 of Definition 5.4 has been modified from what was originally a controllability requirement to the reachability requirement prescribed in this chapter. The original Point 4 required that each low-level module $\mathcal{H}_k^2$ be $\Sigma_{R_k^2}$-controllable with respect to its interface $\mathcal{I}_k^2$:

$$(\forall s \in \mathcal{H}_k^2 \cap \mathcal{I}_k^2),\ \text{Elig}_{\mathcal{I}_k^2}(s) \cap \Sigma_{R_k^2} \subseteq \text{Elig}_{\mathcal{H}_k^2}(s) \tag{5.3}$$

The spirit of this requirement is that the low-level modules have control only over those events shared with the high-level module that are answer events. Therefore, the high-level knows that if it issues a request that is accepted by the interface, the low-level will not disable it. This requirement is mirrored by Point 3 that specifies that the high-level module $\mathcal{H}^1$ be $\Sigma_{A_k^2}$-controllable with respect to each of its interfaces $\mathcal{I}_k^2$. These requirements are at the core of what enables us to draw conclusions about the global system with only local analysis.

The new Point 4 still captures the intent of the original requirement by requiring instead that the low-level be able to reach, via a string of low-level events, each request event allowed by the interface. Therefore, even though the low-level may not allow a request event immediately (as dictated by the original controllability requirement), it will eventually be able to execute the required request event following the occurrence of a string of low-level events. Since the request event is reached by local low-level events, we know that the low-level cannot be prevented from reaching the request event by interaction with the interface or high-level module. The following example helps to illustrate the difference between the original and modified requirements.

Example 5.12. Consider the interface I and low-level module L displayed in Fig. 5.3. Let $\mathcal{I}$ and $\mathcal{L}$ be the respective generated languages lifted to the global alphabet. For the set of request events $\Sigma_R = \{r_1, r_2\}$ and answer events $\Sigma_A = \{a_1, a_2\}$, I is a command-pair interface. It can be seen by inspection that $\mathcal{L}$ is not Σ_R-controllable with respect to $\mathcal{I}$. Specifically, $r_1 \notin \text{Elig}_{\mathcal{L}}(\varepsilon)$, but $r_1 \in \text{Elig}_{\mathcal{I}}(\varepsilon)$, implying a violation of Σ_R-controllability since r_1 is not enabled at state 0 of L. Also, $r_2 \notin \text{Elig}_{\mathcal{L}}(l_1 r_1 a_1 l_2)$, but $r_2 \in \text{Elig}_{\mathcal{I}}(l_1 r_1 a_1 l_2)$, therefore implying a violation since r_2 is not enabled at state 4 of L. L can be modified to remove state 4 (and subsequently state 5) and will generate a non-trivial language such that $l_1 r_1 a_1 l_2 \notin \mathcal{L}$, but the only way to remove the string ε from $\mathcal{L}$ is to make L the empty automaton. Another possible remedy is to replace the event r_1 in the set Σ_R by the event l_1. The problem that we run into here is that it could be the case in a multiple-level architecture that l_1 was employed in an interface from a lower level of the architecture.

The original language $\mathcal{L}$, however, does satisfy the modified Point 4 with respect to the interface language $\mathcal{I}$. For example, even though the request r_1 is not enabled at state 0 of L, r_1 can be reached by low-level events. Likewise, the request event r_2 can be reached from state 4 via low-level events. $\diamond$

Point 4 of the interface consistency definition is specifically employed in Proposition 13 of [37]. A revised version of this proposition using the modified Point 4 can be found in the Appendix. This revised proposition demonstrates that the relaxed

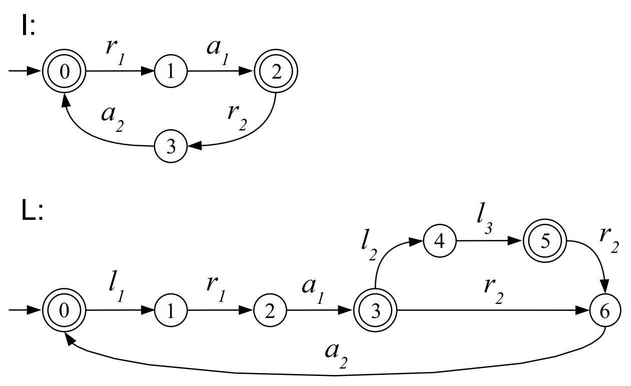

Figure 5.3: Example illustrating the relaxation of Point 4

interface consistency definition holds in the two-level case. Later it will be seen that our interface consistency definition is sufficient for the multiple-level case also.

It should be noted that if this interface-based approach to control is implemented in a distributed fashion, then the new Point 4 will make it difficult for the modules to truly synchronize on a request event, since an event requested by a module does not have to occur in the lower level immediately. If the modular control is implemented on a centralized computer, then the modules can synchronize with one another since there are no problems with communication. If the modular supervisors are distributed across several computers, then when a request is made by a module, the associated lower-level module will have to queue this request until it is able to address it. Even though the modules will not actually be synchronized in time, all the necessary actions will still be performed in the correct order.

Another approach to implementing this modular control in a distributed fashion would be to redesign the component models to include virtual events that could be employed as requests. In this manner, the upper-level module could issue a request that could be enacted by the corresponding lower-level module immediately, such that the lower level is essentially saying that it has received the request. The lower-level module then would go on to carry out the actual task desired by the upper level before issuing an answer. In this approach the original Point 4 could be employed.

5.2.2 Multiple-level serial case

We will now demonstrate results analogous to Theorem 5.5 and Theorem 5.6 for the case where each level of the hierarchy consists of only a single module. This case is referred to as a multiple-level serial-interface architecture. Examination of the proofs for this case will make the logic of the main results of this chapter easier to follow.

Controllability and nonblocking of the multiple-level serial-interface architecture will follow from the results presented for the two-level case. Specifically, the requirements of Theorem 5.10 and Theorem 5.11 must be met for a series of two-level systems consisting of a high level $H^{i-1} = S^{i-1} \| G^{i-1}$, an interface I^i, and a low level $H^i \| I^{i+1} = S^i \| G^i \| I^{i+1}$, where $i = \{2, \ldots, q\}$. The proofs to follow rely on this modified formulation where the low-level plant includes the interface from the level below, that is, the low-level plant is considered $G^i \| I^{i+1}$. The disjointness of the alphabet partitions of equations (5.1) and (5.2) will also be needed. For the interface I^q, H^{q-1} is considered the high-level and H^q is considered the low-level since there is no interface preceding the bottom level of the hierarchy.

Figure 5.4 illustrates the approach taken in the following proofs. The proofs begin with the two-level system at the top of the hierarchy, which is immediately nonblocking and controllable by Theorems 5.10 and 5.11. We then consider this serial system to be the "high-level" and add another module that is considered the "low-level." This process continues where the high-level gets larger and larger and the low-level is just the next module considered. With this in mind, all low-level requirements are immediately met. The high-level properties are shown by induction.

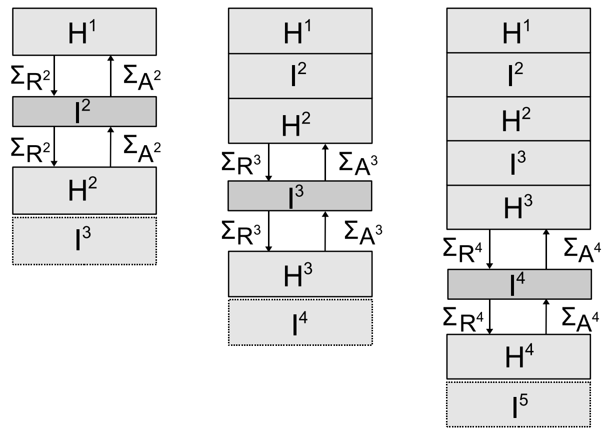

Figure 5.4: Illustration of approach of proofs

The proposition given below will be needed in the proofs to follow. Specifically, this proposition is used to demonstrate controllability between languages that are separated in the hierarchy, that is, languages that do not share relevant events.

Proposition 5.13. *Let $K,\ L \subseteq \Sigma^*$ be prefix-closed languages. If K does not have any relevant events in the set $\Sigma_u \subseteq \Sigma$, then K is Σ_u-controllable with respect to L.*

Proof.

- Let $t \in K\Sigma_u$. This means that $t = s\sigma$ where $s \in K$ and $\sigma \in \Sigma_u$.
- Since all the events in Σ_u are irrelevant to K, $s \in K$ implies $t = s\sigma \in K$ by Definition 2.2. Therefore,

$$K\Sigma_u \subseteq K$$

- Intersecting both sides with L,

$$K\Sigma_u \cap L \subseteq K \cap L$$

- Since $K \cap L \subseteq K$, we therefore have our desired result

$$K\Sigma_u \cap L \subseteq K$$

$\square$

The following two important theorems demonstrate local conditions under which the global multiple-level serial interface system is nonblocking and controllable.

Theorem 5.14. *If the multiple-level serial interface system composed of DES $H^1, H^2, I^2, \ldots, H^q, I^q$, is* multi-level nonblocking *and* multi-level consistent *with respect to the alphabet partitions given by (5.1) and (5.2), then the global system is nonblocking:*

$$\overline{\mathcal{H}^1_m \cap \mathcal{H}^2_m \cap \mathcal{I}^2_m \cap \ldots \cap \mathcal{H}^q_m \cap \mathcal{I}^q_m} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \ldots \cap \mathcal{H}^q \cap \mathcal{I}^q$$

Proof.

• Beginning at the top of the hierarchy, consider a two-level system consisting of a high-level H^1, a low-level $H^2 \| I^3$, and an interface I^2. Because the overall system is multi-level nonblocking, we have for the first level that $\overline{\mathcal{H}^1_m \cap \mathcal{I}^2_m} = \mathcal{H}^1 \cap \mathcal{I}^2$. Similarly for the second level, we have that $\overline{\mathcal{H}^2_m \cap \mathcal{I}^2_m \cap \mathcal{I}^3_m} = \mathcal{H}^2 \cap \mathcal{I}^2 \cap \mathcal{I}^3$. These two results provide that this two-level component is level-wise nonblocking. Additionally, the fact that the overall system is multi-level consistent provides that this two-level component is interface consistent. Therefore, Theorem 5.10 can be applied to show that:

$$\overline{\mathcal{H}^1_m \cap \mathcal{H}^2_m \cap \mathcal{I}^2_m \cap \mathcal{I}^3_m} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \mathcal{I}^3 \tag{5.4}$$

• Now consider a two-level system where the high-level is $H^1 \| H^2 \| I^2$, the low-level is $H^3 \| I^4$, and the interface is I^3. Since the global system is multi-level nonblocking and multi-level consistent, all low-level and multi-level requirements of the level-wise nonblocking and interface consistency definitions are known to be met. The level-wise nonblocking of the high-level has been shown to be met by equation (5.4). The only necessary requirement left to be shown is that Point 3 of the interface consistency definition is satisfied, that is, $\mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2$ is Σ_{A^3}-controllable with respect to the interface $\mathcal{I}^3$.

Since the whole system is multi-level consistent, $\mathcal{H}^2$ is Σ_{A^3}-controllable with respect to the interface $\mathcal{I}^3$. By equations (5.1) and (5.2), the languages $\mathcal{H}^1$ and $\mathcal{I}^2$ do not have any relevant events in the set Σ_{A^3}, therefore, they are both Σ_{A^3}-controllable with respect to $\mathcal{I}^3$ by Proposition 5.13. Hence, the intersection $\mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2$ is also Σ_{A^3}-controllable with respect to $\mathcal{I}^3$ by Proposition 3.4, since the languages are prefix-closed and prefix-closed languages are immediately nonconflicting.

Since this two-level system is therefore level-wise nonblocking and interface consistent, Theorem 5.10 can be employed again to show that:

$$\overline{\mathcal{H}^1_m \cap \mathcal{H}^2_m \cap \mathcal{I}^2_m \cap \mathcal{H}^3_m \cap \mathcal{I}^3_m \cap \mathcal{I}^4_m} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \mathcal{H}^3 \cap \mathcal{I}^3 \cap \mathcal{I}^4$$

• This logic is repeated until all modules have been addressed, leading to the desired result. □

In the above proof, the "low-level" module always stands alone, thus Point 4 of Definition 5.4 is immediately satisfied (as well as all other low-level requirements).

Theorem 5.15. *If the multiple-level serial interface system composed of plant components $G^1, \ldots, G^q$, supervisors $S^1, \ldots, S^q$, and interfaces $I^2, \ldots, I^q$, is* multi-level controllable *with respect to the alphabet partitions given by (5.1) and (5.2), then the supervisor language $\mathcal{S} = \mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2 \cap \ldots \cap \mathcal{S}^q \cap \mathcal{I}^q$ is Σ_u-controllable with respect to the plant language $\mathcal{G} = \mathcal{G}^1 \cap \ldots \cap \mathcal{G}^q$.*

Proof.

• Beginning at the top of the hierarchy, consider a two-level system consisting of a high-level plant G^1 and supervisor S^1, a low-level plant $G^2 \| I^3$ and supervisor S^2, and an interface I^2. Since the overall system is multi-level controllable, we have for the first level that $\mathcal{S}^1$ is Σ_u-controllable with respect to $\mathcal{G}^1 \cap \mathcal{I}^2$. Similarly for the second level, we have that $\mathcal{S}^2 \cap \mathcal{I}^2$ is Σ_u-controllable with respect to $\mathcal{G}^2 \cap \mathcal{I}^3$. These two results provide that this two-level component is level-wise controllable. Therefore, Theorem 5.11 can be applied to show that $\mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2$ is Σ_u-controllable with respect to $\mathcal{G}^1 \cap \mathcal{G}^2 \cap \mathcal{I}^3$.

• Now consider a two-level system with the high-level plant $G^1 \| G^2$ and supervisor $S^1 \| S^2 \| I^2$, a low-level plant $G^3 \| I^4$ and supervisor S^3, and an interface I^3. Since the overall system is multi-level controllable, Points *i)* and *ii)* of the level-wise controllability requirement are satisfied immediately. Point *iii)* is satisfied by the previous step of this proof. Therefore, Theorem 5.11 can be applied again to show that $\mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2 \cap \mathcal{S}^3 \cap \mathcal{I}^3$ is Σ_u-controllable with respect to $\mathcal{G}^1 \cap \mathcal{G}^2 \cap \mathcal{G}^3 \cap \mathcal{I}^4$.

• This logic is repeated until all modules have been addressed, leading to the desired result. □

5.2.3 General multiple-level case

The proofs of the main results of this chapter follow the logic of Theorems 5.14 and 5.15, but with multiple modules per level of the hierarchy. Recall Fig. 5.1 and Fig. 5.2 that illustrate the general multiple-level architecture we are considering.

Proof of Theorem 5.5:

• Beginning at the top of the hierarchy, consider a two-level system consisting of a high-level H_1^1, a set of interfaces $\{I_\ell^2\}$, and a corresponding set of low-level modules $\{H_\ell^2 \| (\|_{k \in J_\ell^2} I_k^3)\}$ where $\ell = \{1, \ldots, n_2\}$. Because the overall system is multi-level nonblocking, we have for the first level that $\overline{\mathcal{H}_{m_1}^1 \cap \bigcap_{\forall \ell} \mathcal{I}_{m_\ell}^2} = \mathcal{H}_1^1 \cap \bigcap_{\forall \ell} \mathcal{I}_\ell^2$. Similarly for each module on the second level, we have that $\overline{\mathcal{H}_{m_\ell}^2 \cap \mathcal{I}_{m_\ell}^2 \cap \bigcap_{k \in J_\ell^2} \mathcal{I}_{m_k}^3} = \mathcal{H}_\ell^1 \cap \mathcal{I}_\ell^2 \cap \bigcap_{k \in J_\ell^2} \mathcal{I}_k^3$. These two results provide that this two-level component is level-wise nonblocking. Additionally, the fact that the overall system is multi-level consistent provides that this two-level component is interface consistent. Therefore, Theorem 5.10 can be applied to show equation (5.5). Within a given level, all modules are included since the system is connected.

$$\overline{\mathcal{H}_m^1 \cap \mathcal{H}_m^2 \cap \mathcal{I}_m^2 \cap \mathcal{I}_m^3} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^3 \cap \mathcal{I}^3 \tag{5.5}$$

• Now consider a system with a high-level $H_1^1 \| H_1^2 \| \ldots \| H_{n_2}^2 \| I_1^2 \| \ldots \| I_{n_2}^2$, a set of interfaces $\{I_k^3\}$, and a corresponding set of low-level modules $\{H_k^3 || (||_{j \in J_k^3} I_j^4)\}$ where $k = \{1, \ldots, n_3\}$. Based on the given assumptions, all low-level and multi-level requirements are known to be met. The level-wise nonblocking of the high-level has been shown to be met by equation (5.5). The only requirement left is Point 3 of the interface consistency definition, that is, it must be shown that the high-level language is $\Sigma_{A_k^3}$-controllable with respect to each $\mathcal{I}_k^3$, $\forall k = \{1, \ldots, n_3\}$.

Consider a single interface language from this two-level system $\mathcal{I}_k^3$. On level 2, there is a single module H_ℓ^2 that shares relevant events with this interface, that is, $(\Sigma_{H_\ell^2} \cap \Sigma_{I_k^3} \neq \emptyset)$. By construction, the language generated by this module $\mathcal{H}_\ell^2$ is $\Sigma_{A_k^3}$-controllable with respect to $\mathcal{I}_k^3$ due to the multi-level consistency requirement. For those modules $H_{\ell'}^2$ from level 2 that do not share relevant events with I_k^3, they do not possess any relevant events that are in the set $\Sigma_{A_k^3}$ by equation (5.2). Therefore by Proposition 5.13, each language $\mathcal{H}_{\ell'}^2$ for which $\ell' \neq \ell$ is also $\Sigma_{A_k^3}$-controllable with respect to $\mathcal{I}_k^3$.

Furthermore, each interface from level 2, I_ℓ^2, and the module from the level 1, H_1^1, also do not share any relevant events with the event set $\Sigma_{A_k^3}$ by equations (5.1) and (5.2). Applying Proposition 5.13 again demonstrates that each of the languages generated by these DES are $\Sigma_{A_k^3}$-controllable with respect to the interface language $\mathcal{I}_k^3$.

Since the module language $\mathcal{H}_1^1$ and the interface and module languages $\mathcal{I}_\ell^2$ and $\mathcal{H}_\ell^2$, where $\ell = \{1, \ldots, n_2\}$, are $\Sigma_{A_k^3}$-controllable with respect to $\mathcal{I}_k^3$, so is the composition of these languages by Proposition 3.4. Otherwise stated, $\mathcal{H}_1^1 \cap \mathcal{H}_1^2 \cap \ldots \cap \mathcal{H}_{n_2}^2 \cap \mathcal{I}_1^2 \cap \ldots \cap \mathcal{I}_{n_2}^2$ is $\Sigma_{A_k^3}$-controllable with respect to the interface language $\mathcal{I}_k^3$. Repeating this logic, this high-level language can be shown to be $\Sigma_{A_k^3}$-controllable with respect to any interface language $\mathcal{I}_k^3$, $\forall k = \{1, \ldots, n_3\}$. Therefore we have shown that Point 3 has been satisfied. Since all level-wise nonblocking and interface consistency requirements are met for this two-level system, Theorem 5.10 then gives us:

$$\overline{\mathcal{H}_m^1 \cap \mathcal{H}_m^2 \cap \mathcal{I}_m^2 \cap \mathcal{H}_m^3 \cap \mathcal{I}_m^3 \cap \mathcal{I}_m^4} = \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2 \cap \mathcal{H}^3 \cap \mathcal{I}^3 \cap \mathcal{I}^4$$

• This logic is repeated until all modules on all q levels have been addressed. Low-level modules that do not have any interfaces below them are slightly different in that each module just has the form H_k^i. However, they still satisfy the level-wise nonblocking and interface consistency requirements leading to the desired result. $\square$

Proof of Theorem 5.6:

• Beginning at the top of the hierarchy, consider a two-level interface system consisting of a high-level plant G_1^1 and supervisor S_1^1, a set of interfaces $\{I_\ell^2\}$, and a corresponding set of low-level plants $\{G_\ell^2 || (||_{k \in J_\ell^2} I_k^3)\}$ and supervisors $\{S_\ell^2\}$ where $\ell = \{1, \ldots, n_2\}$. Since the overall system is multi-level controllable, we have for the first level that $\mathcal{S}_1^1$ is Σ_u-controllable with respect to $\mathcal{G}_1^1 \cap \bigcap_{\forall \ell} \mathcal{I}_\ell^2$. Similarly for the second level, we have that each $\mathcal{S}_\ell^2 \cap \mathcal{I}_\ell^2$ is Σ_u-controllable with respect $\mathcal{G}_\ell^2 \cap \bigcap_{k \in J_\ell^2} \mathcal{I}_k^3$. These two results provide that this two-level component is level-wise controllable. Therefore, Theorem 5.11 can be applied to show that the language $\mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2$ is Σ_u-controllable with respect to $\mathcal{G}^1 \cap \mathcal{G}^2 \cap \mathcal{I}^3$. Within a given level, all modules are included since the system is connected.

• Now consider a interface system with a high-level plant $G_1^1 \| G_1^2 \| \ldots \| G_{n_2}^2$ and supervisor $S_1^1 \| S_1^2 \| \ldots \| S_{n_2}^2 \| I_1^2 \| \ldots \| I_{n_2}^2$, a set of interfaces $\{I_k^3\}$, and a corresponding set of low-level plants $\{G_k^3 || (||_{j \in J_k^3} I_j^4)\}$ and supervisors $\{S_k^3\}$ where $k = \{1, \ldots, n_3\}$. Since the overall system is multi-level controllable, Points *i*) and *ii*) of the level-wise controllability requirement are satisfied. Additionally, Point *iii*) is known to be satisfied based on the previous step of this proof. Therefore, Theorem 5.11 can be applied again to show that the language $\mathcal{S}^1 \cap \mathcal{S}^2 \cap \mathcal{I}^2 \cap \mathcal{S}^3 \cap \mathcal{I}^3$ is Σ_u-controllable with respect to $\mathcal{G}^1 \cap \mathcal{G}^2 \cap \mathcal{G}^3 \cap \mathcal{I}^4$.

• This logic is repeated until all modules on all q levels have been addressed. Lower-level modules that do not have any interfaces below them are slightly different in that their plant components just have the form G_k^i. However, they still satisfy level-wise controllability leading to the desired result. $\square$

5.3 Supervisor Synthesis

In the previous section we demonstrated that for a given multiple-level interface system the local properties of Section 5.1 are sufficient for guaranteeing the global properties of nonblocking and controllability. It was not, however, specified how to construct the multiple-level interface system. In this section we will outline a systematic approach for constructing the component supervisors for a multiple-level hierarchical interface-based architecture that are guaranteed to meet the necessary requirements by construction. The approach presented here follows the work of [39] that provides results on how to construct high and low-level supervisors that are optimal with respect to a given specification and set of interfaces in the two-level case. The extension to the multiple-level case specifically requires that a modular supervisor be synthesized to meet low and high-level requirements simultaneously. The details of this extension logically follows the results of [39].

The basic approach of [39] is similar to the traditional approach for constructing a supremal controllable and nonblocking sublanguage [78]. This involves first the construction of the language that represents the portion of the system's uncontrolled behavior that is allowed by the given specification. This language is then pruned to remove those strings that violate controllability or nonblocking. This construction is in general performed on the automaton generator of the language.

For the multiple-level interface-based architecture of this chapter, we will construct a supervisor S_k^i with respect to a component plant G_k^i, specification E_k^i, and set of interfaces I_k^i, $\{I_j^{i+1}\}$ where j represents all those indices in the set J_k^i for the given module. The starting point for the synthesis of the supervisor for the i-k^{th} module will then be the automaton $Z_k^i = G_k^i \| E_k^i \| I_k^i \| (\|_{j \in J_k^i} I_j^{i+1})$. The generated and marked languages for this automaton lifted to the global alphabet Σ are then:

$$\begin{aligned}
\mathcal{Z}_k^i &= P_{H_k^i}^{-1}(\mathcal{L}(Z_k^i)) = \mathcal{G}_k^i \cap \mathcal{E}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \\
\mathcal{Z}_{m_k}^i &= P_{H_k^i}^{-1}(\mathcal{L}_m(Z_k^i)) = \mathcal{G}_{m_k}^i \cap \mathcal{E}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}
\end{aligned} \tag{5.6}$$

The synthesis of the modular supervisor languages then requires the removal of those strings that violate any of the properties required of the multiple-level interface-based approach to control. Any continuations of these removed strings are also removed in order to generate a prefix-closed language. These requirements are captured in the following definition where Ψ represents a multiple-level specification interface system similar to Φ. Here Ψ differs from Φ in that it includes modular specifications in place of modular supervisors since the supervisors have not been synthesized yet.

Definition 5.16. The multiple-level specification interface system Ψ is said to be *multi-level valid* with respect to the alphabet partitions given by (5.1) and (5.2), if for all $i \in \{1, \ldots, q\}$ and for all $k \in \{1, \ldots, n_i\}$ corresponding to each i, the following conditions are satisfied:

1) The event set of G_k^i and E_k^i is $\Sigma_{H_k^i}$.
2) I_k^i is a command-pair interface. $\diamond$

In the subsequent results of this section, it will be assumed that the multiple-level specification interface system Ψ is multi-level valid with respect to the alphabet partitions given by equations (5.1) and (5.2). A language satisfying the remaining conditions of the definitions of Section 5.1 is then said to be *i-k*th *multi-level interface controllable* (MIC_k^i). This specific term is defined below:

Definition 5.17. Let $\mathcal{Z} \subseteq \Sigma^*$. For system Ψ, the language $\mathcal{Z}$ is *i-k*th *multi-level interface controllable* (MIC_k^i) if for all $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$, the following conditions are satisfied:

1) $\text{Elig}_{\mathcal{G}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \text{Elig}_{\overline{\mathcal{Z}} \cap \mathcal{I}_k^i}(s)$
2) $\text{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \subseteq \text{Elig}_{\mathcal{G}_k^i \cap \overline{\mathcal{Z}}}(s), \ \forall j \in J_k^i$
3) $(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})$,
   $s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*) \ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$
4) $(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i}) \ s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)$,
   $s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$
5) $s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*) \ sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z}$ $\diamond$

Point 1 of the above definition corresponds to the multi-level controllability requirement, while Points 2-5 correspond to Points 3-6 of the multi-level consistency requirement.

To formally present this approach to supervisor synthesis, we now define for an arbitrary language $\mathcal{E} \subseteq \Sigma^*$ a class of sublanguages of $\mathcal{E}$ that are *i-k*th multi-level interface controllable for the given multiple-level specification interface system Ψ:

$$\mathcal{C}_{M_k^i}(\mathcal{E}) := \{\mathcal{Z} \subseteq \mathcal{E} \mid \mathcal{Z} \text{ is } MIC_k^i \text{ with respect to } \Psi\}$$

The following proposition then demonstrates that the set $\mathcal{C}_{M_k^i}(\mathcal{E})$ is nonempty and closed under union. This therefore implies that a unique supremal element exists for this set.

Proposition 5.18. *Let $\mathcal{E} \subseteq \Sigma^*$. For system Ψ, $\mathcal{C}_{M_k^i}(\mathcal{E})$ is nonempty and closed under arbitrary union. In particular, $\mathcal{C}_{M_k^i}(\mathcal{E})$ contains a (unique) supremal element that we will denote* $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$.

Proof. See proof in Appendix. $\square$

With this result, we can then employ the sublanguage $\sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$ to define the supervisor language for the *i*-k^{th} module. Specifically, we can let $\mathcal{S}_{m_k}^i = \sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$ and $\mathcal{S}_k^i = \overline{\mathcal{S}_{m_k}^i}$. This construction and equation (5.6) provide that $\mathcal{S}_{m_k}^i \subseteq \mathcal{Z}_{m_k}^i \subseteq \mathcal{G}_{m_k}^i$ and $\mathcal{S}_k^i \subseteq \overline{\mathcal{Z}_{m_k}^i} \subseteq \overline{\mathcal{G}_{m_k}^i} = \mathcal{G}_k^i$. The resulting supervised behavior is, therefore, nonblocking since $\mathcal{H}_k^i = \mathcal{S}_k^i \cap \mathcal{G}_k^i = \mathcal{S}_k^i$ and $\mathcal{H}_{m_k}^i = \mathcal{H}_k^i \cap \mathcal{S}_{m_k}^i \cap \mathcal{G}_{m_k}^i = \mathcal{S}_{m_k}^i$. This construction leads to a marking supervisor in that the supervisor can in essence unmark strings that are marked in the plant language $\mathcal{G}_{m_k}^i$. This arises due to the inclusion of the marked specification and interface languages in the construction of $\mathcal{Z}_{m_k}^i$ as shown in equation (5.6).

Now that we have established that a supervisor exists that is maximally permissive with respect to a given specification and set of interfaces, we would now like to demonstrate that this language can be constructed. With this in mind, we will define the operator $\Omega_{M_k^i}$ and show that its fixpoint is $\sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$. The language fixpoint operator $\Omega_{M_k^i}$ will be defined in terms of two intermediate operators $\Omega_{MNB_k^i}$ and $\Omega_{MIC_k^i}$ that we will define first. The operator $\Omega_{MNB_k^i}$ specifically returns those strings of a given prefix-closed language that are marked in $\mathcal{Z}_{m_k}^i$.

Definition 5.19. For system Ψ, we define the *nonblocking operator* $\Omega_{MNB_k^i} : \Sigma^* \rightarrow \Sigma^*$, for arbitrary $\mathcal{Z} \subseteq \Sigma^*$ as follows:

$$\Omega_{MNB_k^i}(\mathcal{Z}) := \mathcal{Z} \cap \mathcal{Z}_{m_k}^i \quad \diamond$$

The next operator $\Omega_{MIC_k^i}$ removes from a given prefix-closed language those strings that fail any of the elements of the *i*-k^{th} multi-level interface controllability definition. Continuations of the failed strings are also removed to maintain prefix-closure, as indicated by the $\mathrm{Ext}_{\overline{\mathcal{Z}}}$ operator, that returns the continuations in $\overline{\mathcal{Z}}$ of a given set of strings.

Definition 5.20. For system Ψ, we define the *interface controllable operator* $\Omega_{MIC_k^i} : \Sigma^* \rightarrow \Sigma^*$, for arbitrary $\mathcal{Z} \subseteq \Sigma^*$ as follows:

$$\Omega_{MIC_k^i}(\mathcal{Z}) := \overline{\mathcal{Z}} - \mathrm{Ext}_{\overline{\mathcal{Z}}}(\mathrm{FailIC}_k^i(\overline{\mathcal{Z}}))$$

where

$$\begin{aligned}
\mathrm{FailIC}_k^i(\overline{\mathcal{Z}}) \;:=\; & \{s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}} \mid \neg[\mathrm{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\overline{\mathcal{Z}} \cap \mathcal{I}_k^i}(s)] \\
\vee \;\; & [\exists j \in J_k^i \mid \neg(\mathrm{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_k^i} \subseteq \mathrm{Elig}_{\mathcal{G}_k^i \cap \overline{\mathcal{Z}}}(s))] \\
\vee \;\; & \neg[(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i}) \\
& s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*) \; sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}] \\
\vee \;\; & \neg[(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i}) \\
& s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*) \; s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}] \\
\vee \;\; & \neg[s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*) \; sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z}]\} \;\; \diamond
\end{aligned}$$

We can now define our fixpoint operator $\Omega_{M_k^i}$.

Definition 5.21. For system Ψ, we define the *i-k*th fixpoint operator, $\Omega_{M_k^i} : \Sigma^* \to \Sigma^*$, for arbitrary $\mathcal{Z} \subseteq \Sigma^*$ as follows:

$$\Omega_{M_k^i} := \Omega_{MNB_k^i}(\Omega_{MIC_k^i}(\mathcal{Z})) \;\; \diamond$$

The following important result demonstrates that if $\Omega_{M_k^i}(\mathcal{Z}_k^i)$ reaches a fixpoint in a finite number of steps, then the fixpoint is equal to $\sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$.

Theorem 5.22. *For system Ψ, if there exists $j \in \{0, 1, 2, \ldots\}$ such that $\Omega_{M_k^i}^j(\mathcal{Z}_k^i)$ is a fixpoint, then $\Omega_{M_k^i}^j(\mathcal{Z}_k^i) = \sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$.*

Proof. Proof of this result is identical to the proof of Theorem 8 in [7], after relabeling. □

Finally, the following demonstrates that if the resulting supremal element is employed as our supervisor language, then the necessary interface-based requirements of Section 5.1 are satisfied.

Corollary 5.23. *For system Ψ, if there exists $j \in \{0, 1, 2, \ldots\}$ such that $\Omega_{M_k^i}^j(\mathcal{Z}_k^i)$ is a fixpoint, then the system Φ with $\mathcal{S}_{m_k}^i = \Omega_{M_k^i}^j(\mathcal{Z}_k^i)$ and $\mathcal{S}_k^i = \overline{\mathcal{S}_{m_k}^i}$ satisfies Points 3, 4, 5, and 6 of the multi-level consistency definition, Point ii) of the multi-level controllability definition, and the multi-level nonblocking definition.*

Proof. See proof in Appendix. □

The fixpoint operators that have been presented so far are language-based. The specific supervisor synthesis algorithms presented in [7] for the two-level case are, however, automata-based. While we will not present a specific algorithm for automata-based supervisor construction, like [7] we can show an equivalence between removing strings from a language and removing states from an automaton.

The basic approach to showing the equivalence between a language-based algorithm and an automata-based algorithm is to show that if a string that reaches a state q fails to meet some necessary condition, then all strings that reach this state q will also fail to meet the necessary condition. This way, removing a state from an automaton only removes strings that violate the given requirement. The following proposition from [7] addresses the property of blocking, that is, those strings that cannot be extended to a marked string.

Proposition 5.24. *[7] Let $G = (Q, \Sigma, \delta, q_0, Q_m)$. It thus follows that for all $s, t \in \mathcal{L}(G)$, if $\delta(q_0, s) = \delta(q_0, t)$ then*

$$s \notin \overline{\mathcal{L}_m(G)} \Leftrightarrow t \notin \overline{\mathcal{L}_m(G)}$$

The following proposition similarly addresses the requirements of the *i-k^{th}* multi-level interface controllability definition. In the statement of the proposition, we will employ the automaton $H_k^{i\prime}$ which is equal to the automaton $G_k^i \| I_k^i \| (\big|\big|_{j \in J_k^i} I_j^{i+1}) \| S_k^i$, with self-loops added for all events in the set $\Sigma - \Sigma_{H_k^i}$. Therefore, $H_k^{i\prime}$ has an event set of Σ and $\mathcal{L}(H_k^{i\prime}) = \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{S}_k^i$.

Proposition 5.25. *For the system Φ, Let $H_k^{i\prime} = (Q_k^i, \Sigma, \delta_k^i, q_{0_k}^i, Q_{m_k}^i)$. It thus follows that for all $s, t \in \mathcal{L}(H_k^{i\prime})$, if $\delta_k^i(q_{0_k}^i, s) = \delta_k^i(q_{0_k}^i, t)$ then*

1) $\text{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \nsubseteq \text{Elig}_{\mathcal{S}_k^i \cap \mathcal{I}_k^i}(s) \Leftrightarrow \text{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(t) \cap \Sigma_u \nsubseteq \text{Elig}_{\mathcal{S}_k^i \cap \mathcal{I}_k^i}(t)$
2) $\text{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \nsubseteq \text{Elig}_{\mathcal{H}_k^i}(s) \Leftrightarrow \text{Elig}_{\mathcal{I}_j^{i+1}}(t) \cap \Sigma_{A_j^{i+1}} \nsubseteq \text{Elig}_{\mathcal{H}_k^i}(t), \ \forall j \in J_k^i$
3) $(\forall s, t \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i}) \ [s\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)$
   $sl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Leftrightarrow [t\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*) \ tl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$
4) $(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i}) \ [s\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*) \ s\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Leftrightarrow$
   $[t\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*) \ t\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$
5) $[s \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*) \ sl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}] \Leftrightarrow$
   $[t \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*) \ tl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}]$

Proof. See proof in Appendix. □

With the above two propositions, it becomes apparent that an application of the language-based fixpoint operator $\Omega_{M_k^i}$ is equivalent to removing at least one state from the automaton $H_k^{i\prime}$, or results in a fixpoint. The removal of a state consequently removes strings with continuations from that state from the language $\mathcal{L}(H_k^{i\prime})$. This is consistent with the definition of the operator $\Omega_{M_k^i}$. Assuming that we have regular languages, our automata generators have a finite number of states. Therefore, even a naive algorithm that tests each state of an automaton one at a time, and removes states that are not coreachable or that are reached by strings that violate i-k^{th} multi-level interface controllability, will reach a fixpoint in finite time.

In this work we will not provide an algorithm for synthesizing the component supervisors. We believe, however, that an algorithm can be developed that has polynomial complexity in the number of states and events of a given module with its interfaces. Verifying nonblocking and each of the points of the i-k^{th} multi-level interface controllability definition equates to comparing the feasible event sets of individual automata (controllability tests) and performing reachability searches, to test whether a marked state or a given request event or answer event can be reached. In order to perform these tests, the transition structures of the automaton $H_k^{i\prime}$ and the individual automata from which $H_k^{i\prime}$ is composed must be stored. Reachability information and feasible event sets of the individual components are stored in the transition structures of their automata, while the automaton $H_k^{i\prime}$ provides a mapping between states of the components that are reached by the same given string. Efficient state-based implementation of supervisor synthesis has already been developed in the two-level case [7]. This could provide a good starting point for developing a supervisor synthesis algorithm in the multiple-level case.

5.4 Interface Synthesis

In the previous section we outlined an approach for synthesizing supervisors for each of the modules in a multiple-level architecture. An area that has not yet been addressed in the literature is how to synthesize the interfaces. Traditionally, these interfaces have been arrived at based on designer understanding of the system. The problem that arises here is that a poor choice of interface could make it impossible to meet the necessary requirements; that is, a resulting modular supervisor language could be the empty set. Even if nonempty modular supervisor languages exist, the chosen interface languages could be overly restrictive. We would like to propose here one possible methodology for constructing an interface language $\mathcal{I}_k^i$ in conjunction with the supervisor language for the i-k^{th} module.

The idea behind the approach suggested here is that each interface be viewed as an abstraction of its associated controlled module. This point of view is consistent with the idea that the interfaces serve to hide information between levels of the hierarchy. This notion of employing abstraction has recently been a common theme in the field of supervisory controller design. Existing works, however, place strict requirements on their abstractions. Here we simply propose to employ the natural projection operation. We propose to build off the supervisor synthesis approach of the Section 5.3. Specifically, the idea is that the interface language is allowed to change as the MIC_k^i sublanguage is constructed. This approach can help prevent the resulting control from being overly restrictive. While this approach will provide an explicit methodology for constructing interfaces, the overall MLIBC approach would still rely on heuristics in the choice of the system partition and in the choice of request and answer events.

Based on the preceding discussion, the interface languages will be defined as in equation (5.7) for a given language $\mathcal{Z} \subseteq \mathcal{Z}_{m_k}^i$ and a given set of request and answer events. Here the language $\mathcal{Z}$ is meant to represent an intermediate sublanguage of $\mathcal{Z}_{m_k}^i$ as strings are removed to make the language i-k^{th} multi-level interface controllable. While the following definitions and proofs are presented in terms of languages, the result that removing strings from a language is equivalent to removing states from the automaton generator is still valid.

$$\begin{aligned} \mathcal{L}(I_k^i) &= P_{I_k^i}(\overline{\mathcal{Z}}) \\ \mathcal{L}_m(I_k^i) &= (\Sigma_{R_k^i}.\Sigma_{A_k^i})^* \cap \mathcal{L}(I_k^i) \end{aligned} \tag{5.7}$$

A couple of things to notice about the above construction of the interfaces is that the projection operation does not guarantee that the interface automaton I_k^i will have fewer states than the automaton which generates $\mathcal{Z}$. However, in many cases I_k^i will be smaller. Requiring that the projection additionally be an *observer* will guarantee that the projected language is smaller than the original language [72]. We will not, however, require the observer property.

The construction of equation (5.7) also does not guarantee that I_k^i will have the form of a command-pair interface. As such, it is necessary to modify the definition of Z^i_k from its form in equation (5.6). The basic idea is to add another "specification" $E^i_{alt_k}$ which imposes the requirement that request and answer events alternate, that is, $\mathcal{L}_m(E^i_{alt_k}) = (\Sigma_{R^i_k}.\Sigma_{A^i_k})^*$. In terms of the lifted language, $\mathcal{E}^i_{alt_k} = P^{-1}_{I^i_k}(\mathcal{L}(E^i_{alt_k}))$. Since we are constructing the interface language $\mathcal{I}^i_k$ at the same time as we are constructing the associated supervisor language, we will define $Z^{i\prime}_k = G^i_k \| E^i_k \| E^i_{alt_k} \| (\big\|_{j \in J^i_k} I^{i+1}_j)$. It then follows that:

$$\begin{aligned}
\mathcal{Z}^{i\prime}_k &= P^{-1}_{H^i_k}(\mathcal{L}(Z^{i\prime}_k)) = \mathcal{G}^i_k \cap \mathcal{E}^i_k \cap \mathcal{E}^i_{alt_k} \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j \\
\mathcal{Z}^{i\prime}_{m_k} &= P^{-1}_{H^i_k}(\mathcal{L}_m(Z^{i\prime}_k)) = \mathcal{G}^i_{m_k} \cap \mathcal{E}^i_{m_k} \cap \mathcal{E}^i_{alt_{m_k}} \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_{m_j} \qquad (5.8)
\end{aligned}$$

In a sense, the above definitions provide that $\mathcal{E}^i_{alt_k}$ is our first guess at the interface language $\mathcal{I}^i_k$. As $\mathcal{Z}^{i\prime}_k$ is modified to generate an MIC^i_k sublanguage, so too will $\mathcal{I}^i_k$ be altered. The simplest approach that could be applied here would be to employ the same supervisor synthesis algorithm that would be employed if the interface was predetermined. The difference is that any time a string was removed from $\mathcal{Z}^{i\prime}_k$, or a state was removed from $Z^{i\prime}_k$, the interface I^i_k would be recalculated based on equation (5.7).

Even though in this modified approach I^i_k is allowed to change, it would still remain a command-pair interface. This fact is demonstrated by the following result.

Proposition 5.26. *If the reduced interface DES I^i_k satisfies the relations of equation (5.7) where $\mathcal{Z} \subseteq \mathcal{Z}^{i\prime}_{m_k}$, then I^i_k is a command-pair interface.*

Proof. Point A of Definition 5.1 can be demonstrated for the reduced I^i_k by the following logic. First note the definition of I^i_k in equation (5.7) and the fact that it is given that $\mathcal{Z} \subseteq \mathcal{Z}^{i\prime}_{m_k}$.

$$\mathcal{L}(I^i_k) = P_{I^i_k}(\overline{\mathcal{Z}}) \subseteq P_{I^i_k}(\overline{\mathcal{Z}^{i\prime}_{m_k}}) \qquad (5.9)$$

Now noting the definition of $\mathcal{Z}^{i\prime}_{m_k}$ given in equation (5.8), we have that:

$$P_{I^i_k}(\overline{\mathcal{Z}^{i\prime}_{m_k}}) \subseteq P_{I^i_k}(\overline{\mathcal{E}^i_{alt_{m_k}}}) = \overline{(\Sigma_{R^i_k}.\Sigma_{A^i_k})^*} \qquad (5.10)$$

Combining equations (5.9) and (5.10) provides the desired satisfaction of Point A, $\mathcal{L}(I^i_k) \subseteq \overline{(\Sigma_{R^i_k}.\Sigma_{A^i_k})^*}$. Point B of Definition 5.1 is satisfied by the construction of I^i_k given in equation (5.7), $\mathcal{L}_m(I^i_k) = (\Sigma_{R^i_k}.\Sigma_{A^i_k})^* \cap \mathcal{L}(I^i_k)$. $\square$

Furthermore, even though the interface I_k^i is changing, it will not change the current form of the intermediate language $\mathcal{Z}$ being constructed in the supervisor synthesis algorithm. The idea is that any prior analysis that led to $\mathcal{Z}$ will not be invalidated by changing to a reduced interface. More specifically, if a string s satisfies all five points of Definition 5.17 for an interface language $\mathcal{I}_k^i$, then s will still satisfy the definition for a reduced interface language $\mathcal{I}_k^{i\prime} \subseteq \mathcal{I}_k^i$ as long as the relations of equation (5.7) still hold. This fact is demonstrated by the following proposition.

Proposition 5.27. *Let $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$ satisfy all five points of Definition 5.17 with respect to the interface language $\mathcal{I}_k^i$. If the reduced interface language $\mathcal{I}_k^{i\prime} \subseteq \mathcal{I}_k^i$ satisfies the relations of equation (5.7) where $\mathcal{Z} \subseteq \mathcal{Z}_{m_k}^{i\prime}$, then all five points of Definition 5.17 are still satisfied by s with respect to the new interface language $\mathcal{I}_k^{i\prime}$.*

Proof. First note that it is given that $I_k^{i\prime}$ satisfies the relations of equation (5.7), therefore:

$$\begin{aligned} \mathcal{L}(I_k^{i\prime}) &= P_{I_k^i}(\overline{\mathcal{Z}}) \\ \mathcal{L}_m(I_k^{i\prime}) &= (\Sigma_{R_k^i}.\Sigma_{A_k^i})^* \cap \mathcal{L}(I_k^{i\prime}) \end{aligned} \tag{5.11}$$

By equation (5.11) we have that $\mathcal{I}_k^{i\prime} = P_{I_k^i}^{-1}(P_{I_k^i}(\overline{\mathcal{Z}}))$. Therefore, $\overline{\mathcal{Z}} \subseteq \mathcal{I}_k^{i\prime} \subseteq \mathcal{I}_k^i$ and:

$$\overline{\mathcal{Z}} \cap \mathcal{I}_k^i = \overline{\mathcal{Z}} = \overline{\mathcal{Z}} \cap \mathcal{I}_k^{i\prime} \tag{5.12}$$

Based on equation (5.12), the fact that $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$ means that $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^{i\prime} \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$ also. With these facts in place, we can now demonstrate that each point of Definition 5.17 is still satisfied by s for this reduced interface language $\mathcal{I}_k^{i\prime}$.

1. It is given that Point 1 is satisfied for the original interface language $\mathcal{I}_k^i$, therefore:

   $$\mathrm{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\overline{\mathcal{Z}} \cap \mathcal{I}_k^i}(s)$$

   Based on equation (5.12) we then have that Point 1 is still satisfied for the reduced interface:

   $$\mathrm{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\overline{\mathcal{Z}} \cap \mathcal{I}_k^{i\prime}}(s)$$

2. Since Point 2 is satisfied for the original interface $\mathcal{I}_k^i$, it is automatically satisfied for the reduced interface $\mathcal{I}_k^{i\prime}$ since Point 2 does not depend on $\mathcal{I}_k^i$.

3. Point 3 is also satisfied for the original interface language $\mathcal{I}_k^i$:

$$\begin{aligned}&(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ s\rho \in \mathcal{I}_k^i \\ &\Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}\end{aligned} \tag{5.13}$$

Since $\mathcal{I}_k^{i\prime} \subseteq \mathcal{I}_k^i$, $s\rho \in \mathcal{I}_k^{i\prime}$ implies that $s\rho \in \mathcal{I}_k^i$. Therefore by equation (5.13), $s\rho \in \mathcal{I}_k^{i\prime}$ provides that $(\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$. Furthermore, using equation (5.12) again we have that Point 3 is also satisfied for $\mathcal{I}_k^{i\prime}$:

$$\begin{aligned}&(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ s\rho \in \mathcal{I}_k^{i\prime} \\ &\Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^{i\prime} \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}\end{aligned}$$

4. Recalling that Point 4 is satisfied for $\mathcal{I}_k^i$:

$$\begin{aligned}&(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i})\ s\rho\alpha \in \mathcal{I}_k^i \\ &\Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}\end{aligned} \tag{5.14}$$

Since $\mathcal{I}_k^{i\prime} \subseteq \mathcal{I}_k^i$, $s\rho\alpha \in \mathcal{I}_k^{i\prime}$ implies that $s\rho\alpha \in \mathcal{I}_k^i$. Therefore by equation (5.14), $s\rho\alpha \in \mathcal{I}_k^{i\prime}$ provides that $(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$. Furthermore, using equation (5.12) again we have that Point 4 is also satisfied for $\mathcal{I}_k^{i\prime}$:

$$(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i})\ s\rho\alpha \in \mathcal{I}_k^{i\prime} \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^{i\prime} \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$$

5. It is further given that Point 5 is satisfied for $\mathcal{I}_k^i$:

$$s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z}$$

Since each successive interface satisfies a version of equation (5.11), $\mathcal{L}_m(I_k^i) = (\Sigma_{R_k^i}.\Sigma_{A_k^i})^* \cap \mathcal{L}(I_k^i)$ and $\mathcal{L}_m(I_k^{i\prime}) = (\Sigma_{R_k^i}.\Sigma_{A_k^i})^* \cap \mathcal{L}(I_k^{i\prime})$. Therefore the fact that $\mathcal{I}_k^{i\prime} \subseteq \mathcal{I}_k^i$ implies that $\mathcal{I}_{m_k}^{i\prime} \subseteq \mathcal{I}_{m_k}^i$ also. Hence, $s \in \mathcal{I}_{m_k}^{i\prime}$ implies $s \in \mathcal{I}_{m_k}^i$ which in turn provides that $(\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{Z}$. Furthermore since it is given that $\mathcal{Z} \subseteq \mathcal{Z}^{i\prime}_{m_k}$, equation (5.8) provides that $\mathcal{Z} \subseteq \mathcal{E}^{i}_{alt_{m_k}}$. Therefore by equation (5.11) and the fact that $\overline{\mathcal{Z}} \subseteq \mathcal{I}^{i\prime}_{k}$,

$$\mathcal{Z} = \mathcal{Z} \cap \overline{\mathcal{Z}} \subseteq \mathcal{E}^{i}_{alt_{m_k}} \cap \mathcal{I}^{i\prime}_{k} = \mathcal{I}^{i\prime}_{m_k} \tag{5.15}$$

Now since $\mathcal{Z} \subseteq \mathcal{I}^{i\prime}_{m_k}$ and $\mathcal{I}^{i\prime}_{m_k} \subseteq \mathcal{I}^{i}_{m_k}$, we have that $\mathcal{Z} \cap \mathcal{I}^{i}_{m_k} = \mathcal{Z} \cap \mathcal{I}^{i\prime}_{m_k} \cap \mathcal{I}^{i}_{m_k} = \mathcal{Z} \cap \mathcal{I}^{i\prime}_{m_k}$. Hence, Point 5 still holds for the reduced interface language:

$$s \in \mathcal{I}^{i\prime}_{m_k} \Rightarrow (\exists l \in \Sigma^*_{L^i_k})\ sl \in \mathcal{G}^{i}_{m_k} \cap \mathcal{I}^{i\prime}_{m_k} \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_{m_j} \cap \mathcal{Z}$$

□

With the above two propositions, we have demonstrated that we can employ the following naive algorithm. Begin with the automaton $Z^{i\prime}_k$ and construct an interface language according to equation (5.7) where $\mathcal{Z} = P^{-1}_{H^i_k}(\mathcal{L}(Z^{i\prime}_k))$. Commence to construct a supervisor language in the manner proposed by the previous section, that is, remove states from $Z^{i\prime}_k$ if they are reached by strings that fail any of the points of Definition 5.17 or if they are not coreachable. Every time a state is removed, recalculate the interface language again according to equation (5.7). This process is repeated until the resulting automaton is nonblocking and generates a language that is *i*-*k*th multi-level interface controllable or there are no states remaining. The resulting interface I^i_k is then the last one constructed by equation (5.7).

A drawback of this new approach is that it requires the additional computation of performing a natural projection to generate each new version of the interface. In the worst case, the computation of a projection can have exponential complexity and the minimal automaton that generates the projected language can actually be larger than the minimal automaton that generates the original language. In many instances, however, this is not the case. Furthermore, it is the goal of this approach to keep the modules small enough that the complexity of the projection operation will not be prohibitive. Additionally, we believe that the ultimate resulting interface can be constructed incrementally as the supervisor is synthesized, rather than performing a full natural projection each time a state is removed from $Z^{i\prime}_k$.

Another limitation of this approach to interface construction is that the resulting interface is not unique and depends on the order in which states are removed from $Z^{i\prime}_k$. Also, there is no interface for which the resulting language is supremal. This, however, is more of a drawback to the overall interface-based approach to control, rather than a limitation of this particular approach to interface synthesis. These elements of interface construction are illustrated by the following example.

Example 5.28. Consider the system Z shown in Fig. 5.5 where $\Sigma_R = \{r_1, r_2\}$ and $\Sigma_A = \{a_1, a_2\}$. For this given system, the interface I constructed by equation (5.7) is shown on the left of Fig. 5.6. For this interface, the language $\mathcal{L}(Z)$ fails to be multi-level interface controllable since those strings that reach state 1 cannot be extended to the request event r_2 by low-level events, that is, Point 3 of Definition 5.17 is violated. The same is also true for those strings that reach state 2 in that they cannot be extended to the request event r_1 by low-level events. Removing state 1 and subsequently the unreachable state 3, results in a DES that is multi-level interface controllable with respect to the interface I_1 shown in Fig. 5.6 and generated by equation (5.7). If rather state 2 and state 4 are removed, then the resulting DES is multi-level interface controllable with respect to the interface I_2 constructed according to equation (5.7) and shown in Fig. 5.6. Therefore, depending on whether state 1 or state 2 was removed first, the resulting interface is different. Additionally, the languages generated by the two interfaces are incomparable.

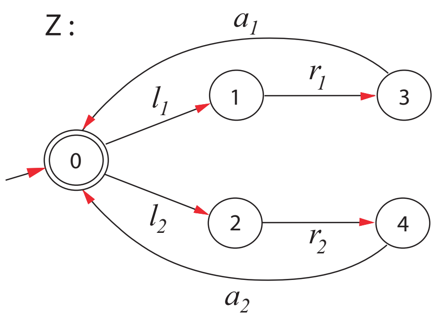

Figure 5.5: Example system for interface synthesis

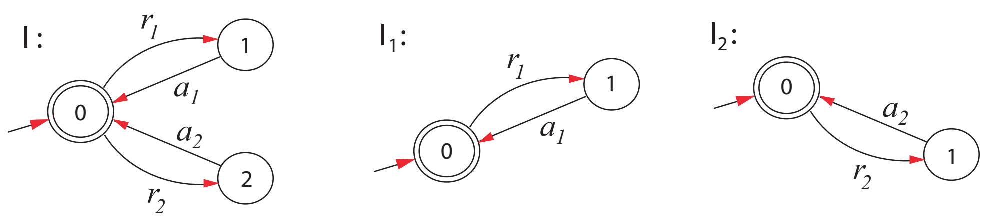

Figure 5.6: Example interfaces for system in Fig. 5.5

5.5 Implementation Examples and Discussion

In this section we will demonstrate an application of the MLIBC approach to an extension of the FMS example first introduced in Chapter 1. This expanded example is shown in Fig. 5.7 and is employed in order to provide a resulting architecture with multiple levels. In this version of the FMS, machines *Robot2* and *Mill*, and buffers *B3* and *B9* are new. These new components and a modified AM are pictured in Fig. 5.8. In the process of this application, we will provide some heuristics for making some of the necessary design choices. At the end of this section, we will also discuss the complexity of this approach, in particular, its scalability as compared to the original two-level architecture.

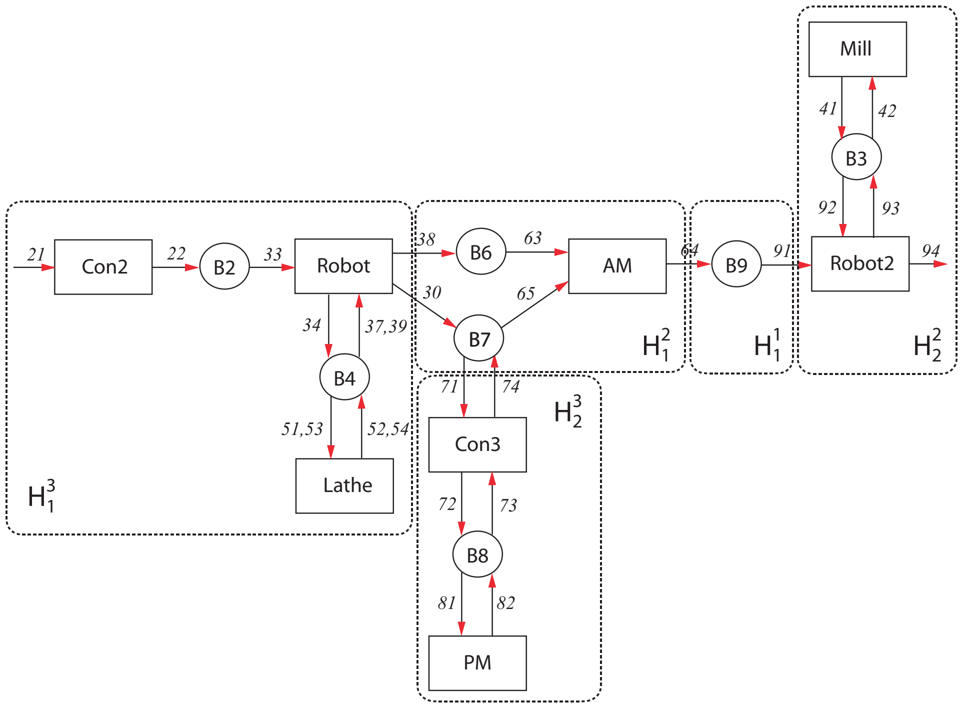

Figure 5.7: Extended FMS example

5.5.1 Flexible Manufacturing System (FMS) example

Application of the MLIBC approach depends in part on designer understanding. Specifically, how the system components are partitioned into modules, how request and answer events are chosen, and how interfaces are constructed are all areas where designer intuition could enter in. In this section we will present a procedure for implementing the MLIBC approach where we provide some heuristics for making

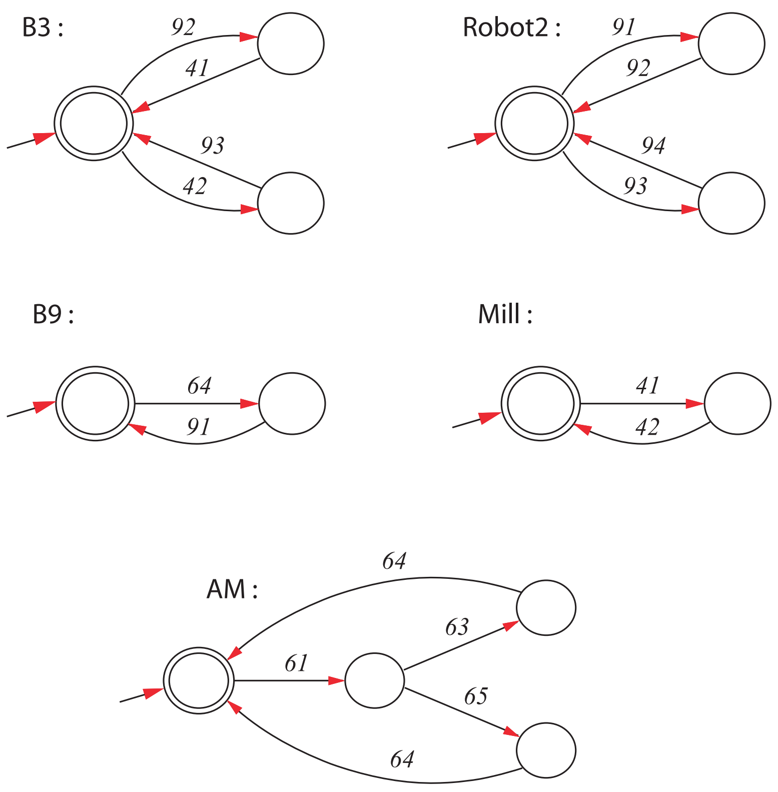

Figure 5.8: Additional components of the extended FMS example

some of the necessary design choices. Ultimately, it may be necessary to try multiple combinations to find a satisfactory solution.

Algorithm 5.29. *Multi-Level Interface-Based Control Construction*
Step 1: Group system components into modules - The grouping of the components of the global system into modules has many different possibilities that in general do not lead to a unique solution. The alphabet partitions of equations (5.1) and (5.2) must be kept in mind during the grouping process. For one, all interaction between modules must take place through interfaces and each interface is completely disjoint from all other interfaces. Additionally, while each module can interact with multiple modules on the level of hierarchy immediately below it, it can only interact with a single module on the level of hierarchy above it. The strict ordering imposed by the alphabet partition also implies that there can be no closed loops formed among the modules, that is, a module cannot be both simultaneously above and below another module in the hierarchy.

We will now present one heuristic approach for grouping the components. To start, pick a specification which is on "the edge" of the system. By the edge, what is meant is a specification that interacts with plant components that do not interact with a lot of other specifications. If this is not possible, all it means is that this module or the module that follows on the next level of hierarchy will have to include more specifications and hence will be larger. For our extended FMS example, inspection of Fig. 5.7 indicates the most logical specification choice would be the buffer *B3*. However, in order to generate an example with multiple levels with multiple modules, we will begin with specification *B2*. The plant components which share relevant events with *B2* are the machines *Con2* and *Robot*. *Con2* does not interact with any other specifications, so it is indeed on the “edge” of the system. *Robot*, however, also shares relevant events with the specifications *B4*, *B6*, and *B7*. Since this module we are building can only interact with a single module on the next level of hierarchy, we will include *B4* in this first module and *B6* and *B7* will be included in the module on the next level. Since the specification *B4* also shares relevant events with the machine *Lathe*, the plant for this first module will be $G^3_1 = \mathit{Con2} \| \mathit{Robot} \| \mathit{Lathe}$ and the corresponding specification will be $E^3_1 = \mathit{B2} \| \mathit{B4}$. The superscript refers to the level of the hierarchy while the subscript refers to the index within a given level. Here we assume we know the index of this level of the hierarchy for the purposes of clarity, but in reality we will not know this until the partitioning process has been completed.

Considering the module on the next level of hierarchy, it is defined by those specifications which share relevant events with the module from the previous level. Examining the plant components associated with the specifications of this module, determine which specifications they further interact with. For our FMS example, the specifications for this module on the second level of hierarchy are *B6* and *B7*. These specifications interact with machines *AM* and *Con3*, which further interact with specifications *B8* and *B9*. Since *B8* is on the “edge” of the system, it is a good choice for another lower-level module. Buffer *B9*, however, interacts with other components of the system so it is a good choice for the next level of hierarchy. Therefore letting $E^3_2 = \mathit{B8}$ and $E^2_1 = \mathit{B6} \| \mathit{B7}$ will satisfy the requirement that each module can interact with multiple lower-level modules, but only a single higher-level module. Looking at the plant components which interact with the specifications, both *B7* and *B8* share relevant events with *Con3*. In our experience, we have determined that in most cases it is preferable to group a plant component with the lowest-level specification with which it shares events. Therefore, the corresponding plants for each of these modules end up being $G_2^3 = \mathit{Con3} \| PM$ and $G_1^2 = AM$.

This process of moving up levels of the hierarchy and backtracking as necessary to meet the conditions of the architecture continues until all specifications have been addressed. Moving up a level of the hierarchy in our example, the specification associated with the next module is $E_1^1 = \mathit{B9}$. The plant component which shares relevant events with this specification is *Robot2*. This plant component in turn interacts with the specification *B3*. Specification *B3* could logically be associated with a module on the next level of the hierarchy, however, in order again to generate an example which has multiple levels with multiple modules, we will associate this specification with a lower-level module. Therefore, $E_2^2 = \mathit{B3}$. Noting that *B3* and *B9* both interact with the plant component *Robot2*, we will again employ the convention of grouping plant components with the lowest-level specification with which it is associated. Therefore, the plant corresponding to *B3* is $G_2^2 = \mathit{Robot2} \| \mathit{Mill}$. Since all of the plant components relevant to *B9* are grouped with lower-level modules, the "plant" associated with this specification does not introduce any further restriction of behavior. Therefore, G_1^3 is an automaton that generates the language Σ^*.

Since there are no further specifications or plant components, the grouping is finished. The dashed boxes in Fig. 5.7 demonstrate how the global system has been partitioned for this example. Figure 5.9 illustrates the hierarchy imposed upon the system and the flow of information.

Step 2: Determine sets of request and answer events - Examine the plant components of each module and choose which events are to comprise the request and answer events associated with each interface. This again is a heuristic process that depends on designer understanding of the system and may require some iteration.

Here we provide some guidelines to help with the process. First, all relevant events shared between any pair of modules must be included in either the request or answer event set for the associated interface. Additionally, it is often helpful to think of request events as events that start a process and answer events as events that finish a process. Examining the plant automata of a module can give some indication of which events begin and which ones finish a process. It is also often helpful that the request events be controllable.

For our example, the events shared between module H_1^3 and module H_1^2 are 30 and

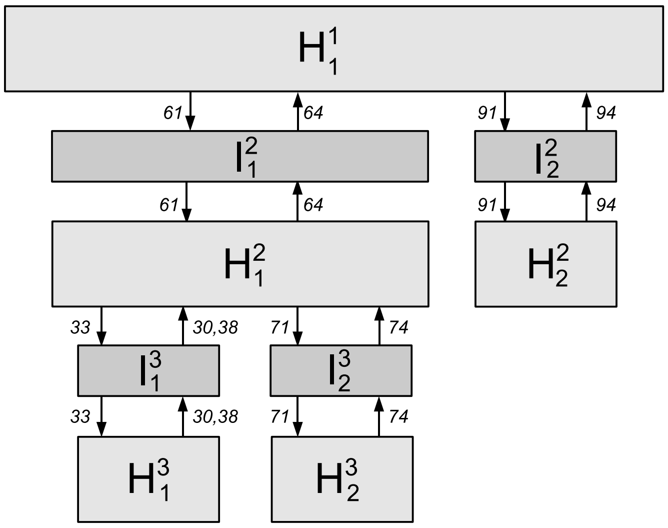

Figure 5.9: Hierarchy imposed on the extended FMS example

38. Examining the automaton model of *Robot* in Fig. 3.10, it can be seen that both of these events represent the completion of a process. Therefore, we will consider them answer events $\Sigma_{A_1^3} = \{30, 38\}$. It can also be seen by inspection that the events that start these two processes correspond to events 39 and 37 respectively. In this case, however, we will take some liberties in what we consider a "process." We will consider our process to be the successive occurrence of two smaller operations. In this instance, the request event is the beginning of the first operation and the answer event is the completion of the second operation. Therefore, we will consider event 33 to be the request event corresponding to both answers $\Sigma_{R_1^3} = \{33\}$. Following this general procedure, we further arrive at the following sets of request and answer events: $\Sigma_{R_2^3} = \{71\}$, $\Sigma_{A_2^3} = \{74\}$, $\Sigma_{R_1^2} = \{61\}$, $\Sigma_{A_1^2} = \{64\}$, $\Sigma_{R_2^2} = \{91\}$, and $\Sigma_{A_2^2} = \{94\}$.

Step 3 (optional): Assume a form for each of the interface automata - Based on the set of request and answer events from the previous step of this procedure, along with the designer's understanding of the system, a form for the interface models may be assumed. Based on the requirements of the MLIBC approach, the interfaces must satisfy the command-pair interface format of Definition 5.1. Often a good interface can be generated based on designer understanding of the system. A bad choice of interface, however, can lead to overly restricted behavior. If a designer is having trouble arriving at a good interface, the interface synthesis approach of Section 5.4

can be employed as part of Step 4 of this procedure.

Step 4: Synthesize supervisors (and optionally interfaces) for each of the lowest-level modules - If interfaces have been assumed in the previous step of this procedure, then supervisors can now be synthesized by the approach of Section 5.3. In this step, we will address those modules that do not have any modules directly below them in the hierarchy. The construction of supervisors then approximates the low-level supervisor synthesis algorithm of [39] with the only difference being with regard to satisfying Point 4 of the multi-level consistency requirement. The basic idea is that the interface, specification, and plant component automata associated with a given module are composed. States of this resulting automaton are then removed if they fail any of the necessary low-level requirements or if they are not coreachable. The low-level requirements needed for these modules correspond to Point 1 and Points 3-5 of Definition 5.17.

If an interface has not been assumed for a given module by the previous step, then it is possible to synthesize an interface along with the supervisor for a given module. Specifically, for a given set of request events $\Sigma_{R_k^i}$ and answer events $\Sigma_{A_k^i}$, the interface can at first be assumed to be the DES that generates the language $(\Sigma_{R_k^i}.\Sigma_{A_k^i})^*$. The start of the supervisor synthesis process is then to compose this interface automaton with the corresponding specification and plant automata. The interface is then considered to be the natural projection of this composition as prescribed in equation (5.7). The states of the resulting composed automaton are then examined to see if they satisfy the necessary low-level requirements and are coreachable. If a state fails a requirement, it is removed and the corresponding interface is then recalculated again according to equation (5.7). Once all states have been examined, then the resulting automaton generates the supervisor language for this module and the projection of the supervisor language is the interface language. Note, the order in which the states are evaluated will affect the resulting interface and in turn the resulting supervisor.

For our example, the lowest-level modules correspond to H_1^3, H_2^3, and H_2^2. Specifically for module H_1^3, the supervisor S_1^3 is built to control the "plant" G_1^3 to meet the low-level requirements with respect to the specification E_1^3 and request and answer events $\Sigma_{R_1^3}$ and $\Sigma_{A_1^3}$. The resulting interfaces for these modules are included in Fig. 5.10.

Step 5: Synthesize supervisors (and optionally interfaces) for modules on the next

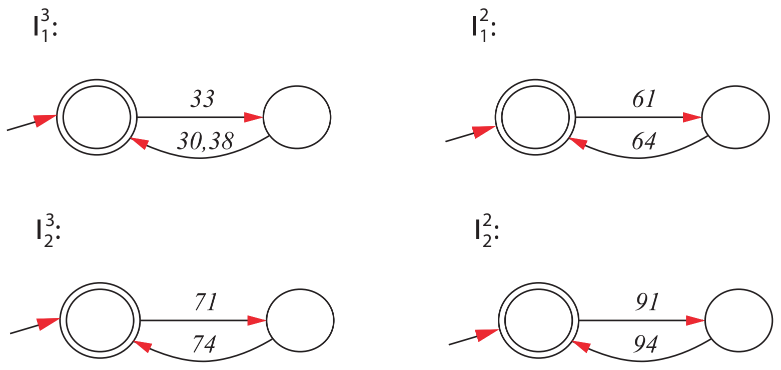

Figure 5.10: Resulting interfaces for the extended FMS example

level of hierarchy - Move up a level of hierarchy to address those modules directly above the ones for which supervisors were just constructed. If there are no further levels above the current module, then the associated supervisor can be constructed using the high-level supervisor synthesis algorithm of [39]. In the multiple-level case, the same high-level requirements are employed, so the algorithm of [39] can be used directly.

If there are levels above a module and interfaces have been assumed, then the approach of Section 5.3 must be employed. If interfaces have not yet been assumed, then they can again be synthesized in conjunction with the supervisors. This process is identical to Step 4, except now both high and low-level requirements must be satisfied. In other words, each state must be reached by strings that satisfy each of the points of the i-k^{th} multi-level interface controllability definition and each state must be coreachable.

For those modules considered in the previous step, the modules that follow them are H_1^2 and H_1^1. However, since H_1^2 precedes H_1^1 in the hierarchy, we will wait to address the supervisor for H_1^1 until later. Therefore, for module H_1^2 the synthesis algorithms are applied to build a supervisor S_1^2 for the plant G_1^2 in order to satisfy requirements with respect to the specification E_1^2, the interfaces that precede the module $\{I_1^3, I_2^3\}$, and the request and answer events $\Sigma_{R_1^2}$ and $\Sigma_{A_1^2}$.

Note that in the process of building the supervisor for the module H_1^2, the preceding modules H_1^3 and H_2^3 did not need to be considered at all. All necessary information was passed through the interfaces. This illustrates how global properties are met through the construction of local supervisors.

Step 6: Repeat until done - Repeat Step 5 until supervisors have been constructed for all modules. At the conclusion of the procedure, the control implemented by the local supervisors and interfaces will provide safe, nonblocking control.

In our example, the only module left is H_1^1. For this module, the high-level synthesis algorithm is applied to generate a supervisor S_1^1 for the plant G_1^1 with respect to the specification E_1^1 and the interfaces I_1^2 and I_2^2. $\diamond$

Table 5.1 summarizes the procedure applied in this example.

Table 5.1: Application of Algorithm 5.29 to extended FMS example

Step	Automaton Built	States (Transitions)	Notes
1	$G_1^3 = Con2\|Robot\|Lathe$	24(92)	
	$E_1^3 = B2\|B4$	8(22)	
	$G_2^3 = Con3\|PM$	6(14)	
	$E_2^3 = B8$	3(4)	
	$G_1^2 = AM$	4(5)	
	$E_1^2 = B6\|B7$	6(14)	
	$G_2^2 = Robot2\|Mill$	6(14)	
	$E_2^2 = B3$	3(4)	
	G_1^1	-	G_1^1 generates the language Σ^*
	$E_1^1 = B9$	2(2)	
2	$E^3_{alt_1}$	2(3)	$\Sigma_{R_1^3} = \{33\}$ $\Sigma_{A_1^3} = \{30, 38\}$
	$E^3_{alt_2}$	2(2)	$\Sigma_{R_2^3} = \{71\}$ $\Sigma_{A_2^3} = \{74\}$
	$E^2_{alt_1}$	2(2)	$\Sigma_{R_1^2} = \{61\}$ $\Sigma_{A_1^2} = \{64\}$
	$E^2_{alt_2}$	2(2)	$\Sigma_{R_2^2} = \{91\}$ $\Sigma_{A_2^2} = \{94\}$
3			skipped
4	$Z_1^3 = G_1^3\|E_1^3\|E^3_{alt_1}$	36(65)	uncontrolled subsystem
	S_1^3	27(46)	component supervisor
	I_1^3	2(3)	higher-level interface
	$Z_2^3 = G_2^3\|E_2^3\|E^3_{alt_2}$	6(6)	uncontrolled subsystem
	S_2^3	6(6)	component supervisor
	I_2^3	2(2)	higher-level interface
	$Z_2^2 = G_2^2\|E_2^2\|E^2_{alt_2}$	6(6)	uncontrolled subsystem
	S_2^2	6(6)	component supervisor
	I_2^2	2(2)	higher-level interface
5	$Z_1^2 = G_1^2\|E_1^2\|E^2_{alt_1}\|I_1^3\|I_2^3$	80(182)	uncontrolled subsystem
	S_1^2	18(29)	component supervisor
	I_1^2	2(2)	higher-level interface
6	$Z_1^1 = G_1^1\|E_1^1\|I_1^2\|I_2^2$	8(12)	uncontrolled subsystem
	S_1^1	6(8)	component supervisor

For the purposes of comparison, the composition of all plant and specification components in the extended FMS example results in an automaton with 291,456 states and 1,226,672 transitions. Furthermore, the supremal controllable sublanguage for the monolithic system is generated by an automaton with 20,232 states and 80,028 transitions. A traditional modular solution greatly reduces the complexity of generating control for this example, but results in blocking.

In the generation of the multiple-level hierarchical interface-based control, the largest automaton that was constructed had 80 states and 182 transitions. This automaton was built in the process of constructing the supervisor for module H_1^2. The resulting global closed-loop behavior is safe and nonblocking. The size of the automata built in this process are substantially smaller than those required in building the monolithic supervisor, thereby giving some indication of the advantage of this approach. Granted, the process by which the sublanguages are constructed in the interface-based solution is different than the process used for constructing the traditional supremal controllable sublanguage, as was done in the monolithic case. The real drawback of the interface-based solution, however, is the loss of optimality. Specifically, the interface-based control only allows for four pieces to be active in the factory at any given time. The monolithic solution allows for a maximum of eight pieces to be active at one time.

If instead a two-level architecture is employed with low-level modules consisting of H_2^2, H_1^3, and H_2^3, and a high-level module made up of the remaining components, then the largest automaton that needs to be constructed has 320 states and 888 transitions. This solution allows a maximum of five pieces to be active at any given time.

For the original FMS employed in the examples of Section 3.4.2 and Section 4.6 and shown in Fig. 1.3, the MLIBC approach can also be employed. For the original AM and a high-level with the partitioning of module H_1^2 and two low-level modules with the partitioning of H_1^3 and H_2^3, the largest automaton that must be constructed again has 80 states and 182 transitions. This compares favorably to the monolithic system that requires an automaton with $13,248$ states and $46,424$ transitions be constructed. In terms of permissiveness, the monolithic solution allows six pieces to be active in this FMS example at the same time, while the interface-based solution only allows three pieces to be operated on at one time.

5.5.2 Complexity discussion

For a two-level interface-based architecture, efficient algorithms for the verification of properties [37] and the synthesis of component supervisors [7] have been developed. For each module with its interfaces, these algorithms have complexity that is polynomial in the number of states and events. Since the high-level module is composed with all of its interfaces at once, it is in most cases the factor limiting how large of a system can be constructed and verified. It has been demonstrated that an interface-based approach is often worthwhile in terms of complexity savings if the interfaces are at least an order of magnitude smaller than their corresponding low-level modules [41].

Since it is required that the low-level modules be completely disjoint from one another, if the global system is made larger, it is often the case that the high-level will grow and the number of low-level modules will increase. Therefore, the scalability of a two-level architecture is limited since the number of states of a synchronous composition grows exponentially with the number of components. This is where the advantage of a multiple-level architecture becomes apparent. Figure 5.11 illustrates a possible partitioning of a larger version of the FMS example used in the previous section for a two-level architecture. In the example of the previous section, the proposed two-level partitioning resulted in a high-level consisting of four automata and three low-level components. For the expanded system of Fig. 5.11, the high-level has grown to include thirteen automata and the number of low-level components has increased to six.

For the multiple-level architecture proposed in this chapter, we have not yet developed efficient algorithms for the verification of properties or the synthesis of component supervisors. We believe, however, that algorithms can be developed that will have polynomial complexity in the number of states and events of a given module and its interfaces just like in the two-level case. In the case of interface synthesis, however, we are not as certain that a polynomial complexity algorithm can be developed. The problem that arises here is that the natural projection operation can in the worst case have exponential complexity. The goal, however, would be to keep the individual modules small enough that the overall complexity is not prohibitive.

As stated earlier, the true advantage of the multiple-level architecture is its scalability. We have argued that in the two-level case as the global system grows the high-level will grow and the number of low-level components will increase. In the multiple-level case, however, it is possible to limit the size of the modules and the number of corresponding low-level components by increasing the number of levels in the hierarchy. Therefore, we can in effect put a bound on the number of interfaces

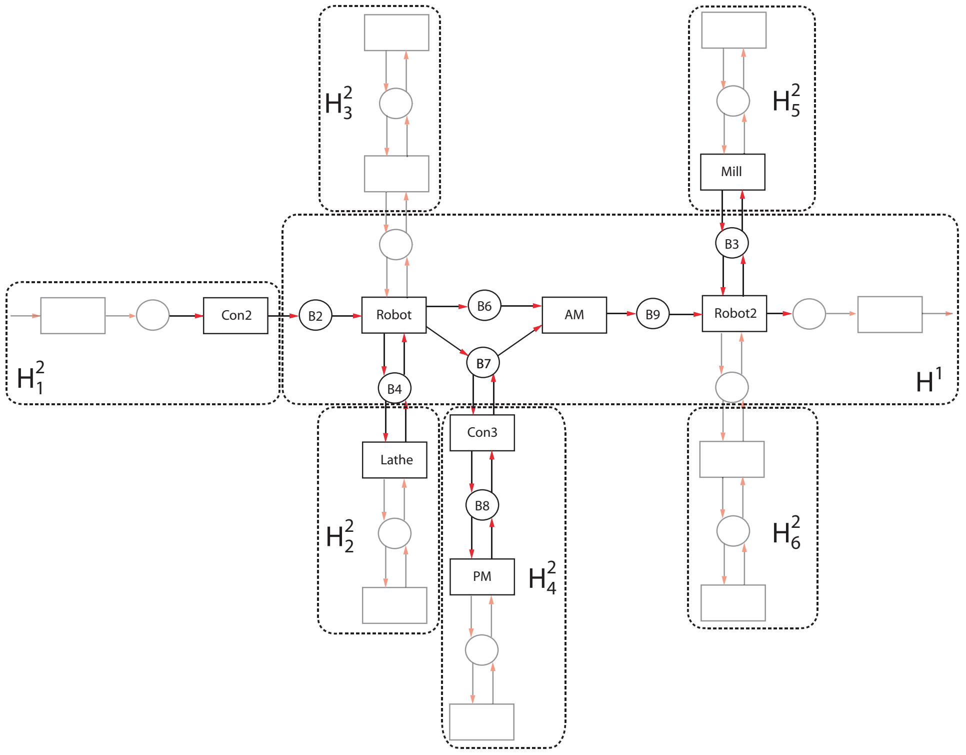

Figure 5.11: Two-level partition of a larger FMS example

that any given module must be analyzed with respect to at once. Considering the FMS example from the previous section with the multiple-level partitioning shown in Fig. 5.7, the most automata in a given module was five and the maximum number of interfaces for a given module was three. For the larger FMS example of this section, Fig. 5.12 shows a possible partitioning for the multiple-level architecture. For this partitioning, the most automata in a given module is also five, while the maximum number of interfaces for a single module has increased to four. Here one can see the size of the individual modules with their interfaces has stayed roughly the same, even though the global system has grown significantly.

5.6 Chapter Summary

In this chapter we have provided requirements for a multiple-level interface-based architecture by which global controllability and nonblocking can be verified locally. This general architecture is an improvement over the special two-level case of [41] in that it allows the global system to be partitioned into smaller modules, thereby leading to less complexity and improved reconfigurability, though at the possible expense

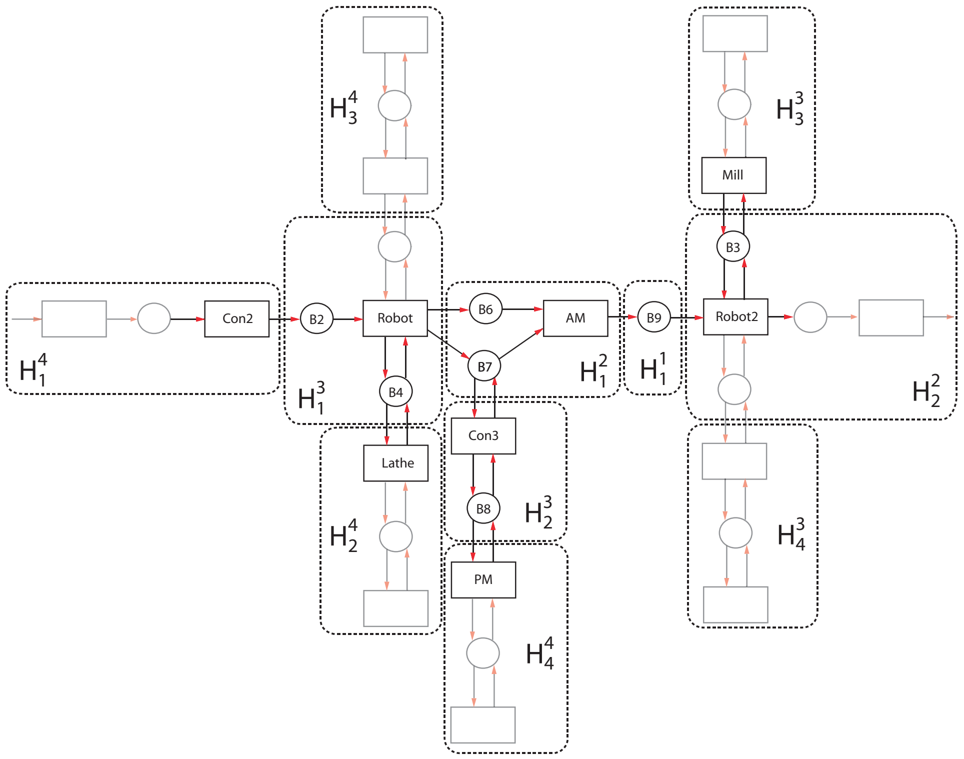

Figure 5.12: Multiple-level partition of a larger FMS example

of increased restrictiveness. Furthermore, the interface consistency requirements of this chapter are shown to be a relaxation of the corresponding requirements of [40]. This chapter also extended the work of [39] to show that a supervisor for each module that is maximally permissive with respect to a given plant, specification, and set of interfaces can be synthesized. It is also demonstrated that these supervisors can be constructed employing automata-based methods.

The use of interfaces, therefore, enables the global controlled system to be verified and designed employing only local information, greatly mitigating the state-space explosion problem in most cases. The general interface-based approach is different than many existing techniques for addressing the complexity problem in that it is truly modular. Most of the techniques developed recently [17] [60] [76], including the IHSC and EBCR approaches of this dissertation, are compositional approaches that build up the global system employing abstraction. These other approaches are thus dependent on their abstraction for the amount of complexity reduction they are able to achieve.

One of the limitations of an interface-based approach to supervisory control is that in many cases it produces suboptimal behavior. An interface-based approach is also difficult to automate since it requires designer input in terms of partitioning the global system, choosing request and answer events, and forming interfaces. In this chapter we tried to help with these design choices by providing some guidelines and by outlining an approach for interface synthesis. The proposed approach constructs interfaces as the projection of the closed-loop behavior of the associated module. This approach offers a promising starting point for a problem that has not been addressed in the literature, though it is not very computationally efficient. By constructing the interfaces as an abstraction of a lower-level module, this overall approach begins to resemble the compositional approaches proposed throughout the literature. In particular, the MLIBC approach is similar to the IHSC approach, except it places requirements on the construction of the supervisor rather than on the abstraction operation.

A natural direction for future work would be to formalize the proposed approach to interface synthesis and to make it more computationally efficient. Other approaches to interface construction could also be explored. Another direction of work would be to attempt to further generalize the hierarchical interface-based architecture. Specifically, it could be useful if conditions were found under which a single “low-level” module could interact with more than one “high-level” module. This generalization would provide more flexibility in the partitioning of the global system.

CHAPTER 6

Conclusions and Future Work

This chapter summarizes the contributions of this dissertation in the context of the greater body of work that exists in the literature. This chapter also proposes some directions for future work that relate to the results that have already been presented.

6.1 Contributions

The contribution of this work is the development of three new approaches to the verification and design of DES. These approaches employ incremental and modular methods, as well as abstraction, to mitigate the problem of state-space explosion that plagues traditional methods for the analysis and design of DES.

6.1.1 Approach I: Incremental Hierarchical Supervisor Construction

The first methodology of this dissertation is referred to as Incremental Hierarchical Supervisor Construction (IHSC) and its details are presented in Chapter 3. The IHSC approach constructs modular supervisors each with respect to a single specification and a "plant" that represents an incrementally larger portion of the global system. Within a given level of the hierarchy, the supervisors are built with respect to disjoint portions of the system. The "plant" for each supervisor is composed of those closed-loop systems from the previous level of the hierarchy and those new subplants that are relevant to the corresponding specification. This approach to synthesis leads the resulting modular supervisor languages to be nonconflicting by construction since either the supervisors address components that are disjoint (within a given level) or the supervisors allow sets of behavior that are subsets of one another (between levels). To reduce the complexity of this approach, abstraction is applied to each supervised subsystem each time we move up a level of the hierarchy. Specifically, a natural projection with the observer property is employed. The resulting modular supervisors are proven to provide safe, nonblocking control when acting in conjunction. The potential complexity savings provided by this approach are demonstrated through application to two moderately sized examples.

6.1.2 Approach II: Equivalence-Based Conflict Resolution

This dissertation's second approach to supervisor construction is presented in Chapter 4 and is referred to as Equivalence-Based Conflict Resolution (EBCR). The EBCR approach first builds traditional local modular supervisors in the sense of [8]. The automata representing the closed-loop modules are then incrementally abstracted and composed. In this process, a conflict-equivalent abstraction is employed and each time a new module is added the resulting composition is checked for blocking. If the composition is nonblocking then nothing needs to be done; if blocking is detected, then a filter law is synthesized to resolve the conflict within the composition. The resulting behavior achieved by the modular supervisors and conflict-resolving filters acting in conjunction is proven to be safe and nonblocking.

The algorithm provided for constructing the conflict-resolving filters is also a contribution in that it synthesizes a covering-based supervisor that provides a solution to the state avoidance problem that is less restrictive than any state-feedback methodologies that currently exist in the literature. This algorithm can also be applied to nondeterministic and partially-observed systems and has polynomial complexity.

The promise of this overall approach is again demonstrated through application to a moderately sized FMS example.

6.1.3 Approach III: Multi-Level Interface-Based Control

The final approach to verification and supervisor construction presented in this work is the Multi-Level Interface-Based Control (MLIBC) approach. The details of the MLIBC approach can be found in Chapter 5. This work partitions the global system into modules and then introduces interfaces between the modules to restrict interaction and allow global properties to be verified by local analysis. Specifically, requirements are presented that can be verified by analyzing each module with respect to its neighboring interfaces. These local requirements are then shown to provide nonblocking of the global system and controllability of the global supervisor with respect to the global plant.

Beyond verification, the use of interfaces also allows control to be synthesized locally. In this work, a naive approach is proposed that is demonstrated to synthesize component supervisors that are maximally permissive with respect to a given plant, specification, and set of interfaces. An approach is also presented for synthesizing the interfaces required of the MLIBC approach. The proposed interface construction methodology offers a promising solution to a problem that has not yet been addressed in the literature. Heuristics for implementing the overall MLIBC approach, as well as its potential complexity savings, are demonstrated through application to different versions of the FMS example.

6.2 Discussion

The three approaches to supervisor construction presented in this dissertation each possess their own advantages and disadvantages. In particular, there is often a trade-off between the complexity reduction provided by an approach and the permissiveness of the control it generates. This is also true of other likeminded approaches to supervisor construction that exist in the literature. Unfortunately, it is often not possible to strictly compare the complexity or permissiveness of two different approaches; this can generally only be accomplished on a case-by-case basis. In this section we will specifically discuss the three approaches of this dissertation with respect to the FMS example first shown in Fig. 1.3. The details of this FMS model can be found in Section 3.4.2. In the process of presenting the results for this example we will try and point out some trends between the three approaches of this dissertation. We will additionally try and put these approaches into context with respect to the larger field of research.

The IHSC approach in many cases generates the most permissive control of the three approaches of this dissertation. For the FMS example, the language representing the behavior allowed by the set of modular supervisors found from both applications of the IHSC approach is contained in the language allowed by the monolithic solution. However, the monolithic and modular solution both allow a maximum of six pieces to be active in the factory at the same time. A monolithic supervisor will always provide maximally permissive behavior and hence is a good benchmark. The loss of optimality in the IHSC approach arises because a controllable event that has been projected away cannot be disabled by a supervisor. Therefore, a supervisor may have to disable a different controllable event that is less desirable in order to keep the system safe.

Like the IHSC approach, the work of [17] that was developed at approximately the same time also employs a natural projection with the observer property. This work additionally requires an output-control-consistency property that limits which controllable events may be projected away in order to guarantee that their modular solution is maximally permissive. The addition of the output-control-consistency property reduces the amount of complexity reduction that is possible. We conjecture that the addition of output-control-control consistency to the IHSC requirements would result in maximally permissive control. This, however, has not been proven. We also conjecture that the removal of the output-control-consistency property from the approach of [17] will result in safe, nonblocking control and a greater reduction in computational complexity.

The architecture of [17] is different than that employed in the IHSC approach in that it builds modular supervisors then generates an additional level of control to resolve conflict among the supervisors. This structure increases the amount of events that are shared among the modules and hence limits the number of events that can be considered for abstraction as compared to the IHSC approach. This fact provides another reason why that in many cases the IHSC approach will require less complexity than the approach of [17].

The EBCR approach of this dissertation employs a similar architecture to [17]. The EBCR approach, however, employs a conflict-equivalent abstraction rather than a natural projection with the observer property. A conflict-equivalent abstraction in general generates a smaller model than an observer-type abstraction. This means that, in most cases, the EBCR approach requires the construction of smaller automata than the IHSC approach or the approach of [17]. This trend is supported by application to the FMS example. As shown in Table 6.1, the IHSC approach requires that an automaton with at least 210 states and 516 transitions be constructed, while the EBCR approach requires that an automaton with only 128 states and 428 transitions be constructed.

Most algorithms presented in this dissertation have complexity that is polynomial

Table 6.1: Summary of Results for the FMS example

case	$\sharp$ states ($\sharp$ transitions) in largest supervisor	$\sharp$ states ($\sharp$ transitions) in largest intermediate automaton	maximum $\sharp$ pieces active
monolithic	2256 (7216)	13,248 (46,424)	6
IHSC modular 1	165 (435)	220 (609)	6
IHSC modular 2	106 (270)	210 (516)	6
EBCR modular	80 (259)	128 (428)	5
MLIBC modular	27 (46)	80 (128)	3

in the number of states and events of a given automaton. Therefore, automata size is a reasonable metric of computational complexity. An exception to the polynomial complexity generalization that arises in the EBCR approach is the generation of the conflict-equivalent abstraction. The process of generating this abstraction is based on a heuristic application of a select set of rules and hence its complexity is less well understood. A better understanding of the generation of conflict-equivalent abstractions remains an open area of research.

Even though the conflict-equivalent abstraction generates a greater reduction in model size than a natural projection with the observer property, there are cases where the IHSC approach requires the construction of smaller automata than the EBCR approach. These cases arise because of the increased sharing of events between modules described in the discussion of [17].

While in most cases the EBCR approach requires the construction of smaller automata than the IHSC approach, it also in most cases generates control that is more restrictive. This trend is exemplified by application to the FMS model. The EBCR solution for this system allows only five pieces to be active in the factory at one time, while the IHSC solution allows six pieces to be active at once. The EBCR approach loses optimality because like the IHSC approach it allows controllable events to be hidden. The further restrictiveness of the EBCR approach arises because the conflict-equivalent abstraction introduces nondeterminism. This nondeterminism reflects the fact that the abstraction hides which state the underlying system is actually in. This in turn leads the filter laws to be more conservative than they would otherwise need to be.

The MLIBC approach is the final methodology presented in this dissertation. This approach in most cases requires the construction of the smallest automata and results in the least permissive control. This statement is supported by the results of the FMS example presented in Table 6.1. The fact that the verification and supervisor design of the MLIBC approach is truly modular is what enables such large complexity savings. The modularity also makes the MLIBC approach more reconfigurable than the other two approaches. The restrictiveness of the approach arises because the structural requirements on the supervision is in most cases stronger than the requirements placed on the abstractions of the other approaches. If we assume that the interfaces are formed heuristically, then we believe algorithms for the verification of the MLIBC requirements and for the synthesis of the modular supervisors can be developed that have polynomial complexity in the number of states and events of a given module with its interfaces.

We further believe the trends demonstrated by the results in Table 6.1 will become more pronounced with application to larger examples.

6.3 Future Work

Directions of future research that relate to the work of this dissertation include addressing the implementation of the existing approaches, applying the methods of the presented work to new problems, and exploring new techniques for model reduction.

6.3.1 Practical implementation

The three approaches to supervisor synthesis presented in this work, as well as many related works, are new theoretical results that have been applied to only a small number of moderately-sized academic examples. In order to better understand the true applicability of these results, as well as to better understand how the different approaches relate to one another, it is necessary that these techniques be applied to a wider variety of larger-scale examples. This goal is made difficult by the fact that many of the existing approaches require input from the designer during the synthesis process. Furthermore, many of the proposed algorithms have not been implemented in software yet. Addressing these difficulties provide one possible direction for future research activities.

Examples where designer input enters in the IHSC approach include choosing the order in which specifications are addressed and determining which events are to be abstracted away. Algorithm 3.18 provides one approach for choosing the ordering of specifications. The investigation of other ordering methodologies is still an open problem. The work of [15] presents a polynomial time algorithm for finding an extension of the set of observable events for a projection such that the projection is an observer. There does not, however, exist in general a minimal set of such events. Therefore, heuristic algorithms for determining a "good" set of events to retain could still be investigated.

For the EBCR approach, the order in which the closed-loop modules are composed and abstracted also requires designer input. Results presented in [21] evaluate several strategies based on the size of the automata or the size of the shared alphabets. This investigation tries to minimize the size of the resulting automata and could also provide inspiration for choosing the order of specifications in the IHSC approach. A reason to revisit the investigation of [21] is that that work was only interested in detecting conflict, not in synthesizing supervisors. Therefore, it would be interesting to see how the ordering schemes compare in terms of the permissiveness of the resulting control. Along these lines, it could also be informative to explore the possibility of waiting to construct filters in the EBCR approach. The reason this may be beneficial is because it is possible that conflict within a composition of modules could be resolved by the addition of other modules without the need of a conflict-resolving filter. The trade-off that arises here is that by waiting to construct the filter, more events are hidden, thereby limiting the set of transitions the filter law can disable to prevent the conflict if it is present. Determining which events to hide is another designer input that could be investigated.

With regard to the EBCR approach, the implementation of the conflict-equivalent abstraction could also be investigated. The abstraction is achieved by the application of a set of heuristic rules. The order in which these rules are applied is something that could be evaluated. It is also possible other rules for generating conflict-equivalent abstractions could be found. To apply the EBCR approach to large-scale systems, it would also be necessary to implement the filter construction algorithm (Algorithm 4.27) in software.

The MLIBC approach also requires designer input in determining the partitioning of the global system, choosing request and answer events, and forming interfaces. Guidelines for making some of these choices were presented in Section 5.5.1, but this area could be explored further. The approach to interface synthesis outlined in Section 5.4 is another area that could be investigated, in particular, with regard to improving the computational complexity of the approach. Finally, algorithms for verifying the MLIBC requirements and implementing the proposed approach to supervisor synthesis need to be developed and implemented in software. It is proposed that existing algorithms for verification and supervisor synthesis in the two-level case be followed [7] [37].

Another way to increase the size of systems to which the approaches of this dissertation can be applied is to employ more computationally efficient data structures than automata, such as binary decision diagrams [3] and state tree structures [46]. For the two-level interface-based architecture, binary decision diagrams have already been employed [62].

6.3.2 Reducing complexity in diagnosis

Another direction for research would be to apply some of the techniques developed in this dissertation for supervisor synthesis to constructing diagnosers for determining the occurrence of faults. Fault diagnosis is a problem that suffers from the same state-space explosion issues that motivated the work of this dissertation.

An existing approach proposed by [79] applies an abstraction to the monolithic plant model such that the diagnoser constructed from this reduced model produces the same diagnostic information as the diagnoser built from the original model. The model reduction employed is a state aggregation approach that partitions the original state space based on the fault properties of the states. This approach is a similar concept to the state aggregation employed in [30] to ease the complexity of supervisor construction. A requirement on the original model is that each state be distinctly labeled as a fault or normal state. It may be possible to transform any automaton model into a form that has this property.

A possible direction for future research would be to apply this approach to model reduction in a more incremental manner. That is, apply the model reduction to each component before composing the components into the monolithic model. This would provide benefits in that it would make the process of model reduction simpler and would avoid the construction of the unabstracted monolithic plant. In order for this approach to work, it would be necessary to be show that the order of the operations of reduction and composition can be commuted. An indication that this might be possible is the fact that the projection operation can be distributed across parallel composition when shared events are not erased.

Another idea concerning reduction of the number of states in the diagnoser relates to the observer property. One of the benefits of the observer property is that it was shown to guarantee that the complexity of the projection operation is polynomial if the operation possesses the observer property [72]. Since the construction of the diagnoser is essentially taking a projection and adding labels, it seems the observer property could guarantee that the diagnoser state space will not grow exponentially. Furthermore, since [54] has shown that the observer property is maintained across composition in the case that shared events are not erased, this means that an incremental approach might be applied here too.

Modular approaches to diagnosis have been also been formulated [6] [10]. The idea behind modular diagnosis is that diagnosers are built for each component of a larger system. In the case that each component is diagnosable, then the system as a whole is diagnosable. Therefore, in this instance, the modular approach to diagnosis provides the same information as a centralized diagnoser built for the monolithic system. The more interesting case, however, is if not all the individual modules are diagnosable. If not all of the components are diagnosable it is possible that the overall system is still diagnosable by a centralized diagnoser. This difference arises because not all strings accepted by a component are necessarily accepted by the monolithic system. This happens because some strings accepted by a particular component may be blocked by a different component. Therefore, if there are two traces accepted by a component that reflect different failure conditions, it is possible that one of the strings will be blocked by one of the other components thereby eliminating the ambiguity.

The work of [10] addresses this uncertainty by generating a test that determines whether or not a system is modularly diagnosable. Unfortunately, the test requires knowledge of the monolithic diagnoser which defeats part of the motivation for using a modular approach. The work of [6] rather uses an incremental approach that begins with a component diagnoser that cannot be diagnosed and considers additional modular diagnosers one at a time until the ambiguity is resolved or until all components are exhausted. A direction for future work would be to improve upon these modular approaches, in particular, by combining them with model reduction techniques.

6.3.3 Additional model reduction techniques

Within this work two types of abstraction were employed, natural projection with the observer property and conflict-equivalent abstraction. These two approaches to model reduction maintain different system properties and hence are suited to different approaches to supervisor synthesis. Specifically, the observer property provides observation equivalence so that the supervisor can be designed for the reduced model and will achieve safe, nonblocking behavior when it is applied to the full model. Conflict-equivalent abstraction maintains conflict properties and is therefore well-suited to designing a conflict-resolving level of control. Other abstraction techniques have also been developed that may be suited to other approaches to supervisor synthesis.

A new type of reduction that has been developed that could be investigated is the notion of a supervision-equivalent abstraction [22]. This approach to reduction is able to capture the same information as the maximally permissive supervisor, but in a more compact manner. It is, however, unclear what the complexity associated with synthesizing the supervisor is. Another approach to model reduction is referred to as partial order techniques [14]. The idea here is that when concurrent subsystems are composed, there may be events in the alphabets of the subsystems whose relative order is not important. Therefore, partial-order techniques reduce the complexity of a model by not capturing all the permutations of the orderings of these events. Another approach to abstraction is able to generate a reduced model by recognizing repeated structure in the components of a larger system. This approach to abstraction is referred to as a symmetry reduction [12] [59].

Each of the above reduction techniques could be explored in the context of supervisor or diagnoser synthesis. Additionally, multiple reduction techniques could be employed together. For example, either the IHSC approach or the EBCR approach could be applied to an individual module in the multiple-level interface-based architecture. Similarly, an observer-type abstraction could possibly be employed in constructing the modular supervisors in the EBCR approach before the conflict-resolving filters are constructed.

APPENDIX

APPENDIX

Proposition 3.5:
Let K_1, K_2, $L \subseteq \Sigma^*$ be languages and let $K = K_1 \cap K_2$. Also let $\Sigma_u \subseteq \Sigma$ and K_1 and K_2 be nonconflicting. If K_2 is Σ_u-controllable with respect to $\overline{K_1} \cap L$, and K_1 is Σ_u-controllable with respect to L, then K is Σ_u-controllable with respect to L.

Proof.

$$\begin{aligned}
\overline{K_2}\Sigma_u \cap (\overline{K_1} \cap L) &\subseteq \overline{K_2} \\
\Rightarrow \overline{K_2}\Sigma_u \cap (\overline{K_1} \cap L) &\subseteq \overline{K_1} \cap \overline{K_2} \\
\Rightarrow \overline{K_2}\Sigma_u \cap ((\overline{K_1}\Sigma_u \cap L) \cap L) &\subseteq \overline{K_2}\Sigma_u \cap (\overline{K_1} \cap L) \subseteq \overline{K_1} \cap \overline{K_2} \\
\Rightarrow (\overline{K_2} \cap \overline{K_1})\Sigma_u \cap L &\subseteq \overline{K_1} \cap \overline{K_2} \\
\Rightarrow (\overline{K_1 \cap K_2})\Sigma_u \cap L &\subseteq \overline{K_1 \cap K_2}
\end{aligned}$$

□

Proposition 3.11:
Let $P : \Sigma^* \rightarrow \Sigma_a^*$ be a natural projection and let $L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. Also let $\overline{\widetilde{K}_m} = P^{-1}(\overline{K_a}) \cap L$, $\Sigma_u \subseteq \Sigma$ and $\Sigma_{u,a} = \Sigma_u \cap \Sigma_a$. If K_a is $\Sigma_{u,a}$-controllable with respect to $P(L)$, then $\widetilde{K}_m$ is Σ_u-controllable with respect to L.

Proof. Since it is given that K_a is $\Sigma_{u,a}$-controllable with respect to $P(L)$, we have that $\overline{K_a}\Sigma_{u,a} \cap P(L) \subseteq \overline{K_a}$.

If $\sigma \in \Sigma_{u,a}$, then by the above we obtain

$$\begin{aligned}
\overline{K_a}\sigma \cap P(L) &\subseteq \overline{K_a} \\
P^{-1}(\overline{K_a}\sigma) \cap P^{-1}(P(L)) &\subseteq P^{-1}(\overline{K_a}) \\
P^{-1}(\overline{K_a})\sigma \cap L &\subseteq P^{-1}(\overline{K_a}) \\
(P^{-1}(\overline{K_a}) \cap L)\sigma \cap L &\subseteq P^{-1}(\overline{K_a}) \cap L \\
\overline{\widetilde{K}_m}\sigma \cap L &\subseteq \overline{\widetilde{K}_m}
\end{aligned}$$

If $\sigma \in (\Sigma_u - \Sigma_{u,a})$, then $P(\sigma) = \varepsilon$, so

$$\begin{aligned} P[P^{-1}(\overline{K_a})\sigma] &\subseteq \overline{K_a} \\ P^{-1}(\overline{K_a})\sigma &\subseteq P^{-1}(\overline{K_a}) \\ (P^{-1}(\overline{K_a}) \cap L)\sigma \cap L &\subseteq P^{-1}(\overline{K_a}) \cap L \\ \overline{\widetilde{K}_m}\sigma \cap L &\subseteq \overline{\widetilde{K}_m} \end{aligned}$$

□

Proposition 4.7:

Let $P : \Sigma^* \longrightarrow \Sigma_a^*$ be a natural projection and let $\overline{L_m} = L \subseteq \Sigma^*$ and $K_a \subseteq \Sigma_a^*$ be languages. Also let $\widetilde{K} = P^{-1}(\overline{K_a}) \cap L$ and $\widetilde{K}_m = P^{-1}(\overline{K_a}) \cap L_m$. If the projection P possesses the L_m-observer property and $K_a \subseteq P(L_m)$, then $\overline{\widetilde{K}_m} = \widetilde{K}$.

Proof. Note that if the L_m-observer property holds for all strings $P(s)t \in P(L_m)$, then it will also hold for all strings $P(s)t \in K_a \subseteq P(L_m)$. In general, $\overline{P^{-1}(\overline{K_a}) \cap L_m} \subseteq \overline{P^{-1}(\overline{K_a})} \cap \overline{L_m} = P^{-1}(\overline{K_a}) \cap L$, therefore it is only necessary to show that $P^{-1}(\overline{K_a}) \cap L \subseteq \overline{P^{-1}(\overline{K_a}) \cap L_m}$.

Let $s \in P^{-1}(\overline{K_a}) \cap L$, therefore, $s \in P^{-1}(\overline{K_a})$. Taking projection of both sides, $P(s) \in P(P^{-1}(\overline{K_a})) = \overline{K_a}$. Hence, $P(s)t \in K_a$ for some $t \in \Sigma_a^*$. Since P has the L_m-observer property, we then know $\exists u \in \Sigma^*$ such that $su \in L_m$ and $P(su) = P(s)t \in K_a$. Therefore, $su \in P^{-1}(K_a)$ since $P^{-1}(P(su)) \subseteq P^{-1}(K_a)$ and hence, $su \in P^{-1}(K_a) \cap L_m$. Therefore, $s \in \overline{P^{-1}(K_a) \cap L_m} \subseteq \overline{P^{-1}(\overline{K_a}) \cap L_m}$. And thus we have shown our desired result, $P^{-1}(\overline{K_a}) \cap L \subseteq \overline{P^{-1}(\overline{K_a}) \cap L_m}$ □

Proposition 4.25:

If for the subautomaton $H \sqsubseteq G$, $H^{\uparrow}$ is nonempty and f' is given by equation (4.23), then $Q_{f'} = R'(Q_h^{\uparrow})$.

Proof. By assumption, $H^{\uparrow}$ is nonempty. Step 1 of Algorithm 4.23 then provides that $q_0 \in R'(Q_h^{\uparrow})$. Additionally, since f' only disables transitions, q_0 will be reachable under control. That is, $q_0 \in Q_{f'}$.
($\subseteq$) We will next show that $Q_{f'} \subseteq R'(Q_h^{\uparrow})$ by induction. For any $q \in Q_{f'} - \{q_0\}$, there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying conditions ($C5$-$C7$). For

the basis step, we already have $q_0 \in R'(Q_h^{\uparrow})$. For the induction step, suppose that $q_k \in R'(Q_h^{\uparrow}) \subseteq Q_h^{\uparrow}$. We now want to show that $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq R'(Q_h^{\uparrow})$. Consider two cases:

1. If $\sigma_k \in \Sigma_u$, we then have that $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq Q_h^{\uparrow}$ since $q_k \in Q_h^{\uparrow}$ and $Q_h^{\uparrow}$ is Σ_u-invariant. Furthermore, since $\sigma_k \in \Sigma_u$ we have that $\sigma_k \notin A'_{H^{\uparrow}}(q_k)$. Therefore, $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq R'(Q_h^{\uparrow})$ by Step 2 of Algorithm 4.23.

2. If $\sigma_k \in \Sigma_c$, then by condition $C6$ and equation (4.23), we have $\sigma_k \in f'(q_k) = \Sigma_\tau - A'_{H^{\uparrow}}(q_k)$. Therefore, by Step 2 of Algorithm 4.23 we again have that $q_{k+1} \in \delta_g(q_k, \sigma_k) \subseteq R'(Q_h^{\uparrow})$.

This completes the induction.

($\supseteq$) We will now show that $Q_{f'} \supseteq R'(Q_h^{\uparrow})$. For any $q \in R'(Q_h^{\uparrow}) - \{q_0\}$ there exist $q_1, q_2, \ldots, q_m \in Q_g$ and $\sigma_0, \sigma_1, \ldots, \sigma_{m-1} \in \Sigma_\tau$ satisfying conditions analogous to ($C1$-$C4$), but for $Q_h^{\uparrow}$ instead of Q_h. Since $q \in R'(Q_h^{\uparrow})$ implies $q \in Q_g$, to show that $q \in Q_{f'}$, it is sufficient to prove that $\sigma_i \in f'(q_i)(i = 0, 1, \ldots, m-1)$. By condition $C3$ and equation (4.23), we have $\sigma_i \in \Sigma_\tau - A'_{H^{\uparrow}}(q_i) = f'(q_i)$. □

Theorem 5.10:

If the two-level interface system composed of DES $H^1, H_1^2, I_1^2, \ldots, H_n^2, I_n^2$, is *level-wise nonblocking* and *interface consistent* with respect to the alphabet partition given by (5.1), then the global system is nonblocking:

$$\overline{\mathcal{H}_m^1 \cap \bigcap_{j=1,\ldots,n} (\mathcal{H}_{m_j}^2 \cap \mathcal{I}_{m_j}^2)} = \mathcal{H}^1 \cap \bigcap_{j=1,\ldots,n} (\mathcal{H}_j^2 \cap \mathcal{I}_j^2)$$

Proof. Proof of this theorem follows exactly the logic presented in [37], with the exception of Proposition 13 of [37]. A modification of this proposition is proven below. □

Proposition 13 from [37] applies to a serial two-level interface system, that is, a system with a single high-level module H^1, a single interface I^2, and a single low-level module H^2. The event sets Σ^1 and Σ^2 will represent those events relevant to H^1 and H^2 respectively, that are not request or answer events. In our modification of

Proposition 13, we will employ a modified version of the *serial interface consistency* definition of [37]. Specifically, the Point 4 of this definition has been relaxed as follows:

$$(\forall s \in (\Sigma^*.\Sigma_{A^2})^*.(\Sigma^2)^* \cap \mathcal{H}^2 \cap \mathcal{I}^2)(\forall \rho \in \Sigma_{R^2})\ s\rho \in \mathcal{I}^2 \Rightarrow \\ (\exists l \in (\Sigma^2)^*)\ sl\rho \in \mathcal{H}^2 \cap \mathcal{I}^2 \qquad (1)$$

We will additionally employ the following modified version of Point 5:

$$(\forall s \in \mathcal{H}^2 \cap \mathcal{I}^2)(\forall \rho \in \Sigma_{R^2})(\forall \alpha \in \Sigma_{A^2})\ s\rho\alpha \in \mathcal{I}^2 \Rightarrow (\exists l \in (\Sigma^2)^*)\ s\rho l\alpha \in \mathcal{H}^2 \cap \mathcal{I}^2$$

The following alternate definition of a *command-pair interface* and Proposition 8 from [37] will be referred to in the revised proof of Proposition 13 given below.

Definition: [37] A DES $I^2 = (X^2, \Sigma_{R^2}\dot{\cup}\Sigma_{A^2}, \xi^2, x_0^2, X_m^2)$ is a *command-pair interface* if the following are true:

A) $\Sigma_I^2 = \Sigma_{R^2}\dot{\cup}\Sigma_{A^2}$

B) $(\forall s \in \mathcal{L}(I^2))(\forall \rho \in \Sigma_{R^2})\ s\rho \in \mathcal{L}(I^2) \Rightarrow s \in \mathcal{L}_m(I^2)$

C) $(\forall s \in \mathcal{L}_m(I^2))(\forall \sigma \in \Sigma_{I^2})\ s\sigma \in \mathcal{L}(I^2) \Rightarrow \sigma \notin \Sigma_{A^2}$

D) $\mathcal{L}_m(I^2) = \{\varepsilon\} \cup (\Sigma_{I^2}^*.\Sigma_{A^2} \cap \mathcal{L}(I^2))$

E) $\mathcal{L}(I^2) \subseteq \overline{(\Sigma_{R^2}.\Sigma_{A^2})^*}$

Proposition 8: [37]

a) $(\forall s, s' \in \Sigma^*)\ s \in \mathcal{H}^1$ *and* $P_{H^1}(s) = P_{H^1}(s') \Rightarrow s' \in \mathcal{H}^1$

b) $(\forall s, s' \in \Sigma^*)\ s \in \mathcal{H}_m^1$ *and* $P_{H^1}(s) = P_{H^1}(s') \Rightarrow s' \in \mathcal{H}_m^1$

c) $(\forall s, s' \in \Sigma^*)\ s \in \mathcal{H}^2$ *and* $P_{H^2}(s) = P_{H^2}(s') \Rightarrow s' \in \mathcal{H}^2$

d) $(\forall s, s' \in \Sigma^*)\ s \in \mathcal{H}_m^2$ *and* $P_{H^2}(s) = P_{H^2}(s') \Rightarrow s' \in \mathcal{H}_m^2$

e) $(\forall s, s' \in \Sigma^*)\ s \in \mathcal{I}^2$ *and* $P_{I^2}(s) = P_{I^2}(s') \Rightarrow s' \in \mathcal{I}^2$

f) $(\forall s, s' \in \Sigma^*)\ s \in \mathcal{I}_m^2$ *and* $P_{I^2}(s) = P_{I^2}(s') \Rightarrow s' \in \mathcal{I}_m^2$

Before we begin the revised proof of Proposition 13, we first want to show that the Point 4 of (1) implies the following is true:

$$(\forall s \in (\Sigma^*.\Sigma_{A^2})^*.(\Sigma^1 \cup \Sigma^2)^* \cap \mathcal{H}^2 \cap \mathcal{I}^2)(\forall \rho \in \Sigma_{R^2})\ s\rho \in \mathcal{I}^2 \Rightarrow \\ (\exists l \in (\Sigma^2)^*)\ sl\rho \in \mathcal{H}^2 \cap \mathcal{I}^2 \qquad (2)$$

First let $s \in (\Sigma^*.\Sigma_{A^2})^*.(\Sigma^1 \cup \Sigma^2)^* \cap \mathcal{H}^2 \cap \mathcal{I}^2$ and $s' = P_{H^2}(s)$. Since the projection operation is idempotent this means that $P_{H^2}(s') = P_{H^2}(P_{H^2}(s)) = P_{H^2}(s)$. Also since $\Sigma_{I^2} \subseteq \Sigma_{H^2}$, this means that $P_{I^2}(s') = P_{I^2}(P_{H^2}(s)) = P_{I^2}(s)$. Applying Proposition 8 Point c and Point e, therefore, provides that $s' \in \mathcal{H}^2 \cap \mathcal{I}^2$. We can similarly show that $s' \in (\Sigma^*.\Sigma_{A^2})^*.(\Sigma^2)$. This, therefore, means that s' is subject to (1). Hence, $s'\rho \in \mathcal{I}^2$ where $\rho \in \Sigma_{R^2}$ implies there exists an $l \in (\Sigma^2)^*$ such that $s'l\rho \in \mathcal{H}^2$. Also, since $P_{I^2}(s'\rho) = P_{I^2}(s\rho)$, Proposition 8 Point e provides that $s'\rho \in \mathcal{I}^2$ if and only if $s\rho \in \mathcal{I}^2$. Hence, $s\rho \in \mathcal{I}^2$ implies that there exists an $l \in (\Sigma^2)^*$ such that $s'l\rho \in \mathcal{H}^2$. Since $P_{H^2}(s'l\rho) = P_{H^2}(sl\rho)$, Proposition 8 Point c then provides that $sl\rho \in \mathcal{H}^2$ and we have shown that (1) implies (2). We can therefore use (2) as our Point 4 in the following proposition.

Proposition 13:

If the system composed of DES H^1, H^2, and I^2 is *serial level-wise nonblocking* and *serial interface consistent* with respect to the alphabet partition $\Sigma := \Sigma^1 \dot\cup \Sigma^2 \dot\cup \Sigma_{R^2} \dot\cup \Sigma_{A^2}$, then

$$(\forall s \in \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2)(\forall h \in (\Sigma^1)^*.\Sigma_{R^2}.(\Sigma^1)^*.\Sigma_{A^2})\ sh \in \mathcal{H}^1 \cap \mathcal{I}^2 \Rightarrow$$
$$(\exists u \in \Sigma^*)\ s.t.\ (su \in \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2_m) \wedge (P_{H^1}(u) = h)$$

Proof.

Assume the system is serial level-wise nonblocking and interface consistent. (3)

$$\text{Let } s \in \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2,\ h \in (\Sigma^1)^*.\Sigma_{R^2}.(\Sigma^1)^*.\Sigma_{A^2},\ \text{and } sh \in \mathcal{H}^1 \cap \mathcal{I}^2. \qquad (4)$$

We will now show this implies we can construct a string u with the desired properties.

We first note that $h \in (\Sigma^1)^*.\Sigma_{R^2}.(\Sigma^1)^*.\Sigma_{A^2}$ implies:

$$(\exists h' \in (\Sigma^1)^*)(\rho \in \Sigma_{R^2})(h'' \in (\Sigma^1)^*)(\alpha \in \Sigma_{A^2}) \text{ s.t. } h'\rho h''\alpha = h \qquad (5)$$

We will show that we can construct strings $l', l'' \in (\Sigma^2)^*$ such that $sh'l'\rho h''l''\alpha \in \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}^2_m$. We will also show that $P_{H^2}(h'l'\rho h''l''\alpha) = h$.

Our approach will be to show that $s \in (\Sigma^*.\Sigma_{A^2})^*.(\Sigma^1 \cup \Sigma^2)^* \cap \mathcal{H}^2 \cap \mathcal{I}^2$ and $\rho \in \text{Elig}_{\mathcal{I}^2}(s)$. We will then use Point 4 of the serial interface consistency definition to construct a suitable string l'. Similarly, we will show that $sh'l'\rho \in \Sigma^*.\Sigma_{R^2} \cap \mathcal{H}^2 \cap \mathcal{I}^2$

and $\alpha \in \mathrm{Elig}_{\mathcal{I}^2}(sh'l'\rho)$. We will then use Point 5 of the serial interface consistency definition to construct a suitable string l''.

Since $sh \in \mathcal{I}^2$ (by (4)), we have that $P_{I^2}(sh) \in \mathcal{L}(I^2)$. Additionally, it is given that I^2 is a command-pair interface (by (3)). Therefore by Point E of the command-pair interface definition, $P_{I^2}(sh) = P_{I^2}(s)P_{I^2}(h) \in \overline{(\Sigma_{R^2}.\Sigma_{A^2})^*}$. Since it is given that $h \in (\Sigma^1)^*.\Sigma_{R^2}.(\Sigma^1)^*.\Sigma_{A^2}$, $P_{I^2}(h) = \Sigma_{R^2}.\Sigma_{A^2}$, therefore, $P_{I^2}(s) = (\Sigma_{R^2}.\Sigma_{A^2})^*$. This, therefore, implies that $s \in P_{I^2}^{-1}((\Sigma_{R^2}.\Sigma_{A^2})^*)$, hence:

$$s \in (\Sigma^*.\Sigma_{A^2})^*.(\Sigma^1 \cup \Sigma^2)^* \tag{6}$$

Now note that $sh \in \mathcal{H}^1 \cap \mathcal{I}^2$ (by (4)), and that $h'\rho \leq h$ (by (5)). As $\mathcal{H}^1$ and $\mathcal{I}^2$ are closed languages, we can now conclude:

$$sh'\rho \in \mathcal{H}^1 \cap \mathcal{I}^2$$

As $h' \in (\Sigma^1)^*$ (by (5)), we also have $P_{I^2}(sh'\rho) = P_{I^2}(s\rho)$. Therefore, we can apply Proposition 8 Point e, and conclude:

$$s\rho \in \mathcal{I}^2$$

We now have $s \in (\Sigma^*.\Sigma_{A^2})^*.(\Sigma^1 \cup \Sigma^2)^* \cap \mathcal{H}^2 \cap \mathcal{I}^2$ (by (4) and (6)) and $s\rho \in \mathcal{I}^2$ from the previous step. This allows us to apply Point 4 of the serial interface consistency definition and conclude:

$$(\exists l' \in (\Sigma^2)^*)\ sl'\rho \in \mathcal{H}^2 \cap \mathcal{I}^2 \tag{7}$$

As $h' \in (\Sigma^2)^*$ by (5), we can conclude $P_{I^2}(sh'l'\rho) = P_{I^2}(sl'\rho)$ and $P_{H^2}(sh'l'\rho) = P_{H^2}(sl'\rho)$. These facts along with (7) allows us to apply Proposition 8 Point c and Point e, to show that:

$$sh'l'\rho \in \mathcal{H}^2 \cap \mathcal{I}^2 \tag{8}$$

From (4) and (5), we also have that $sh'\rho h''\alpha \in \mathcal{I}^2$. Additionally, since $h'' \in (\Sigma^1)^*$ and $l' \in (\Sigma^2)^*$ we have that $P_{I^2}(sh'\rho h''\alpha) = P_{I^2}(sh'l'\rho h''\alpha) = P_{I^2}(sh'l'\rho\alpha)$. Therefore, we can apply Proposition 8 Point e, and conclude:

$$sh'l'\rho\alpha \in \mathcal{I}^2$$

We also have that $sh'l' \in \mathcal{H}^2 \cap \mathcal{I}^2$ due to (8) and the fact that $\mathcal{H}^2$ and $\mathcal{I}^2$ are closed languages. This along with the above means we can apply Point 5 of the interface consistency properties and conclude:

$$(\exists l'' \in (\Sigma^2)^*)\ sh'l'\rho l''\alpha \in \mathcal{H}^2 \cap \mathcal{I}^2$$

Also since $h'' \in (\Sigma^1)^*$ by (5), we can conclude $P_{I^2}(sh'l'\rho l''\alpha) = P_{I^2}(sh'l'\rho h''l''\alpha)$ and $P_{H^2}(sh'l'\rho l''\alpha) = P_{H^2}(sh'l'\rho h''l''\alpha)$. These facts along with (9) allows us to apply Proposition 8 Point c and Point e, to show that:

$$sh'l'\rho h''l''\alpha \in \mathcal{H}^2 \cap \mathcal{I}^2 \tag{9}$$

We next note that DES I^2 is a command-pair interface by (3).
As $\alpha \in \Sigma_{A^2}$ (by (5)), we can now conclude:
$P_{I^2}(sh'l'\rho h''l''\alpha) \in \Sigma_{I^2}^*.\Sigma_{A^2} \cap \mathcal{L}(I^2)$

$\Rightarrow P_{I^2}(sh'l'\rho h''l''\alpha) \in \mathcal{L}_m(I^2)$ by Point D of the command-pair interface definition.

$$\Rightarrow sh'l'\rho h''l''\alpha \in \mathcal{I}_m^2 \tag{10}$$

From (4) and (5), we have $sh'\rho h''\alpha \in \mathcal{H}^1$. As $l', l'' \in (\Sigma^2)^*$ (by (7) and (9)), we can conclude:

$$P_{H^1}(sh'\rho h''\alpha) = P_{H^1}(sh'l'\rho h''l''\alpha) \tag{11}$$

We can now apply Proposition 8 Point a, and conclude:

$$sh'l'\rho h''l''\alpha \in \mathcal{H}^1 \tag{12}$$

Combining (12) with (9), (10), and (11), we have $sh'l'\rho h''l''\alpha \in \mathcal{H}^1 \cap \mathcal{H}^2 \cap \mathcal{I}_m^2$ and $P_{H^1}(h'l'\rho h''l''\alpha) = P_{H^1}(h'\rho h''\alpha) = h$. We take $u = h'l'\rho h''l''\alpha$, and the proof is complete. $\square$

Proposition 5.18:
Let $\mathcal{E} \subseteq \Sigma^*$. For system Ψ, $\mathcal{C}_{M_k^i}(\mathcal{E})$ is nonempty and closed under arbitrary union. In particular, $\mathcal{C}_{M_k^i}(\mathcal{E})$ contains a (unique) supremal element that we will denote $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$.

Proof.

We will break the proof into three parts: 1) show $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ is nonempty, 2) show $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ is closed under arbitrary union, and 3) show $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ contains a (unique) supremal element.

1) show $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ is nonempty.

Clearly, $\emptyset \subseteq \mathcal{E}$ and the empty set is MIC_k^i with respect to Ψ and is thus in $\mathcal{C}_{M_k^i}(\mathcal{E})$.

2) show $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ is closed under arbitrary union.

Let $\mathcal{Z}_\beta \in \mathcal{C}_{M_k^i}(\mathcal{E})$ for all $\beta \in B$, where B is an index set. Let $\mathcal{Z} = \cup\{\mathcal{Z}_\beta | \beta \in B\}$. Clearly $\mathcal{Z}_\beta \subseteq \mathcal{Z}$ for each $\beta \in B$.

$$\Rightarrow (\forall \beta \in B)\ \overline{\mathcal{Z}_\beta} \subseteq \overline{\mathcal{Z}} \tag{13}$$

since prefix-closure preserves ordering. It is, therefore, sufficient to show that $\mathcal{Z} \in \mathcal{C}_{M_k^i}(\mathcal{E})$.

Clearly, $\mathcal{Z} \subseteq \mathcal{E}$. Hence, all we need to show is that $\mathcal{Z}$ is MIC_k^i with respect to Ψ. This means showing for all $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$, the following conditions are satisfied:

1. $\text{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \text{Elig}_{\overline{\mathcal{Z}} \cap \mathcal{I}_k^i}(s)$
2. $\text{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \subseteq \text{Elig}_{\mathcal{G}_k^i \cap \overline{\mathcal{Z}}}(s),\ \forall j \in J_k^i$
3. $(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$
4. $(\forall \rho \in \Sigma_{R_k^i})\ (\forall \alpha \in \Sigma_{A_k^i})\ s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$
5. $s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z}$

Let

$$s \in \mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{I}_k^i \cap \overline{\mathcal{Z}}. \tag{14}$$

We first note that this gives us that $s \in \overline{\mathcal{Z}}$.

$\Rightarrow (\exists s' \in \Sigma^*)\ ss' \in \mathcal{Z}$

$\Rightarrow (\exists \beta \in B)\ ss' \in \mathcal{Z}_\beta$, by definition of $\mathcal{Z}$.

$\Rightarrow s \in \overline{\mathcal{Z}_\beta}$

Therefore, by (14) we have that $\exists \beta \in B$ for which

$$s \in \mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{I}_k^i \cap \overline{\mathcal{Z}_\beta}. \tag{15}$$

a) We will now show $\mathrm{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\overline{\mathcal{Z}} \cap \mathcal{I}_k^i}(s)$.

It is sufficient to show $(\forall \sigma \in \Sigma_u)\ s\sigma \in \mathcal{G}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \Rightarrow s\sigma \in \overline{\mathcal{Z}} \cap \mathcal{I}_k^i$. Let

$$\sigma \in \Sigma_u. \tag{16}$$

Assume

$$s\sigma \in \mathcal{G}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}. \tag{17}$$

We will now show this implies $s\sigma \in \overline{\mathcal{Z}} \cap \mathcal{I}_k^i$. We immediately have that $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}_\beta}$, $\sigma \in \Sigma_u$, and $s\sigma \in \mathcal{G}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}$, by (15), (16), and (17). As $\mathcal{Z}_\beta \in \mathcal{C}_{M_k^i}(\mathcal{E})$ by definition and is thus MIC_k^i for Ψ, we can conclude that $s\sigma \in \overline{\mathcal{Z}_\beta} \cap \mathcal{I}_k^i$.

$\Rightarrow s\sigma \in \overline{\mathcal{Z}} \cap \mathcal{I}_k^i$ by (13), as required.

Part (a) is complete.

b) We will now show $\mathrm{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \subseteq \mathrm{Elig}_{\mathcal{G}_k^i \cap \overline{\mathcal{Z}}}(s),\ j \in J_k^i$.

Let $j \in J_k^i$. It is sufficient to show that $(\forall \alpha \in \Sigma_{A_j^{i+1}})\ s\alpha \in \mathcal{I}_j^{i+1} \Rightarrow s\alpha \in \mathcal{G}_k^i \cap \overline{\mathcal{Z}}$. Let

$$\alpha \in \Sigma_{A_j^{i+1}}. \tag{18}$$

Assume

$$s\alpha \in \mathcal{I}_j^{i+1}. \tag{19}$$

We will now show this implies $s\alpha \in \mathcal{G}_k^i \cap \overline{\mathcal{Z}}$. We immediately have that $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}_\beta}$, $\alpha \in \Sigma_{A_j^{i+1}}$, and $s\alpha \in \mathcal{I}_j^{i+1}$, by (15), (18), and (19). As $\mathcal{Z}_\beta \in \mathcal{C}_{M_k^i}(\mathcal{E})$ by definition and is thus MIC_k^i for Ψ, we can conclude that $s\alpha \in \mathcal{G}_k^i \cap \overline{\mathcal{Z}_\beta}$.

$\Rightarrow s\alpha \in \mathcal{G}_k^i \cap \overline{\mathcal{Z}}$ by (13), as required.

Part (b) is complete.

c) We will now show $(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$.

Let

$$s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^* \text{ and } \rho \in \Sigma_{R_k^i}. \tag{20}$$

Assume

$$s\rho \in \mathcal{I}_k^i. \tag{21}$$

We will now show this implies that $(\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$. We immediately have that $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}_\beta}$, $s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*$, $\rho \in \Sigma_{R_k^i}$, and $s\rho \in \mathcal{I}_k^i$, by (15), (20), and (21). As $\mathcal{Z}_\beta \in \mathcal{C}_{M_k^i}(\mathcal{E})$ by definition and is thus MIC_k^i for Ψ, we can conclude that $(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}_\beta}$

$\Rightarrow s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}}$ by (13), as required.

Part (c) is complete.

d) We will now show that $(\forall \rho \in \Sigma_{R_k^i})\ (\forall \alpha \in \Sigma_{A_k^i})\ s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}} \cap \mathcal{I}_k^i$.

Let

$$\rho \in \Sigma_{R_k^i}, \alpha \in \Sigma_{A_k^i}. \tag{22}$$

Assume

$$s\rho\alpha \in \mathcal{I}_k^i. \tag{23}$$

We will now show that this implies $(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}} \cap \mathcal{I}_k^i$. We immediately have that $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}_\beta}$, $\rho \in \Sigma_{R_k^i}$, $\alpha \in \Sigma_{A_k^i}$, and $s\rho\alpha \in \mathcal{I}_k^i$, by (15), (22), and (23). As $\mathcal{Z}_\beta \in \mathcal{C}_{M_k^i}(\mathcal{E})$ by definition and is thus MIC_k^i for Ψ, we can conclude that $(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}_\beta} \cap \mathcal{I}_k^i$.

$\Rightarrow s\rho l\alpha \in \mathcal{G}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}} \cap \mathcal{I}_k^i$ by (13), as required.

Part (d) is complete.

e) We will now show that $s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z} \cap \mathcal{I}_{m_k}^i$.

Assume

$$s \in \mathcal{I}_{m_k}^i. \tag{24}$$

We will now show this implies $(\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z} \cap \mathcal{I}_{m_k}^i$. We immediately have that $s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \overline{\mathcal{Z}_\beta}$ and $s \in \mathcal{I}_{m_k}^i$ by (15) and (24). As $\mathcal{Z}_\beta \in \mathcal{C}_{M_k^i}(\mathcal{E})$ by definition and is thus MIC_k^i for Ψ, we can conclude that $(\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z}_\beta \cap \mathcal{I}_{m_k}^i$.

$\Rightarrow sl \in \mathcal{G}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{Z} \cap \mathcal{I}_{m_k}^i$ by (13), as required.

Part (e) is complete.

From Parts (a), (b), (c), (d), and (e), we can conclude that $\mathcal{Z}$ is MIC_k^i with respect to the system Ψ. We can thus conclude $\mathcal{Z} \in \mathcal{C}_{M_k^i}(\mathcal{E})$, as required.

Part 2 is complete.

3) Show that $\mathcal{C}_{M_k^i}(\mathcal{E})$ contains a (unique) supremal element.

It is sufficient to show that a supremal element exists, as uniqueness would thus follow. Let $\sup \mathcal{C}_{M_k^i}(\mathcal{E}) = \cup\{\mathcal{Z} | \mathcal{Z} \in \mathcal{C}_{M_k^i}(\mathcal{E})\}$.

Claim: $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ is the supremal element.

From Part 2, we have that $\sup \mathcal{C}_{M_k^i}(\mathcal{E}) \in \mathcal{C}_{M_k^i}(\mathcal{E})$. Clearly, $(\forall \mathcal{Z} \in \mathcal{C}_{M_k^i}(\mathcal{E}))\ \mathcal{Z} \subseteq \sup \mathcal{C}_{M_k^i}(\mathcal{E})$, thus $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ is an upper bound for $\mathcal{C}_{M_k^i}$. All that remains to be shown is that

$$(\forall \mathcal{Z}' \in \mathcal{C}_{M_k^i}(\mathcal{E})) \text{ if } \mathcal{Z} \subseteq \mathcal{Z}'\ (\forall \mathcal{Z} \in \mathcal{C}_{M_k^i}(\mathcal{E}))\ \Rightarrow \sup \mathcal{C}_{M_k^i}(\mathcal{E}) \subseteq \mathcal{Z}'.$$

Let $\mathcal{Z}' \in \mathcal{C}_{M_k^i}(\mathcal{E})$. Assume

$$(\forall \mathcal{Z} \in \mathcal{C}_{M_k^i})\ \mathcal{Z} \subseteq \mathcal{Z}'. \tag{25}$$

We must now show this implies $\sup \mathcal{C}_{M_k^i}(\mathcal{E}) \subseteq \mathcal{Z}'$. Let $s \in \sup \mathcal{C}_{M_k^i}(\mathcal{E})$, hence we must show that $s \in \mathcal{Z}'$.

$s \in \sup \mathcal{C}_{M_k^i}(\mathcal{E}) \Rightarrow (\exists \mathcal{Z} \in \mathcal{C}_{M_k^i}(\mathcal{E}))$ such that $s \in \mathcal{Z}$, by definition of $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$.

$\Rightarrow s \in \mathcal{Z}'$, by (25).

We can thus conclude that $\sup \mathcal{C}_{M_k^i}(\mathcal{E})$ is the supremal element.

Part 3 is complete. □

Corollary 5.23:

For system Ψ, if there exists $j \in \{0, 1, 2, \ldots\}$ such that $\Omega^j_{M_k^i}(\mathcal{Z}_k^i)$ is a fixpoint, then the system Φ with $\mathcal{S}^i_{m_k} = \Omega^j_{M_k^i}(\mathcal{Z}_k^i)$ and $\mathcal{S}_k^i = \overline{\mathcal{S}^i_{m_k}}$ satisfies Points 3, 4, 5, and 6 of the multi-level consistency definition, Point ii) of the multi-level controllability definition, and the multi-level nonblocking definition.

Proof.

Assume $\exists j \in \{0,\ 1,\ 2, \ldots\}$ such that $\Omega_{M_k^i}(\Omega^j_{M_k^i}(\mathcal{Z}_k^i)) = \Omega^j_{M_k^i}(\mathcal{Z}_k^i)$. Let $\mathcal{S}^i_{m_k} = \Omega^j_{M_k^i}(\mathcal{Z}_k^i)$ and $\mathcal{S}_k^i = \overline{\mathcal{S}^i_{m_k}}$.

$$\text{By Theorem 5.22, } \mathcal{S}^i_{m_k} = \sup \mathcal{C}_{M_k^i}(\mathcal{Z}^i_{m_k}) \text{ is } MIC_k^i \text{ with respect to } \Psi. \tag{26}$$

By Definition 5.17 and using the fact that $\mathcal{S}_k^i = \overline{\mathcal{S}^i_{m_k}}$, we have for all

$$s \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{S}_k^i : \tag{27}$$

1. $\mathrm{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \subseteq \mathrm{Elig}_{\mathcal{S}_k^i \cap \mathcal{I}_k^i}(s)$

2. $\mathrm{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \subseteq \mathrm{Elig}_{\mathcal{G}_k^i \cap \mathcal{S}_k^i}(s),\ \forall j \in J_k^i$

3. $(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{S}_k^i$

4. $(\forall \rho \in \Sigma_{R_k^i})\ (\forall \alpha \in \Sigma_{A_k^i})\ s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{S}_k^i$

5. $s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{S}_{m_k}^i$

We immediately note that Point II of the level-wise controllability definition follows immediately from Point 1 of (27).

We can then use the fact that $\mathcal{H}_k^i = \mathcal{G}_k^i \cap \mathcal{S}_k^i$ and $\mathcal{H}_{m_k}^i = \mathcal{G}_{m_k}^i \cap \mathcal{S}_{m_k}^i$ to rewrite Points 2-5 of (27) for all $s \in \mathcal{H}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{I}_k^i$: (28)

1. $\text{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \subseteq \text{Elig}_{\mathcal{H}_k^i}(s),\ \forall j \in J_k^i$

2. $(\forall s \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ s\rho \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1}$

3. $(\forall \rho \in \Sigma_{R_k^i})\ (\forall \alpha \in \Sigma_{A_k^i})\ s\rho\alpha \in \mathcal{I}_k^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \in \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1}$

4. $s \in \mathcal{I}_{m_k}^i \Rightarrow (\exists l \in \Sigma_{L_k^i}^*)\ sl \in \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1}$

We now note that Points 3, 4, 5, and 6 of the interface consistency definition follow immediately from (28). All that remains is to show the level-wise nonblocking definition is satisfied. This means showing that $\overline{\mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1}} = \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_j^{i+1}$. By (26), we have $\mathcal{S}_{m_k}^i = \sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i)$.

$$\Rightarrow \mathcal{S}_{m_k}^i \subseteq \mathcal{Z}_{m_k}^i,\ \text{since}\ \sup \mathcal{C}_{M_k^i}(\mathcal{Z}_{m_k}^i) \subseteq \mathcal{Z}_{m_k}^i\ \text{by definition.} \tag{29}$$

$\Rightarrow \mathcal{S}_{m_k}^i \subseteq \mathcal{Z}_k^i$ as $\mathcal{Z}_{m_k}^i \subseteq \mathcal{Z}_k^i$.
$\Rightarrow \overline{\mathcal{S}_{m_k}^i} \subseteq \mathcal{Z}_k^i$, as $\mathcal{Z}_k^i$ is closed and prefix-closure preserves ordering.

$$\Rightarrow \mathcal{S}_k^i \subseteq \mathcal{Z}_k^i,\ \text{by definition of}\ \mathcal{S}_k^i. \tag{30}$$

Substituting for $\mathcal{Z}_{m_k}^i$ in (29), we get $\mathcal{S}_{m_k}^i \subseteq \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j\in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{E}_{m_k}^i$. (31)

$$\text{Substituting for } \mathcal{Z}_k^i \text{ in (30), we get } \mathcal{S}_k^i \subseteq \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{E}_k^i. \tag{32}$$

Using the fact that $\mathcal{H}_{m_k}^i = \mathcal{G}_{m_k}^i \cap \mathcal{S}_{m_k}^i$, we get $\mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1} = \mathcal{G}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1} \cap \mathcal{S}_{m_k}^i$.

$$\Rightarrow \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1} = \mathcal{S}_{m_k}^i, \text{ by (31).} \tag{33}$$

Using the fact that $\mathcal{H}_k^i = \mathcal{G}_k^i \cap \mathcal{S}_k^i$, we get $\mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} = \mathcal{G}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} \cap \mathcal{S}_k^i$.

$$\Rightarrow \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1} = \mathcal{S}_k^i, \text{ by (32).} \tag{34}$$

As $\mathcal{S}_k^i = \overline{\mathcal{S}_{m_k}^i}$ by definition, if follows from (33) and (34) that $\overline{\mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}} = \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}$, as required. □

Proposition 5.25:
For system Φ, Let $H_k^{i\prime} = (Q_k^i, \Sigma, \delta_k^i, q_{0_k}^i, Q_{m_k}^i)$. It thus follows that for all $s, t \in \mathcal{L}(H_k^{i\prime})$, if $\delta_k^i(q_{0_k}^i, s) = \delta_k^i(q_{0_k}^i, t)$ then

1. $\text{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(s) \cap \Sigma_u \nsubseteq \text{Elig}_{\mathcal{S}_k^i \cap \mathcal{I}_k^i}(s) \Leftrightarrow \text{Elig}_{\mathcal{G}_k^i \cap \bigcap \mathcal{I}_j^{i+1}}(t) \cap \Sigma_u \nsubseteq \text{Elig}_{\mathcal{S}_k^i \cap \mathcal{I}_k^i}(t)$
2. $\text{Elig}_{\mathcal{I}_j^{i+1}}(s) \cap \Sigma_{A_j^{i+1}} \nsubseteq \text{Elig}_{\mathcal{H}_k^i}(s) \Leftrightarrow \text{Elig}_{\mathcal{I}_j^{i+1}}(t) \cap \Sigma_{A_j^{i+1}} \nsubseteq \text{Elig}_{\mathcal{H}_k^i}(t),\ \forall j \in J_k^i$
3. $(\forall s, t \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ [s\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Leftrightarrow [t\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$
4. $(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i})\ [s\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Leftrightarrow [t\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ t\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$
5. $[s \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}] \Leftrightarrow [t \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}]$

Before we present the proof of Proposition 5.25, we need to state two results from [7]. In the following, the notation $s \equiv_L t$ means that s is *nerode equivalent* to t on the language L. In other words, s and t have the same continuations in the language L.

Proposition 14: [7]
Let DES $G_i = (Q_i, \Sigma, \delta_i, q_{0i}, Q_{mi})$ $(i = 1, 2)$ and $G = G_1 \| G_2 \| \ldots \| G_n = (Q, \Sigma, \delta, q_0, Q_m)$. It then follows that

$$(\forall i \in \{1, 2, \ldots, n\})(\forall s, t \in \Sigma^*)\ \delta(q_0, s) = \delta(q_0, t) \Rightarrow s \equiv_{\mathcal{G}_i} t \wedge s \equiv_{\mathcal{G}_{m_i}} t$$

Proposition 15: [7]
Let $G = G_1 \| G_2 \| \ldots \| G_n = (Q, \Sigma, \delta, q_0, Q_m)$, $\Sigma_a \subseteq \Sigma$, and I_1 and I_2 be nonempty index sets for our n DES. It thus follows that for all $s, t \in \Sigma^*$, if $\delta(q_0, s) = \delta(q_0, t)$ then

$$\mathrm{Elig}_{\mathcal{G}_{I_1}}(s) \cap \Sigma_a \not\subseteq \mathrm{Elig}_{\mathcal{G}_{I_2}}(s) \Leftrightarrow \mathrm{Elig}_{\mathcal{G}_{I_1}}(t) \cap \Sigma_a \not\subseteq \mathrm{Elig}_{\mathcal{G}_{I_2}}(t)$$

We now present the proof of Proposition 5.25.

Proof.

Let $s, t \in \Sigma^*$. Assume

$$\delta(q_0, s) = \delta(q_0, t) \tag{35}$$

Using (35) we can apply Proposition 14 of [7] and conclude:

$$\begin{aligned} s \equiv_{\mathcal{G}^i_k} t \quad &\wedge \quad s \equiv_{\mathcal{G}^i_{m_k}} t \\ s \equiv_{\mathcal{H}^i_k} t \quad &\wedge \quad s \equiv_{\mathcal{H}^i_{m_k}} t \\ s \equiv_{\mathcal{I}^i_k} t \quad &\wedge \quad s \equiv_{\mathcal{I}^i_{m_k}} t \\ s \equiv_{\mathcal{I}^{i+1}_j} t \quad &\wedge \quad s \equiv_{\mathcal{I}^{i+1}_{m_j}} t \end{aligned} \tag{36}$$

1. Show $\mathrm{Elig}_{\mathcal{G}^i_k \cap \bigcap \mathcal{I}^{i+1}_j}(s) \cap \Sigma_u \not\subseteq \mathrm{Elig}_{\mathcal{S}^i_k \cap \mathcal{I}^i_k}(s) \Leftrightarrow \mathrm{Elig}_{\mathcal{G}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j}(t) \cap \Sigma_u \not\subseteq \mathrm{Elig}_{\mathcal{S}^i_k \cap \mathcal{I}^i_k}(t)$.
   This follows from Proposition 15 of [7] when we take $\Sigma_a = \Sigma_u$, set index I_1 to represent $\mathcal{G}^i_k \cap \bigcap_{j \in J^i_k} \mathcal{I}^{i+1}_j$, and set index I_2 to represent $\mathcal{S}^i_k \cap \mathcal{I}^i_k$.

2. Show $\mathrm{Elig}_{\mathcal{I}^{i+1}_j}(s) \cap \Sigma_{A^{i+1}_j} \not\subseteq \mathrm{Elig}_{\mathcal{H}^i_k}(s) \Leftrightarrow \mathrm{Elig}_{\mathcal{I}^{i+1}_j}(t) \cap \Sigma_{A^{i+1}_j} \not\subseteq \mathrm{Elig}_{\mathcal{H}^i_k}(t)$, $\forall j \in J^i_k$.
   Let $j \in J^i_k$. The desired result follows from Proposition 15 of [7] when we take $\Sigma_a = \Sigma_{A^{i+1}_j}$, set index I_1 to represent $\mathcal{I}^{i+1}_j$, and set index I_2 to represent $\mathcal{H}^i_k$.

3. Show $(\forall s, t \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)(\forall \rho \in \Sigma_{R_k^i})\ [s\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Leftrightarrow [t\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$.

   Let $s, t \in (\Sigma^*.\Sigma_{A_k^i})^*.(\Sigma_{L_k^i})^*)$ and $\rho \in \Sigma_{R_k^i}$. We first note that as s and t are arbitrary and the condition to be proved is symmetric, it is, therefore, sufficient to prove $[s\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Rightarrow [t\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$. Assume $[s\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$.

   From the above and (36), we can now conclude that $[t\rho \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl\rho \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$ as desired.

4. Show $(\forall \rho \in \Sigma_{R_k^i})(\forall \alpha \in \Sigma_{A_k^i})\ [s\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Leftrightarrow [t\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ t\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$.

   Let $\rho \in \Sigma_{R_k^i}$ and $\alpha \in \Sigma_{A_k^i}$. We first note that as s and t are arbitrary and the condition to be proved is symmetric, it is, therefore, sufficient to prove $[s\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}] \Rightarrow [t\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ t\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$. Assume $[s\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ s\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$.

   Based on the above and (36), we can conclude $[t\rho\alpha \in \mathcal{I}_k^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ t\rho l\alpha \notin \mathcal{H}_k^i \cap \mathcal{I}_k^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_j^{i+1}]$ as desired.

5. Show $[s \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}] \Leftrightarrow [t \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}]$.

   We first note that as s and t are arbitrary and the condition to be proved is symmetric, it is, therefore, sufficient to prove $[s \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}] \Rightarrow [t \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}]$. Assume $[s \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ sl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}]$.

   Based on the above and (36), we can thus conclude $[t \in \mathcal{I}_{m_k}^i] \wedge [(\exists l \in \Sigma_{L_k^i}^*)\ tl \notin \mathcal{H}_{m_k}^i \cap \mathcal{I}_{m_k}^i \cap \bigcap_{j \in J_k^i} \mathcal{I}_{m_j}^{i+1}]$ as desired.

$\square$

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Supremica. Available: http://www.supremica.org.

[2] B. A. Brandin, R. Malik, and P. Malik. Incremental verification and synthesis of discrete-event systems guided by counter examples. *IEEE Transactions on Control Systems Technology*, 12(3):387–401, May 2004.

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

[4] P. E. Caines and Y. J. Wei. The hierarchical lattices of a finite state machine. *Systems Control Letters*, 25:257–263, July 1995.

[5] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, Boston, MA, 1999.

[6] O. Contant, S. Lafortune, and D. Teneketzis. Diagnosability of discrete event systems with modular structure. *Discrete Event Dynamic Systems: Theory and Applications*, 16:9–37, 2006.

[7] P. Dai. Synthesis method for hierarchical interface-based supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Canada, 2006.

[8] M. H. de Queiroz and J. E. R. Cury. Modular supervisory control of composed systems. In *Proc. American Control Conf.*, pages 4051–4055, Chicago, USA, 2000.

[9] M.H. de Queiroz, J.E.R. Cury, and W.M. Wonham. Multitasking supervisory control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 15:375–395, 2005.

[10] R. Debouk. Diagnosis of discrete event systems: A modular approach. In *Proc. IEEE Int Conf. Systems, Man, and Cybernetics*, pages 306–311, 2003.

[11] E.W. Endsley and D.M. Tilbury. Modular verification of modular finite state machines. In *Proc. 43rd IEEE Conf. Decision & Control*, pages 972–979, The Bahamas, 2004.

[12] J.M. Eyzell and J.E.R. Cury. Exploiting symmetry in the synthesis of supervisors for discrete event systems. *IEEE Trans. Automat. Contr.*, 46(9):1500–1505, 2001.

[13] M. Fabian and B. Lennartson. On non-deterministic supervisory control. In *Proc. 35th IEEE Conf. Decision & Control*, 1996.

[14] E. Fabre and A. Benveniste. Partial order techniques for distributed discrete event systems: Why you can't avoid using them. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 1–2, Ann Arbor, USA, 2006.

[15] L. Feng. On the computation of natural observers in discrete-event systems. Technical report, University of Toronto, Systems and Control Group, Toronto, Canada, January 2006.

[16] L. Feng. *Computationally Efficient Supervisor Design in Discrete-Event Systems.* PhD thesis, University of Toronto, Toronto, Canada, 2007.

[17] L. Feng and W.M. Wonham. Computationally efficient supervisor design: Abstraction and modularity. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 3–8, Ann Arbor, USA, 2006.

[18] L. Feng and W.M. Wonham. Computationally efficient supervisor design: Control flow decomposition. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 9–14, Ann Arbor, USA, 2006.

[19] H. Flordal. *Compositional Approaches in Supervisory Control.* PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 2006.

[20] H. Flordal, M. Fabian, K. Akesson, and A. Hellgren. Controllability revisited: A generalization for the modular approach. In *Proceedings of the 11th IFAC Symposium of Information Control Problems in Manufacturing*, Salvador, Brazil, 2004.

[21] H. Flordal and R. Malik. Modular nonblocking verification using conflict equivalence. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 100–106, Ann Arbor, USA, 2006.

[22] H. Flordal and R. Malik. Supervision equivalence [supervisor synthesis]. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 155–160, Ann Arbor, USA, 2006.

[23] B. Gaudin and H. Marchand. Modular supervisory control of a class of concurrent discrete event systems. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 181–186, Reims, France, 2004.

[24] M. Heymann and F. Lin. Nonblocking supervisory control of nondeterministic systems. Technical Report CIS-9620, Technion, Israel Institute of Technology, Haifa, Israel, October 1996.

[25] R.C. Hill, J.E.R. Cury, M.H. de Queiroz, and D.M. Tilbury. Modular requirements for hierarchical interface-based supervisory control with multiple levels. In *Proc. American Control Conf.*, Seattle, USA, 2008.

[26] R.C. Hill and D.M. Tilbury. Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 399–406, Ann Arbor, USA, 2006.

[27] R.C. Hill and D.M. Tilbury. Incremental hierarchical construction of modular supervisors for discrete-event systems. *to appear in the International Journal of Control*, 2008.

[28] R.C. Hill, D.M. Tilbury, and S. Lafortune. Modular supervisory control with equivalence-based conflict resolution. In *Proc. American Control Conf.*, Seattle, USA, 2008.

[29] L.E. Holloway and B.H. Krogh. Synthesis of feedback logic control for a class of controlled petri nets. *IEEE Trans. Automat. Contr.*, 35(5):514–523, 1990.

[30] P. Hubbard and P. E. Caines. A state aggregation approach to hierarchical supervisory control with applications to a transfer line example. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, Cagliari, Italy, 1998.

[31] K. Inan. Supervisory control: Theory and application to the gateway synthesis problem. In *Belgian-French-Netherlands Summer School on Discrete Event Systems*, Spa, Belgium, 1993.

[32] J. Komenda and J. van Schuppen. Optimal solutions of modular supervisory control problems with indecomposable specification languages. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 143–148, Ann Arbor, USA, 2006.

[33] J. Komenda, J. van Schuppen, B. Gaudin, and H. Marchand. Modular supervisory control with general indecomposible specification languages. In *Proc. 44th IEEE Conf. Decision & Control and European Control Conf.*, pages 3474–3479, Sevilla, Spain, 2005.

[34] R. Kumar, V.K. Garg, and S.I. Marcus. Predicates and predicate transformers for supervisory control of discrete event dynamical systems. *IEEE Trans. Automat. Contr.*, 38:232–247, 1993.

[35] R. Kumar, S. Jiang, C. Zhou, and W. Qiu. Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control. *IEEE Trans. Automat. Contr.*, 50(4):463–475, April 2005.

[36] R. Kumar and M.A. Shayman. Non-blocking supervisory control of nondeterministic discrete-event systems via prioritized synchronoziation. *IEEE Trans. Automat. Contr.*, 41(8):1160–1175, August 1996.

[37] R.J. Leduc. *Hierarchical Interface-based Supervisory Control.* PhD thesis, University of Toronto, Toronto, Canada, 2002.

[38] R.J. Leduc, B.A. Brandin, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control–part I: Serial case. *IEEE Trans. Automat. Contr.*, 50(9):1322–1335, 2005.

[39] R.J. Leduc and P. Dai. Synthesis method for hierarchical interface-based supervisory control. In *Proc. American Control Conf.*, pages 4260–4267, New York, USA, 2007.

[40] R.J. Leduc, M. Lawford, and P. Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Transactions on Control Systems Technology*, 14(4), 2006.

[41] R.J. Leduc, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control–part II: Parallel case. *IEEE Trans. Automat. Contr.*, 50(9):1336–1348, 2005.

[42] S.H. Lee and K.C. Wong. Decentralised control of concurrent discrete-event systems with non-prefix closed local specifications. In *Proc. 36th IEEE Conf. Decision & Control*, pages 2958–2963, San Diego, USA, 1997.

[43] Y. Li. *Control of Vector Discrete-Event Systems.* PhD thesis, University of Toronto, Toronto, Canada, 1991.

[44] Y. Li and W.M. Wonham. Control of vector discrete event systems-part I: The base model. *IEEE Trans. Automat. Contr.*, 38(8):1215–1227, 1993.

[45] F. Lin and W.M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[46] C. Ma. *Nonblocking Supervisory Control of State Tree Structures.* PhD thesis, University of Toronto, Toronto, Canada, 2004.

[47] P. Madhusudan and P.S. Thiagarajan. Branching time controlers for discrete event systems. *Theoretical Computer Science*, 274:117–149, 2002.

[48] P. Malik, R. Malik, D. Streader, and S. Reeves. Modular synthesis of discrete controllers. In *Proc. 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS'07)*, 2007.

[49] R. Malik, H. Flordal, and P. Pena. Conflicts and projections. In *Proc. 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07)*, 2007.

[50] R. Malik, D. Streader, and S. Reeves. Conflicts and fair testing. *International Journal of Foundations of Computer Science*, 17(4):797–813, 2006.

[51] R. Milner. *Communication and Concurrency.* Prentice-Hall, Inc, London, 1989.

[52] A. Overkamp. Supervisory control using failure semantics and partial specification. *IEEE Trans. Automat. Contr.*, 42:498–510, April 1997.

[53] S.J. Park and J.T. Lim. Nonblocking supervisory control of nondeterministic systems based on multiple deterministic model approach. *IEICE Trans. Inf. & Syst.*, E83-D(5):1177–1180, May 2000.

[54] P. Pena, J.E.R. Cury, and S. Lafortune. Testing modularity of local supervisors: An approach based on abstractions. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 107–112, Ann Arbor, USA, 2006.

[55] H. Qin and P. Lewis. Factorization of finite state machines under strong and observational equivalences. *Formal Aspects of Computing*, 3:284–307, 1991.

[56] P.J. Ramadge and W.M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal of Control and Optimization*, 25(5):1202–1218, 1987.

[57] P.J. Ramadge and W.M. Wonham. Modular supervisory control of discrete event systems. *Mathematics of Control, Signal and Systems*, 1:13–30, 1988.

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

[59] K. Rohloff and S. Lafortune. The verification and control of interacting similar discrete-event systems. *SIAM Journal on Control and Optimization*, 45(2):634–667, June 2006.

[60] K. Schmidt, T. Moor, and S. Perk. A hierarchical architecture for nonblocking control of discrete event systems. In *Mediterranean Conf. Control and Automation*, pages 902–907, Limassol, Cyprus, 2005.

[61] K. Schmidt, J. Reger, and T. Moor. Hierarchical control of structural decentralized des. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, 2004.

[62] R. Song and R.J. Leduc. Symbolic synthesis and verification of hierarchical interface-based supervisory control. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 419–426, Ann Arbor, USA, 2006.

[63] R. Su. *Distributed Diagnosis for Discrete-Event Systems.* PhD thesis, University of Toronto, Toronto, Canada, 2004.

[64] R. Su and J. Thistle. A distributed supervisor synthesis approach based on weak bisimulation. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 64–69, Ann Arbor, USA, 2006.

[65] R. Su and W.M. Wonham. Distributed diagnosis under global consistency. In *Proc. 43rd IEEE Conf. Decision & Control*, pages 525–530, The Bahamas, 2004.

[66] R. Su and W.M. Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 14:31–53, 2004.

[67] P. Tabuada. Open maps, alternating simulations and control synthesis. In *International Conference on Concurrency Theory*, pages 466–480, 2004.

[68] S. Takai and S. Kodama. M-controllable subpredicates arising in state feedback control of discrete event systems. *Int. J. of Control*, 67(4):553–566, 1997.

[69] S. Takai and S. Kodama. Characterization of all m-controllable subpredicates of a given predicate. *Int. J. of Control*, 70(4):541–549, 1998.

[70] S. Takai, T. Ushio, and S. Kodama. Static-state feedback control of discrete-event systems under partial observation. *IEEE Trans. Automat. Contr.*, 40(11):1950–1954, November 1995.

[71] W. Wang, S. Lafortune, and F. Lin. An algorithm for calculating indistinguishible states and clusters in finite state automata with partially observable transitions. *Systems Control Letters*, 56:656–661, 2007.

[72] K.C. Wong. On the complexity of projections of discrete event systems. In *Proc. Int. Workshop on Discrete Event Systems (WODES)*, pages 201–206, Cagliari, Italy, 1998.

[73] K.C. Wong, J.G. Thistle, H.-H. Hoang, and R.P. Malhamé. Conflict resolution in modular control with applications to feature interaction. In *Proc. IEEE Conf. on Decision & Control*, pages 416–421, New Orleans, USA, 1995.

[74] K.C. Wong, J.G. Thistle, R.P. Malhame, and H.H. Hoang. Supervisory control of distributed systems: Conflict resolution. *Discrete Event Dynamic Systems: Theory and Applications*, 10:131–186, 2000.

[75] K.C. Wong and W.M. Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6:241–273, 1996.

[76] K.C. Wong and W.M. Wonham. Modular control and coordination of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 8:247–297, 1998.

[77] W.M. Wonham. *Supervisory Control of Discrete-Event Systems.* ECE Dept., University of Toronto. current update 2006.07.01, available at http://www.control.utoronto.ca/DES.

[78] W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *Siam J. of Control Optim.*, 25(3):637–659, 1987.

[79] S.H. Zad, R.H. Kwong, and W.M. Wonham. Fault diagnosis in discrete-event systems: Framework and model reduction. *IEEE Trans. Automat. Contr.*, 48(7):1199–1211, July 2003.

[80] H. Zhong and W.M. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Trans. Automat. Contr.*, 35(10):1125–1134, October 1990.

[81] C. Zhou, R. Kumar, and S. Jiang. Control of nondeterministic discrete-event systems for bisimulation equivalence. *IEEE Trans. Automat. Contr.*, 51(5):754–765, May 2006.