Hierarchical Interface-Based Decentralized Supervisory Control

HIERARCHICAL INTERFACE-BASED DECENTRALIZED SUPERVISORY CONTROL

BY

HUAILIANG LIU, M.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING & SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

PH.D. IN COMPUTER SCIENCE

Ph.D. in Computer Science (2015) McMaster University

(Computing & Software) Hamilton, Ontario, Canada

TITLE:	Hierarchical Interface-Based Decentralized Supervisory Control
AUTHOR:	Huailiang Liu M.Eng., (Computer Engineering) Guangdong University of Technology, Guangzhou, China
SUPERVISOR:	Dr. Ryan J. Leduc, Dr. S. L. Ricker
NUMBER OF PAGES:	xi, 130

To My Father and Mother

In those years, it could not be imagined how hard it was to survive, but you still held extremely strong will to raise the family and sacrifice everything to give children education opportunities!

Abstract

In decentralized control, agents have only a partial view and partial control of the system and must cooperate to achieve the control objective. In order to synthesize a decentralized control solution, a specification must satisfy the co-observability property. Existing co-observability verification methods require the possibly intractable construction of the complete system. To address this issue, we introduce an incremental verification of co-observability approach. Selected subgroups of the system are evaluated individually, until verification is complete. The new method is potentially much more efficient than the monolithic approaches, in particular for systems composed of many subsystems, allowing for some intractable problems to be manageable. Properties of this new strategy are presented, along with a corresponding algorithm and an example.

To further increase the scalability of decentralized control, we wish to adapt the existing Hierarchical Interface-Based Supervisory Control (HISC) to support it. We introduce the Hierarchical Interface-Based Decentralized Supervisory Control (HIDSC) framework that extends HISC to decentralized control. To adapt co-observability for HIDSC, we propose a per-component definition of co-observability along with a verification strategy that requires only a single component at a time in order to verify co-observability. Finally, we provide and prove the necessary and sufficient conditions

for supervisory control existence in the HIDSC framework and illustrate our approach with an example. As the entire system model never needs to be constructed, HIDSC potentially provides significant savings.

Acknowledgements

A bunch of thanks should first be given to my supervisors Dr. Leduc and Dr. Ricker for giving me insightful direction on my research and numerous good advice on improving my Ph.D. study. Bi-weekly meetings and frequently email communications with my supervisors give me plenty of fresh ideas and practical guide on how to do my Ph.D. work.

Special thanks for Dr. Malik for giving me excellent suggestions for improving my work on Incremental Verification of Co-observability.

A lot of thanks should be given to Dr. Down and Dr. Kahl for their guide in my supervisor committee member meetings.

Thanks for Dr. Ricker and her student Micah for the excellent work of helping to implement the software for the thesis.

Very importantly, I should thank the scholarship providers: my supervisors, the School of Graduate Studies and the Department of Computing & Software at McMaster University, and the Ontario Graduate Scholarship (OGS), which provide the basis of my living for the four years of my Ph.D. study.

Last but not least, I need to thank my sister's support for my secondary study, my wife for doing so much work at home and giving me enough time to complete my Ph.D. study, and my daughter of giving me such happy family life.

Contents

List of Figures

Chapter 1

Introduction

In control theory, the dynamic behaviour of continuous-state systems is typically modeled by differential or difference equations. Over the past few decades, the rapid evolution of computing, communication, and sensor technologies has brought about a new type of dynamic systems called discrete-event systems (DES). These systems are often highly complex resulting in system models with large state spaces.

Formally, a DES can be thought of as a dynamic system, namely an entity equipped with a state space and a state-transition structure. In particular, a DES is discrete in time and in state space. It is asynchronous or event-driven which means that it is driven by events other than, or in addition to, the tick of a clock. DES are also nondeterministic in the sense that it is capable of transitional 'choices' by internal chance or other mechanisms not necessarily modeled by the system analyst [Won14].

Some typical examples of DES are flexible manufacturing systems; communication protocols; semiconductor chip design; real-time systems; computer networks;

automated manufacturing systems; traffic control systems; highly integrated command, control, communication, and information systems; advanced monitoring and control systems in automobiles or large buildings; intelligent transportation systems; distributed software systems; multi-agent systems and logistic (service) systems.

In supervisory control of DES, a discrete-event system is modeled as a set of sequences of events that describes the behaviour of a physical process. A discrete-event control problem arises when we want to restrict the system to performing a specified subset of the overall system behaviour. A solution to a discrete-event control problem exists when we can construct a controller (also called supervisors, or agents) to achieve the desirable (or legal) behaviour by either preventing some events from taking place (disabling a controllable event) or allowing—but not forcing—others to occur (enabling an event). The two key properties we wish to evaluate for a discrete-event system are controllability and nonblocking. Controllability tests to see if we can correctly enforce our desired behaviour on a given plant (the uncontrolled system), while nonblocking is a simple form of deadlock checking.

1.1 Motivation

Decentralized discrete-event control problems arise naturally through the investigation of a large variety of distributed systems such as communication networks, integrated sensor networks, networked control systems, and automated guided vehicular system. These systems have many controllers that jointly control a given system that is inherently distributed. In decentralized control, agents have only a partial view and partial control of the system and must cooperate to achieve the control objective. A

decentralized approach is used when the physical system is such that controllers implemented at different locations would naturally only be able to see and effect events occurring in their local vicinity, and have no access to other events. The goal is to be able to implement a set of decentralized controllers that produce the same control actions as a centralized controller with full view and control of the system.

Building the complete state space for the system model may lead to the computation of a very large – possibly computationally intractable – number of states, that grows exponentially and this is often referred to as the state-space explosion problem. The existing approaches for verifying decentralized properties require the construction of the complete system model. As systems requiring control become more complex, computing the complete system may be, at best, an example of the state-space explosion problem, and at worst intractable. To address the decentralized control of interesting systems, it is necessary to improve on the existing monolithic strategies. We first want to examine incremental approaches that allow the verification of properties relevant to the synthesis (construction) of decentralized controllers, without the need to construct the entire system model.

Decentralized problems do not just occur in isolation. In many systems of interest, decentralized problems are embedded into the description of more complicated frameworks. In particular, the control of hierarchical systems has not yet evolved to include decentralized architectures with partial observation. To extend DES theory to be useful to a wider variety of real systems, we are interested in incorporating the power of decentralized control strategies to the existing Hierarchical Interface-Based Supervisory Control (HISC) framework [Led02, Led09]. The HISC approach decomposes a system into a high-level subsystem (component) which communicates with

$n \geq 1$ parallel low-level subsystems through separate interfaces that restrict the interaction of the subsystems. We wish to develop a per-component verification method of decentralized properties using the HISC structure.

1.2 Problem Statement

One of the main obstacles in the control of DES is the combinatorial explosion of the product state space which can increase in size very quickly. In practice, a complex DES is often modeled as the product of a number of simpler components. Each time a new component is added (such as with state space size N), the state size of the whole system is multiplied, worst case, by N; thus, the size of the model increases exponentially with the number of components.

To effectively address the control of systems composed of many smaller subsystems, existing monolithic strategies must be replaced with those that consider computation at the subsystem level. In many cases, the classic approach succumbs to the state-space explosion problem, or worse, is computationally infeasible.

The verification of the property of co-observability [RW92b] is key to the synthesis of decentralized controllers. However, it requires that the complete state space be constructed before the verification algorithm is performed, after which an even larger structure is built. Incremental verification strategies, where verification can be performed on subsystems instead of the entire system, have been successfully applied to centralized DES. We are interested in extending these ideas to co-observability.

Finally, we will incorporate decentralized architectures into a hierarchical framework (HISC). We will keep the HISC structure, but replace the centralized controller

of each component with a set of decentralized controllers. In addition, to avoid computational issues (e.g. intractability), we want to extend the HISC component-based verification strategies to verify the desired decentralized properties. The addition of decentralized controllers into this framework will allow for the application of decentralized supervisory control to more complex systems.

1.3 Objectives

In this work, our goal is to develop a scalable method that can handle the combinatorial explosion of the product state space in decentralized supervisory control with partial observations.

- We are interested in applying an incremental strategy to the verification of co-observability, one of the properties that must be satisfied for synthesizing solutions to decentralized supervisory control problems. Similar strategies have been introduced in the verification of centralized DES properties such as controllability, and have been shown to be successful in reducing the state space that must be computed to check the property in question. Such a strategy has particular interest for the study of decentralized architectures, where the uncontrolled system is often composed of many subsystems. In many cases, the construction of the complete system model, which is required for the current method for verifying co-observability, is simply intractable.

- We introduce decentralized control to the Hierarchical Interface-Based Supervisory Control (HISC) framework as this framework does not support decentralized control but has been shown to be very scalable. The new framework is

called Hierarchical Interface-Based Decentralized Supervisory Control (HIDSC). This requires the identification of the necessary and sufficient conditions for the existence of decentralized controllers in HIDSC. As co-observability must still be verified in this new framework, we will introduce a per-component strategy for verifying co-observability in HIDSC, which we expect to be useful in working with large distributed systems.

1.4 Literature Review

In supervisory control of DES, the computation and complexity for many of the control solutions entail only polynomial effort in the model's state size. Nevertheless, some of them need non-polynomial time, for instance, the supervisor design problem in DES is proved to be NP-hard [GW00, AM96, SW04]. Furthermore, the the computation and complexity is worse in the case of control with partial observations: some problem with full observation is polynomial, however it is exponential (rather than polynomial) when the situation is partially observable [Tsi89, RW95, RYL03, Tri04, TL09, YL02b].

On the topic of computational complexity results for decentralized DES control, several papers provide different ways of examining the undecidability of decentralized DES control: [TL09, Tri04, Tsi89, RW95]. In addition, some further results on complexity have been considered [RYL03, YL02b]. The computational complexity to verify co-observability using the monolithic approach is discussed in [RW95].

We will use incremental methods to verify co-observable property to reduce the state-space explosion problem in decentralized supervisory control with partial observations. Currently, incremental methods have been successfully used in verification

of controllability in the work of [BMM04]. Nonblocking verification is addressed by a compositional approach in [PCL06, FM06a, FM09], which construct the global system incrementally using abstraction in order to reduce the complexity of verification. This approach is based on making abstractions, at the modular level, that reduce the state-space explosion without qualitatively changing the system. In [PCL06], natural projection with the observer property introduced by [WW96] that preserves observation equivalence, is used to calculate appropriate abstractions. In contrast, work in [FM06a, FM09] uses a process-algebraic equivalence, namely conflict equivalence [MSR04], in order to obtain better abstraction of nondeterministic models. These works are very useful, but still leave the problem of what to do if conflict is detected.

An approach to create equivalent simpler supervisors from a given monolithic supervisor has been proposed in [SW04]. This work also showed that finding a supervisor of minimal size is NP-hard. The results are very impressive, but the method relies on a monolithic supervisor to be constructed first, and thus remains limited by its size.

Research work in [FM06b, FMFÅ07] employ a different type of abstraction based on supervision equivalence to incrementally construct the monolithic supervisor for a system. This work is not a modular approach to control, but succeeds in producing a very compact representation of the monolithic solution.

The work of [WW98] applies notions of hierarchical and modular control to reduce complexity. Their approach specifically builds modular supervisors then adds another level of control to resolve the conflict between the supervisors. Abstraction is employed to further reduce the complexity. Nonblocking control is achieved by requiring

an observer property of the abstraction and optimality (in the sense of maximally permissive) is achieved by additionally requiring an output control-consistency property. Although the work provides results for general abstractions and control structures, the work is rather theoretically complex and it is very hard to create computational tools to carry out the proposed methods.

The work of [FW06a, FW06b, FW08, FCW09] also employs this kind of abstraction method, but considers the specific case of natural projection with the observer property and a supervisory control structure to overcome the theoretical complexity. Their approaches result in a greater number of relatively simple decentralized supervisors: some are dedicated to the given control specifications, preferably in one-to-one correspondence, while others are dedicated only to resolving conflicts.

Abstraction methods in hierarchical architecture were also proposed in [SM06, SMP08, SB11], but have different objectives. The work in [FW06a, FW06b, FW08, FCW09] is to reduce the computational complexity of designing an optimal and nonblocking supervision using the hierarchical architecture, while the main objective of [SM06, SMP08, SB11] is to implement a given (hierarchical) control by decentralized supervisors.

Similarly, work in [HT06, HT08] have proposed an incremental hierarchical computational approach, which uses control architecture and model abstraction. Their approach does not, however, allow subsystems at the same level to share events. The result is a hierarchy with more levels. Furthermore, the supervisors in [HT06, HT08] may combine control actions for enforcing control specifications with actions that resolve system blocking, and does not guarantee maximally permissive control.

In addition, all the above literature based on their work on the condition of supervisors with full observations, and did not address the condition when supervisors only have partial observations and can only control part of the controllable events. We will consider the more general situation that supervisors have partial observations and can only control part (possibly overlapping) of the controllable events.

The evolution of the centralized architecture with partial observation of one controller (also called supervisor or agents) into a decentralized architecture with partial observation of more than one controller is described as follows.

On the topic of partial observation with multiple controllers who of enable by default and unconditional co-observability, the following papers investigated the synthesis of decentralized controllers under conditions of partial observation: [LW88b, CDFV88, RW92b]. Most of the conditions required for synthesizing decentralized controllers that solve the decentralized DES problem were formulated in [RW92b]. Some follow-up fully decentralized solutions of supervisory control problems are described in [KW95]. Additional examination of state-based control appeared in [TKU94].

These papers all dealt with a scenario where the default decision (when a controller is uncertain of the correct control decision to take) is to enable. In the work of [RW92b], the property of co-observability is introduced, and involves the fusion of the local decisions of the decentralized controllers by performing their conjunction (i.e., disable = 0 and enable = 1). Decision fusion is discussed in [PKK97]. Verification of co-observability is presented in [RW95]. Several variations of the original problem are described in [TU00]. An interesting formulation based on Nash Equilibrium was described by [OvS00]. An update on super/sub-languages for decentralized DES appears in [TKU05].

The state-space explosion problems become a bottleneck for the application of supervisory control of DES. It is surely attractive to refine the approach and explore structured system architecture methods to render large complex DES more manageable. So far, there are two kinds of well known existing architectures to lower the complexity: horizontal and vertical modularity, or so called modular and hierarchical control.

Modular control (under full observation) [WR88, CCdB93, DQC00, SB01] is one of the earliest and useful methods. As opposed to the monolithic control, this method designs multiple supervisors and each supervisor implements a portion of the control specification.

This method led to modular control under partial observation, the form of decentralized control with partial observation addressed in [LW88b, LW90], in which each supervisor can access only partial information and is allowed to disable only a subset of controllable events.

The modular control approaches are able to provide controllability, but do not guarantee nonblocking unless the modular supervisors are shown to be nonconflicting. Unfortunately, verifying nonconflicting property typically is as computationally expensive as building the monolithic DES [WR88, Won14]. Although some of the work in modular control achieved good results for verifying controllability, nonblocking was still not addressed [BMM04, GM04, KvS06, KvSGM05, KvSGM08]. The existing algorithms for modular control in general still require a large computation and complexity. Therefore the interest in hierarchical control was invoked.

Hierarchical supervisory control in DES is an architecture method by exploiting the advantages of vertical modularity. The hierarchical supervisory control was

initially introduced in [ZW90], in which a low-level DES and a high-level aggregate DES were proposed. In this paper, hierarchical consistency was introduced to ensure that the abstract high-level supervisor can actually be implemented in the actual low-level DES, and output-control consistency was employed to guarantee optimality of the behaviour achieved at the low level. Later, this theory was generalized by [WW96], based on the concepts of control structures and observers, and the hierarchical consistency can be achieved from a new condition called control consistency. This kind of hierarchical control is bottom-up in the sense that aggregate models are derived from actual low-level DES to the abstract high-level DES by using either state-based or language-based aggregation methods. There are also some other related work based on this kind of model aggregation methods [CH00, CL01, EC01, HC98, QJ99, SC02, SM06, SMP08, SB11, FW08, FCW09].

Although the model aggregation approach can be effective in constructing high-level models with reduced state spaces, they suffer some drawbacks: First, there is no direct connection between control actions at the high level, and at lower levels. To create an implementation, a control action at the high level may need to be "interpreted" as equivalent control action(s) at the low level. Second, they require that the monolithic system be built before the abstraction is applied. But, due to the state-space explosion problem, the monolithic system may be too large to be built at first or is too computationally expensive. Third, these aggregate models must be constructed sequentially from the bottom up, starting from the lowest level; thus a given level can not be constructed and verified in parallel with the levels below it, making the distributed design process difficult. Fourth, the requirements on the abstractions used may be too strict and will limit the amount of model reduction

that can be achieved. Finally, this approach causes the individual levels to be tightly coupled; a change made to the lowest level may require that all aggregate models and results have to be re-evaluated.

The Hierarchical Interface-Based Supervisory Control (HISC) framework proposed by Leduc *et al.* in [Led02, LBLW05, LLW05, LLD06, LDS09, Led09] does not have the above drawbacks and can alleviate the state-space explosion problem. The HISC approach decomposes a system into a high-level subsystem which communicates with $n \geq 1$ parallel low-level subsystems through separate interfaces that restrict the interaction of the subsystems. It provides a set of local conditions that can be used to verify global conditions such as nonblocking and controllability, such that the complete system model never needs to be constructed. The sufficient conditions of HISC allow the independent design and verification of different levels, ensuring that a change to one level of the hierarchy will not impact the others.

To the best of our knowledge, there is no existing work where HISC supports decentralized discrete-event control architecture. Although there is some literature proposing hierarchical control of decentralized discrete event systems [SM06, SMP08, SB11] these approaches assume full observation. Note that to say decentralized control, partial observation is implied, otherwise the problem is trivially reduced to the centralized problem. Henceforth, when we say decentralized control, we assume partial observation.

1.5 Contributions of the Thesis

Contributions of this thesis are as follows:

- Incremental verification for co-observable property.

Using the above method, checking co-observability is done incrementally by evaluating selected sub-groupings of the system individually until the entire system has been shown to be co-observable. A set of properties for this new strategy are presented and proved, along with a corresponding algorithm and an example. Compared to the traditional monolithic method, the new method should be much more efficient, especially for very large, distributed systems composed of many sub-systems.

- Per-component verification for co-observability property.

The main job here is to verify co-observability (we call it *level-wise co-observability*) for each component. For each low-level component, the component supervisor synchronized with its interface will act as the specification for the component plant. For the high level, the high-level supervisor will be looked on as the representation of the specification, and be verified whether it is co-observable with respect to the high-level plant and the interfaces.

In this approach, we do per-component decentralized control, but achieve the same control decisions as centralized HISC supervisors. Since the system satisfies both HISC and co-observable properties, therefore if the centralized version is nonblocking and controllable, the decentralized version will also be nonblocking and controllable.

- Verification of co-observability for the global system based on per-component verification.

We show that the above level-wise HIDSC co-observability implies that the global system is co-observable. The research result here is to prove the HIDSC co-observability

theorem in this form: if the system is level-wise co-observable, then the global system is co-observable.

The tricky problem here is that in level-wise co-observable verification, we only consider the local specification and local N_i decentralized observers (controllers) for each component i. But for the global system, the specification is the global specification, and the decentralized observers are the total of all the observers $N = N_1 + ... + N_n$, where n is the number of components including the high-level plant component.

Note that the global specification is represented as local supervisors and their interfaces synchronized together, which may be very large. It may not be possible to construct them explicitly, and possibly can only be specified theoretically due to the state-space explosion problem.

- Supervisory control existence theorem in HIDSC.

We provide a supervisory control existence theorem for HIDSC systems, and proved the necessary and sufficient conditions for decentralized control in HIDSC. The new HIDSC approach allows marking non-blocking decentralized supervisory control, which is more expressive compared with traditional methods.

1.6 Outline of the Thesis

The remainder of the thesis is organized as follows. In Chapter 2 we review the relevant definitions and results from supervisory control theory. In Chapter 3 we present a strategy for verifying co-observability in an incremental way, potentially allowing for more straightforward verification of this property for large distributed architectures. We provide the theoretical conditions for the incremental approach, culminating in

the presentation of a new algorithm. Chapter 4 contains results from a case study on verifying co-observability for the data transmission problem. We examine the effect of performance heuristics on the incremental algorithm of Chapter 3, and compare this to the performance of the classical monolithic algorithm for verifying co-observability. Chapter 5 introduces our results on incorporating decentralized decision-making into the HISC framework. We introduce a per-component co-observability definition and show that the new property implies global co-observability. We present a necessary and sufficient condition for solving the control problem within this new framework. In Chapter 6 we examine the results of a case study for the verification of HIDSC modeling of a small manufacturing system. We summarize our results in Chapter 7 and describe areas for further research.

Chapter 2

Preliminaries

This chapter provides a brief review of the key DES concepts used in this thesis. Readers unfamiliar with the notation and definitions may refer to [CL08, Won14].

2.1 Languages

Event sequences and languages are simple ways to describe DES behaviour. Let Σ be a finite set of distinct symbols (*events*), and let Σ^+ be the set of all finite nonempty sequences of events. Let $\Sigma^* := \Sigma^+ \cup \{\epsilon\}$ be the set of all finite sequences of events plus ϵ, the *empty string*. A language L over Σ is any subset $L \subseteq \Sigma^*$.

The *concatenation* of two strings $s,\ t \in \Sigma^*$, is written as st. Languages and alphabets can also be concatenated. For $L \subseteq \Sigma^*$ and $\Sigma' \subseteq \Sigma$, the concatenation of language L and event set Σ' is defined as $L\Sigma' := \{s\sigma | s \in L,\ \sigma \in \Sigma'\}$.

For strings $s,\ t \in \Sigma^*$, we say that t is a *prefix* of s (written $t \leq s$) if $s = tu$, for some $u \in \Sigma^*$. We also say that t can be *extended* to s. The *prefix closure* $\overline{L}$ of a language $L \subseteq \Sigma^*$ is defined as follows: $\overline{L} := \{t \in \Sigma^* | t \leq s \text{ for some } s \in L\}$. A

language L is said to be *prefix-closed* if $L = \overline{L}$.

Let $\mathrm{Pwr}(\Sigma)$ denote the power set of Σ (i.e., the set of all subsets of Σ). For language L, the eligibility operator $\mathrm{Elig}_L : \Sigma^* \to \mathrm{Pwr}(\Sigma)$ is given by $\mathrm{Elig}_L(s) := \{\sigma \in \Sigma \,|\, s\sigma \in L\}$ for $s \in \Sigma^*$.

2.2 Natural Projection and Inverse Projection

Let $\Sigma_o \subseteq \Sigma$. Let $s \in \Sigma^*$, and $\sigma \in \Sigma$. To capture the notion of partial observation, we define the *natural projection* $P : \Sigma^* \to \Sigma_o^*$ according to:

$$P(\epsilon) := \epsilon$$

$$P(\sigma) := \begin{cases} \epsilon, \text{ if } \sigma \notin \Sigma_o \\ \sigma, \text{ if } \sigma \in \Sigma_o \end{cases}$$

$$P(s\sigma) := P(s)P(\sigma)$$

The effect of the natural projection P on a given string $s \in \Sigma^*$ is to erase those events in string s that are not in the alphabet of Σ_o, but keep those events in Σ_o unchanged.

Given any language $L \subseteq \Sigma^*$, the *natural projection of a language* L, is

$$P(L) := \{P(s) \mid s \in L\}.$$

The *inverse projection* P^{-1}: $Pwr(\Sigma_o^*) \to Pwr(\Sigma^*)$ is defined over subsets of languages, where $Pwr(\Sigma_o^*)$ and $Pwr(\Sigma^*)$ denote all subsets of Σ_o^* and Σ^*, respectively. Given any $L \subseteq \Sigma_o^*$, the inverse projection of L is defined as:

$$P^{-1}(L) := \{s \in \Sigma^* \mid P(s) \in L\}.$$

2.3 DES

A DES is represented as a tuple:

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

with finite state set Q, finite alphabet set Σ, partial transition function $\delta:\ Q \times \Sigma \to Q$, initial state q_0, and the set of marker states Q_m. We will always assume that a DES has a finite state and event set, and is deterministic. By deterministic, we mean the DES has a single initial state and at most one transition defined at a given state for any $\sigma \in \Sigma$.

We use $\delta(q,\ \sigma)!$ to represent that δ is defined for $\sigma \in \Sigma$ at state $q \in Q$. Function δ can be extended to Σ^* by defining:

$$\delta(q,\ \epsilon) := q \text{ and } \delta(q,\ s\sigma) := \delta(\delta(q,\ s),\ \sigma),$$

provided that $q' = \delta(q,\ s)!$ and $\delta(q',\ \sigma)!$, for $s \in \Sigma^*$ and $q \in Q$.

For DES $\mathbf{G}$, its *closed behaviour* is denoted by:

$$L(\mathbf{G}) := \{s \in \Sigma^* | \delta(q_0,\ s)!\}.$$

The *marked behaviour* of $\mathbf{G}$, is defined as:

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) |\ \delta(q_o, s) \in Q_m\}.$$

The *reachable state subset* of DES $\mathbf{G}$, denoted as: Q_r, is defined as

$$Q_r := \{q \in Q | (\exists s \in \Sigma^*)\delta(q_0,\ s) = q\}.$$

A DES $\mathbf{G}$ is *reachable* if $Q_r = Q$.

The coreachable state subset, denoted by Q_{cr}, is

$$Q_{cr} := \{q \in Q |\ (\exists s \in \Sigma^*)\, \delta(q, s) \in Q_m\}.$$

A DES is *coreachable* if $Q_{cr} = Q$.

An important concept is nonblocking, a simple form of deadlock checking.

Definition 2.3.1. *A DES* $\mathbf{G}$ *is said to be* ***nonblocking*** *if*

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

This is equivalent to saying that every reachable state is also coreachable.

Definition 2.3.2. *Let* $K \subseteq L_m(\mathbf{G}) \subseteq \Sigma^*$. *We say that the language* K *is* $L_m(\mathbf{G})$***-closed*** *if*

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

We note that K being $L_m(\mathbf{G})$-closed means it contains all of its prefixes that belong to $L_m(\mathbf{G})$.

2.4 Product DES

Let $\mathbf{G}_1 = (Q_1,\ \Sigma,\ \delta_1,\ q_{0,1},\ Q_{m1})$ and $\mathbf{G}_2 = (Q_2,\ \Sigma,\ \delta_2,\ q_{0,2}\ Q_{m2})$ be two automata, over the alphabet Σ. The *product* of the two DES is defined as:

$$\mathbf{G}_1 \times \mathbf{G}_2 = (Q_1 \times Q_2,\ \Sigma,\ \delta_1 \times \delta_2,\ (q_{0,1},\ q_{0,2}),\ Q_{m1} \times Q_{m2}),$$

where $\delta_1 \times \delta_2$: $Q_1 \times Q_2 \times \Sigma \to Q_1 \times Q_2$ is given by
$(\delta_1 \times \delta_2)((q_1, q_2),\ \sigma) = (\delta_1(q_1, \sigma),\ \delta_2(q_2, \sigma))$, whenever $\delta_1(q_1, \sigma)!$ and $\delta_2(q_2, \sigma)!$.

Note that $L(\mathbf{G}_1 \times \mathbf{G}_2) := L(\mathbf{G}_1) \cap L(\mathbf{G}_2)$ and $L_m(\mathbf{G}_1 \times \mathbf{G}_2) := L_m(\mathbf{G}_1) \cap L_m(\mathbf{G}_2)$.

2.5 Synchronous Product

Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $L_1 \subseteq \Sigma_1^*$ and $L_1 \subseteq \Sigma_2^*$. For $i = 1,\ 2$, let $P_i :\ \Sigma^* \to \Sigma_i^*$ be natural projections. The *synchronous product of languages* L_1 and L_2, denoted by $L_1||L_2$, is defined to be:

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

where $P_i^{-1} : \mathrm{Pwr}(\Sigma_i^*) \to \mathrm{Pwr}(\Sigma^*)$ is the inverse image function of P_i $(i = 1, 2)$.

If both L_1 and L_2 are over the same event set Σ, then their languages have the following property:

$$L_1||L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2) = L_1 \cap L_2.$$

Definition 2.5.1. *Let* $\mathbf{G}_i = (Q_i,\ \Sigma_i,\ \delta_i,\ q_{0,i},\ Q_{mi})$, $i = 1, 2$. *We define the* ***synchronous product*** *of* $\mathbf{G}_1$ *and* $\mathbf{G}_2$, *denoted* $\mathbf{G}_1||\mathbf{G}_2$, *as:*

$$\mathbf{G}_1||\mathbf{G}_2 = (Q_1 \times Q_2,\ \Sigma_1 \cup \Sigma_2,\ \delta,\ (q_{0,1},\ q_{0,2}),\ Q_{m1} \times Q_{m2}),$$

where $\delta((q_1, q_2),\ \sigma)$ *is defined as:*

$$\begin{cases} (\delta_1(q_1,\sigma), \delta_2(q_2,\sigma)), \textit{if} \sigma \in \Sigma_1 \cap \Sigma_2, \delta_1(q_1,\sigma)!, \delta_2(q_2,\sigma)!; \\ (\delta_1(q_1,\sigma), q_2),\ \textit{if } \sigma \in \Sigma_1 \backslash \Sigma_2 \textit{ and } \delta_1(q_1,\sigma)!; \\ (q_1, \delta_2(q_2,\sigma)),\ \textit{if } \sigma \in \Sigma_2 \backslash \Sigma_1 \textit{ and } \delta_2(q_2,\sigma)!; \\ \textit{Undefined, otherwise.} \end{cases}$$

Therefore, the synchronous product of DES $\mathbf{G}_1$ and $\mathbf{G}_2$ is a DES $\mathbf{G} = \mathbf{G}_1||\mathbf{G}_2$ with event set $\Sigma = \Sigma_1 \cup \Sigma_2$ and properties:

$$L_m(\mathbf{G}) = L_m(\mathbf{G}_1)||L_m(\mathbf{G}_2),\ L(\mathbf{G}) = L(\mathbf{G}_1)||L(\mathbf{G}_2).$$

We use the synchronous product symbol || for both languages and automata, and the choice of arguments will make the meaning clear.

For our purposes, we will assume that $\mathbf{G}_1||\mathbf{G}_2$ is implemented by first adding selfloops to all states of $\mathbf{G}_i$ $(i = 1,\ 2)$ for events in $\Sigma\backslash\Sigma_i$, and then constructing the product of the two DES.

If both $\mathbf{G}_1$ and $\mathbf{G}_2$ are over the same event set Σ, then:

$$\mathbf{G}_1||\mathbf{G}_2 = \mathbf{G}_1 \times \mathbf{G}_2.$$

2.6 Controllability

Another important concept in DES is controllability. It is essentially a test to see if a set of controllable behaviour represented by language $K \subseteq \Sigma^*$ is enforcible against the systems uncontrolled behaviour represented by language $L = \overline{L} \subseteq \Sigma^*$.

In supervisory control, the event set Σ is partitioned into two disjoint sets: the *controllable* event set Σ_c and the *uncontrollable* event set Σ_{uc}. Controllable events can be prevented from happening (disabled) by a supervisor, while uncontrollable events cannot be disabled.

Definition 2.6.1. *Let K and $L = \overline{L}$ be languages over event set Σ. K is said to be* ***controllable*** *with respect to L and Σ_{uc} if and only if,*

$$\overline{K}\Sigma_{uc} \cap L \subseteq \overline{K}.$$

2.7 Decentralized Control

A *decentralized controller* is an automaton S_i, for an index set of decentralized controllers $I = \{1, ..., n\}$, that has only a partial view of the system behaviour and control a subset of the controllable events. Each controller issues its own local control decision and a final control decision is taken by fusing or combining all the local decisions with a particular fusion rule. The rule varies depending on the decentralized architecture in use.

To describe events that each decentralized controller $i \in I$ controls, we use the notation $\Sigma_{c,i} \subseteq \Sigma_c$, where $\cup_{i=1}^{n} \Sigma_{c,i} = \Sigma_c$ and each $\Sigma_{c,i}$ is not necessarily disjoint. We refer to the set of controllers that control $\sigma \in \Sigma_c$ by $I_c(\sigma) := \{i \in I \,|\sigma \in \Sigma_{c,i}\}$.

In decentralized control, the event set Σ is further partitioned into *observable events* Σ_o and *unobservable events* Σ_{uo}. To describe events that each decentralized controller $i \in I$ observes, we use the notation $\Sigma_{o,i} \subseteq \Sigma_o$, where $\cup_{i=1}^{n} \Sigma_{o,i} = \Sigma_o$ and each $\Sigma_{o,i}$ is not necessarily disjoint. We refer to the set of controllers that observe event $e \in \Sigma_o$ by $I_o(e) := \{i \in I \,|e \in \Sigma_{o,i}\}$. Correspondingly, the natural projection describing the partial view of each controller is denoted by $P_i : \Sigma^* \to \Sigma_{o,i}^*$, for all $i \in I$.

2.8 Co-observability

The property of co-observability was introduced in [RW92b], whereas the verification algorithm was introduced in [RW95]. The following is the definition of co-observability adapted from [RW92b, BL00].

Definition 2.8.1. *Let K, $L = \overline{L}$ be languages over event set Σ. Let $I = \{1, ..., n\}$*

be an index set. Let $\Sigma_{c,i} \subseteq \Sigma$ *and* $\Sigma_{o,i} \subseteq \Sigma$ *be sets of controllable and observable events, respectively, for* $i \in I$*, where* $\Sigma_c = \cup_{i=1}^{n} \Sigma_{c,i}$ *and* $I_c(\sigma) := \{i \in I \,|\sigma \in \Sigma_{c,i}\}$*. Let* $P_i : \Sigma^* \to \Sigma_{o,i}^*$ *be natural projections. A language* K *is said to be co-observable with respect to* L*,* $\Sigma_{o,i}$*,* $\Sigma_{c,i}$*,* $i \in I$*, if,*

$$(\forall t \in \overline{K} \cap L)\ (\forall \sigma \in \Sigma_c)\ t\sigma \in L\backslash\overline{K} \Rightarrow (\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset.$$

Notice that in the definition of co-observability, when there is only one controller, i.e. $I = \{1\}$, the property is called *observability* [LW88a]. Since the specification K is not necessarily a subset of L, unlike the original definition, we do not require that $K \subseteq L$. Instead of checking all strings in $\overline{K}$, reasonably, we check all strings in $\overline{K} \cap L$.

In the co-observable scenario, decentralized controllers take local control decisions based on their partial observations. When the system leaves K (i.e., $t\sigma \in L\backslash\overline{K}$, where $t\sigma \in L$ *and* $t\sigma \notin \overline{K}$) there must be at least one controller (i.e., $\exists i \in I_c(\sigma)$) that has sufficient information from its own view of the system to take the correct control decision (i.e., disable σ) for each $\sigma \in \Sigma_c$. Note that a controller i will enable all events $\sigma \in \Sigma\backslash\Sigma_{c,i}$ by default.

If an event σ needs to be disabled (i.e., $t \in \overline{K}$, $t\sigma \in L\backslash\overline{K}$), then at least one of the controllers that control σ must unambiguously know that it must disable the event σ (i.e., $P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$). So, from this supervisor's viewpoint, disabling σ does not prevent any string in $\overline{K} \cap L$. For all other controllers that are uncertain about whether they should disable the event σ, they will enable the event σ, and the final fusion rule used here is the conjunction of all the decisions of controllers.

We can synthesize decentralized controllers that cooperate to ensure that the supervised system generates exactly the behaviour in the specification K if K is controllable, $L_m(\mathbf{G})$-closed, and satisfies co-observability. In Chapter 5, we will present

a result that removes the $L_m(\mathbf{G})$-closure requirement.

The co-observability defined in this thesis is called C&P-coobservability [YL02a], where the C refers to the "conjunctive" architecture for controllable events, and the P refers to the "permissive" local decision rule. There are analogous architectures, such as D&A-coobservability and general architecture [YL02a], and conditional co-observability [YL04]. In this thesis, when we talk about co-observability, we mean the C&P-coobservability.

In the following chapters, when there is no ambiguity, instead of saying that K is co-observable with respect to L, $\Sigma_{o,i}$, $\Sigma_{c,i}$, $i = 1, ..., n$, we will say that K is co-observable w.r.t. L.

Chapter 3

Incremental Verification of Co-observability

Due to the distributed nature of the system, in decentralized supervisory control of DES, each decentralized supervisor controls and partially observes some subset of the system's behaviour. Here we assume that the common goal is to exactly realize the specification language using the decentralized decision-making process of fusing local control decisions to arrive at the final control decision. The synthesis of decentralized supervisors requires that the specification satisfies co-observability [RW92b].

When the system is very large and composed of many subsystems, verifying co-observability using existing monolithic methods requires the construction of the complete system model which may be intractable in practice due to the state-space explosion problem.

To address this problem, we introduce a new approach called *incremental verification of co-observability*. Using this method, verifying co-observability is done incrementally by evaluating selected subgroups of the system individually, until the

entire system has been shown to be co-observable. The incremental selection of suitable subgroups is guided by *counter examples*, which are defined in Section 3.2.

Properties of verifying co-observability for plants that are composed of multiple components (and the associated specifications) are presented, along with illustrative small examples. These results lead directly to the introduction of the algorithm for incremental verification of co-observability.

Most of the material in this chapter first appeared in a joint paper with R. Malik, R. Leduc and L. Ricker [LLMR14].

3.1 Co-observability of Multi-component Decentralized Systems

We consider decentralized control problems where the plant and specification languages are defined in terms of the synchronous product of many smaller languages, i.e., $L = L_1||\cdots||L_m$ and $K = K_1||\cdots||K_r$. Verifying key properties of such a system may be better accomplished, in terms of computational feasibility, by only examining subsets of these component languages. In this section we focus on the verification of co-observability, one of the key properties that must be satisfied to synthesize decentralized controllers.

Recall that if languages are defined over the same event set Σ, then the synchronous product of L (and, by extension, K) has the following property: $L = L_1||\cdots||L_m = L_1 \cap \cdots \cap L_m$. We will use this result throughout this chapter.

We first apply the property of co-observability to sublanguages.

Proposition 3.1.1. *Let K, $L = \overline{L}$, and $M = \overline{M}$ be languages defined over Σ. If K*

is co-observable w.r.t. M and $L \subseteq M$, then K is co-observable with respect to L.

Proof. Assume K is co-observable w.r.t. M and $L \subseteq M$.

Must show K is co-observable w.r.t. L.

Let $t \in \overline{K} \cap L$ and $\sigma \in \Sigma_c$.

Assume $t\sigma \in L \backslash \overline{K}$.

Sufficient to show $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.

As $L \subseteq M$ and $t \in \overline{K} \cap L$, we have $t \in \overline{K} \cap M$.

As $t\sigma \in L \backslash \overline{K}$, we have $t\sigma \in L$ and $t\sigma \notin \overline{K}$.

$\Rightarrow t\sigma \in M$ and $t\sigma \notin \overline{K}$, as $L \subseteq M$.

$\Rightarrow t\sigma \in M \backslash \overline{K}$.

Then $t \in \overline{K} \cap M$ and $t\sigma \in M \backslash \overline{K}$.

As K is co-observable w.r.t. M, we have $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap M = \emptyset$.

As $L \subseteq M$, we have $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.

As $t \in \overline{K} \cap L$ and $\sigma \in \Sigma_c$ were chosen arbitrarily, we conclude that K is co-observable w.r.t. L.

□

We illustrate the utility of Proposition 3.1.1 with the following example.

Example 1. *Consider the plant automaton G_1 in Figure 3.1 and associated specification automaton H_1 in Figure 3.2.*

Suppose that $I = \{1, 2\}$ and $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ and $\Sigma_{c,2} = \{b, g\}$. We can determine that $L(H_1) = \overline{\{\mathsf{abg}, \mathsf{bag}\}}$ is co-observable w.r.t $L(G_1) = \overline{\{\mathsf{abg}, \mathsf{bag}, \mathsf{aag}, \mathsf{bbg}\}}$ as follows. According to Definition 2.8.1, the sequences $t\sigma$ involved in disablement decisions are those in $L(G_1) \backslash (L(H_1) \cap L(G_1)) = \{\mathsf{aa}, \mathsf{bb}, \mathsf{aag}, \mathsf{bbg}\}$. We examine each in turn.

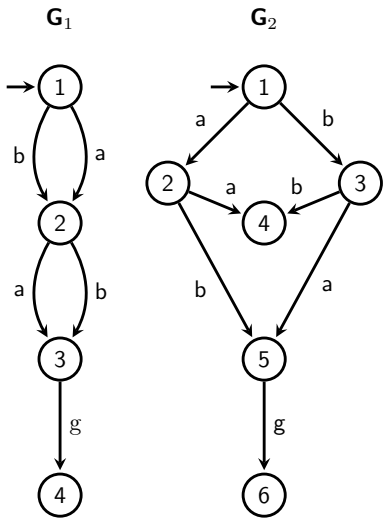


Figure 3.1: The Plant Components G_1, G_2.

Let $t\sigma = \mathsf{aa}$. Since $I_c(\mathsf{a}) = 1$, we compute $P_1^{-1}[P_1(\mathsf{a})]\mathsf{a} \cap L(G_1) = \{\mathsf{aa}, \mathsf{aag}\}$. Subsequently, we take the intersection of this with $L(H_1) \cap L(G_1) = \overline{\{\mathsf{abg}, \mathsf{bag}\}}$, which is $\emptyset$.

We continue with $t\sigma = \mathsf{bb}$. Since $I_c(\mathsf{b}) = \{2\}$, we compute $P_2^{-1}[P_2(\mathsf{b})]\mathsf{b} \cap L(G_1) = \{\mathsf{bb}, \mathsf{bbg}\}$. Taking the intersection with $L(H_1) \cap L(G_1) = \overline{\{\mathsf{abg}, \mathsf{bag}\}}$ is also $\emptyset$.

The next sequence to examine is $t\sigma = \mathsf{aag}$. Here $I_c(\mathsf{g}) = \{1, 2\}$, so we must compute $P_1^{-1}[P_1(\mathsf{aa})]\mathsf{g} \cap L(G_1) = \{\mathsf{aag}\}$ or $P_2^{-1}[P_2(\mathsf{aa})]\mathsf{g} \cap L(G_1) = \{\mathsf{aag}\}$. Although we need only one $i \in I_c(\mathsf{g})$ to satisfy Definition 2.8.1 for this particular $t\sigma$, in this case all $i \in I_c(\mathsf{g})$ result in $P_i^{-1}[P_i(\mathsf{aa})]\mathsf{g} \cap L(G_1) \cap \overline{\{\mathsf{abg}, \mathsf{bag}\}} = \emptyset$. Thus, either controller is capable of taking the correct control decision regarding g after its partial observation of the occurrence of aa.

A nearly-identical argument follows for the controllers in $I_c(\mathsf{g})$ when $t\sigma = \mathsf{bbg}$. We conclude that for each of the disablement decisions that must be taken, there is at least one controller that will correctly disable the relevant event. Thus, we conclude

that $L(H_1)$ is co-observable w.r.t.

$L(G_1)$.

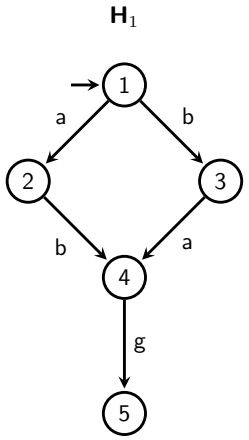


Figure 3.2: The Specification H_1.

We want to know if $L(H_1)$ is also co-observable w.r.t.

$L(G_2) = \overline{\{abg, bag, aa, bb\}}$*, with its corresponding automaton in Figure 3.1. We first check to see if $L(G_2) \subseteq L(G_1)$, an operation for which a polynomial-time algorithm exists—a significant improvement on the exponential time verification of co-observability. Since $L(G_2) \subseteq L(G_1)$, we conclude, using Proposition 3.1.1, that $L(H_1)$ is co-observable w.r.t.*

$L(G_2)$.

Proposition 3.1.1 is a fundamental proposition which can be paraphrased as follows: if a specification language K is co-observable w.r.t. a language M, then it must be co-observable w.r.t. all the prefix-closed sublanguages of M. We also know that if L_1 and L_2 are prefix-closed, so is $L = L_1 \cap L_2$.

Corollary 3.1.1. *Let K, $L_1 = \overline{L_1}$, and $L_2 = \overline{L_2}$ be languages defined over Σ. If K is co-observable with respect to L_1 then K is co-observable with respect to $L = L_1 \cap L_2$.*

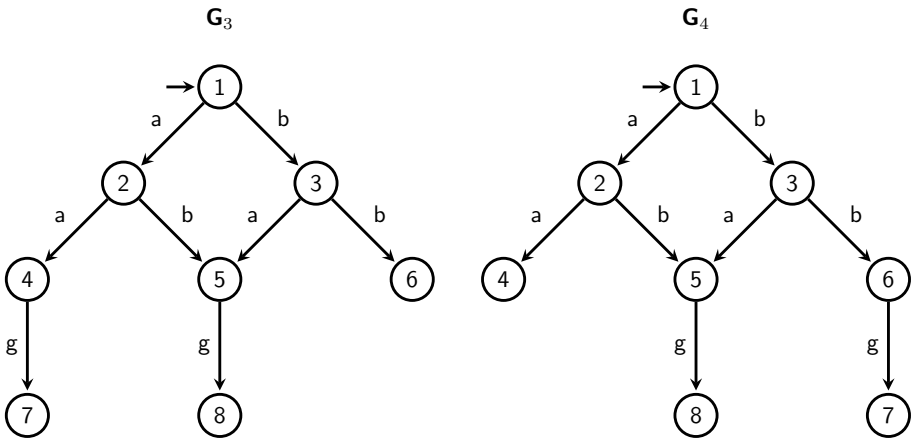


Figure 3.3: Plant Components G_3 and G_4.

Proof. Since K is co-observable w.r.t. L_1 and $L = L_1 \cap L_2 \subseteq L_1$, by Proposition 3.1.1, we have K is co-observable w.r.t. L. □

Example 2. *Consider plant automata G_3 and G_4 in Figure 3.3 and their corresponding languages $L(G_3) = \overline{\{aag, abg, bag, bb\}}$ and $L(G_4) = \overline{\{aa, abg, bag, bbg\}}$. We reuse the specification automaton H_1 in Figure 3.2, where $L(H_1) = \overline{\{abg, bag\}}$. Suppose that $I = \{1, 2\}$ such that $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ and $\Sigma_{c,2} = \{b, g\}$.*

Table 3.1 summarizes, with reference to the components of Definition 2.8.1, the verification that $L(H_1)$ is co-observable w.r.t. $L(G_3)$. In the event that $|I_c(\sigma)| > 1$, it is enough to establish that only one of the controllers satisfies the condition in column 3 of the table; however, we include all computations for completeness. Note that the sequences outside of the specification language are $L(G_3) \setminus (L(H_1) \cap L(G_3))$ $= \{aa, aag, bb\}$. Also $L(H_1) \cap L(G_3) = L(H_1)$.

We want to determine whether $L(H_1)$ is co-observable w.r.t. $L(G_3) \cap L(G_4)$. By Corollary 3.1.1, since $L(H_1)$ is co-observable w.r.t. $L(G_3)$, we can conclude exactly that. The same conclusion can be drawn if one starts with $L(G_4)$ in place of $L(G_3)$.

Table 3.1: Verification that $L(H_1)$ is co-observable w.r.t. $L(G_3)$.

$t\sigma \in L(G_3) \setminus (L(H_1) \cap L(G_3))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G_3) \cap (\overline{L(H_1) \cap L(G_3)})$
aa	1	$\{aa\} \cap \overline{\{abg, bag\}} = \emptyset$
aag	1	$\{aag\} \cap \overline{\{abg, bag\}} = \emptyset$
	2	$\{aag\} \cap \overline{\{abg, bag\}} = \emptyset$
bb	2	$\{bb\} \cap \overline{\{abg, bag\}} = \emptyset$

For ease of exposition, the uncontrolled system is described by only two automata components; however, our results can be extended to a system containing any number of components. For example, suppose that we have a plant consisting of many components $G = G_1||...||G_m$ and a single specification automaton H. We want to verify whether $L(H)$ is co-observable with respect to $L(G) = L(G_1)||...||L(G_m)$. All we need to do is determine that $L(H)$ is co-observable w.r.t. $L(G_j)$ for $j \in \{1, \ldots, m\}$, then we can conclude that $L(H)$ is co-observable w.r.t. $L(G)$.

Furthermore, if no single component can be identified, then we need only find a subset index $\{j_1, ..., j_k\} \subseteq \{1 \ldots, m\}$ such that $L(H)$ is co-observable with respect to $L(G_{j_1})||...||L(G_{j_k}) \subseteq L(G)$, then we will know that $L(H)$ is co-observable w.r.t. $L(G)$. This follows from Corollary 3.1.1.

Another aspect of incremental verification arises when examining whether or not the intersection of two co-observable languages is itself co-observable. In the following proposition, we require that K_1 and K_2 be prefix-closed since in general co-observability is not closed under intersection. Also note that co-observability of K is equivalent to co-observability of $\overline{K}$.

Proposition 3.1.2. *Let K_1, K_2 and L be prefix-closed languages defined over Σ. If both K_1 and K_2 are co-observable w.r.t. L, then $K_1 \cap K_2$ is co-observable w.r.t. L.*

Proof. Assume K_1, K_2 and L are prefix-closed languages defined over Σ. Suppose that K_1 and K_2 are both co-observable w.r.t. L.

Must show $K_1 \cap K_2$ is co-observable w.r.t. L.

Let $t \in \overline{K_1 \cap K_2} \cap L$ and let $\sigma \in \Sigma_c$.

Assume $t\sigma \in L \backslash \overline{K_1 \cap K_2}$.

Sufficient to show $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1 \cap K_2} \cap L = \emptyset$.

As $K_1 \cap K_2$, K_1 and K_2 are prefix-closed, we have $\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$.

As $t \in \overline{K_1 \cap K_2} \cap L$, we have $t \in \overline{K_1} \cap \overline{K_2} \cap L$.

$\Rightarrow t \in \overline{K_1} \cap L$ and $t \in \overline{K_2} \cap L$.

As $t\sigma \in L \setminus \overline{K_1 \cap K_2}$, we have $t\sigma \in L$ and $t\sigma \notin \overline{K_1 \cap K_2}$.

As $t\sigma \notin \overline{K_1 \cap K_2}$, and $\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$, we have $t\sigma \notin \overline{K_1}$ or $t\sigma \notin \overline{K_2}$.

$\Rightarrow t\sigma \notin \overline{K_1}$ *and* $t\sigma \in L$ or $t\sigma \notin \overline{K_2}$ *and* $t\sigma \in L$.

Case 1) $t\sigma \notin \overline{K_1}$ *and* $t\sigma \in L$.

Then $t \in \overline{K_1} \cap L$, $t\sigma \notin \overline{K_1}$ *and* $t\sigma \in L$.

$\Rightarrow t \in \overline{K_1} \cap L$, $t\sigma \in L \backslash \overline{K_1}$.

As K_1 is co-observable w.r.t. L, we have $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1} \cap L = \emptyset$.

As $\overline{K} \subseteq \overline{K_1}$, we have $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.

Case 2) $t\sigma \notin \overline{K_2}$ *and* $t\sigma \in L$.

Then $t \in \overline{K_2} \cap L$, $t\sigma \notin \overline{K_2}$ *and* $t\sigma \in L$.

$\Rightarrow t \in \overline{K_2} \cap L$, $t\sigma \in L \backslash \overline{K_2}$.

As K_2 is co-observable w.r.t. L, we have $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_2} \cap L = \emptyset$.

As $\overline{K} \subseteq \overline{K_2}$, we have $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.

As $t \in \overline{K_1 \cap K_2}$ and $\sigma \in \Sigma_c$ are chosen arbitrarily, we conclude by Cases 1) and 2) that $K_1 \cap K_2$ is co-observable w.r.t. L.

□

Example 3. *Consider the specification automata H_1 in Figure 3.2 and H_2 in Figure 3.4. Suppose that $I = \{1, 2\}$ such that $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ and $\Sigma_{c,2} = \{b, g\}$.*

We want to examine co-observability of $L(H_1)$ and $L(H_2)$ w.r.t. the language generated by the plant automaton G_1 in Figure 3.1.

In Example 1, we established that $L(H_1)$ is co-observable w.r.t. $L(G_1)$, so now we examine $L(H_2)$.

Table 3.2 summarizes the details of verifying that $L(H_2)$ is co-observable w.r.t. $L(G_1)$. We thus know, by Proposition 3.1.2, that $L(H_1) \cap L(H_2)$, which in this case is simply $L(H_2)$, is co-observable w.r.t. $L(G_1)$.

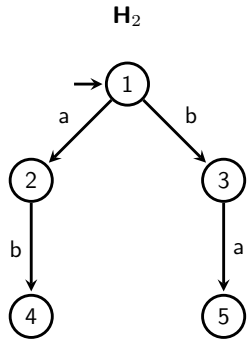


Figure 3.4: Specification H_2.

Proposition 3.1.2 can also be extended to an arbitrary number of specification languages.

In the incremental verification of co-observability, given a specification language $K = K_1 \cap \cdots \cap K_r$ and a language L, if we want to verify whether K is co-observable w.r.t. L, it is enough to simply show that for each $j \in \{1, ..., r\}$, K_j is co-observable

Table 3.2: Verification that $L(H_2)$ is co-observable w.r.t. $L(G_1)$.

$t\sigma \in L(G_1) \setminus (L(H_2) \cap L(G_1))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G_1) \cap (L(H_2) \cap L(G_1))$
aa	1	$\{aa\} \cap \overline{\{ab, ba\}} = \emptyset$
bb	2	$\{bb\} \cap \overline{\{ab, ba\}} = \emptyset$
aag	1	$\{aag\} \cap \overline{\{ab, ba\}} = \emptyset$
	2	$\{aag\} \cap \overline{\{ab, ba\}} = \emptyset$
bbg	1	$\{bbg\} \cap \overline{\{ab, ba\}} = \emptyset$
	2	$\{bbg\} \cap \overline{\{ab, ba\}} = \emptyset$
abg	1	$\{abg, bag\} \cap \overline{\{ab, ba\}} = \emptyset$
	2	$\{abg, bag\} \cap \overline{\{ab, ba\}} = \emptyset$
bag	1	$\{abg, bag\} \cap \overline{\{ab, ba\}} = \emptyset$
	2	$\{abg, bag\} \cap \overline{\{ab, ba\}} = \emptyset$

w.r.t. L. Combining this with Proposition 3.1.1, we see that we can use a subsystem L' instead of the global system L for the verification.

Proposition 3.1.3. *Let K_1, K_2, M_1 and M_2 be prefix-closed languages defined over Σ. If K_1 is co-observable w.r.t. M_1 and K_2 is co-observable with respect to M_2, then $K_1 \cap K_2$ is co-observable w.r.t. $M_1 \cap M_2$.*

Proof. Assume K_1, K_2, M_1, and M_2 are prefix-closed. Suppose that K_1 is co-observable w.r.t. M_1 and K_2 is co-observable w.r.t. M_2.

Must show K is co-observable w.r.t. $M_1 \cap M_2$.

We (trivially) have $M_1 \cap M_2 \subseteq M_1$ *and* $M_1 \cap M_2 \subseteq M_2$.

As K_1 is co-observable w.r.t. M_1, and $M_1 \cap M_2 \subseteq M_1$, we have K_1 is co-observable w.r.t. $M_1 \cap M_2$, by Proposition 3.1.1.

As K_2 is co-observable w.r.t. M_2, and $M_1 \cap M_2 \subseteq M_2$, we have K_2 is co-observable w.r.t. $M_1 \cap M_2$, by Proposition 3.1.1.

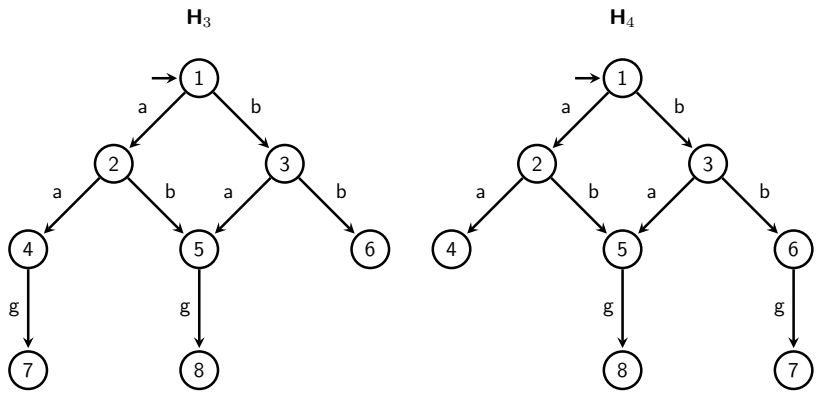


Figure 3.5: Specification Automata H_3 and H_4.

As K_1 and K_2 are both co-observable w.r.t. $M_1 \cap M_2$, and since K_1, K_2, and M are prefix-closed, we have $K_1 \cap K_2$ is co-observable w.r.t. $M_1 \cap M_2$ by Proposition 3.1.2. □

Example 4. *In this example, we recast the plant automata from Example 2 as specification automata in Figure 3.5. Thus we have two specification languages* $L(H_3) = \overline{\{aag, abg, bag, bb\}}$ *and* $L(H_4) = \overline{\{aa, abg, bag, bbg\}}$. *Again, we let* $I = \{1, 2\}$ *such that* $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ *and* $\Sigma_{c,2} = \{b, g\}$. *We want to examine the co-observability of these languages w.r.t. the languages generated by two new plant automata* G_5 *and* G_6, *namely* $L(G_5) = \overline{\{abg, bag, aag, aaa, bbg\}}$ *and* $L(G_6) = \overline{\{abg,\ bag, aag, bbg, bbb\}}$. *The plant component automata are illustrated in Figure 3.6.*

We want verify that $L(H_3)$ *is co-observable w.r.t.* $L(G_5)$ *and that* $L(H_4)$ *is co-observable w.r.t.* $L(G_6)$. *Then by Proposition 3.1.3, we will know that* $L(H_3) \cap L(H_4)$ *is co-observable w.r.t.* $L(G_5) \cap L(G_6)$.

Table 3.3 summarizes the particulars of why $L(H_3)$ *is co-observable w.r.t.* $L(G_5)$, *whereas Table 3.4 summarizes the specifics in showing that* $L(H_4)$ *is co-observable w.r.t.* $L(G_6)$. *Thus, we can conclude, by Proposition 3.1.3, that* $L(H_3) \cap L(H_4)$ *is*

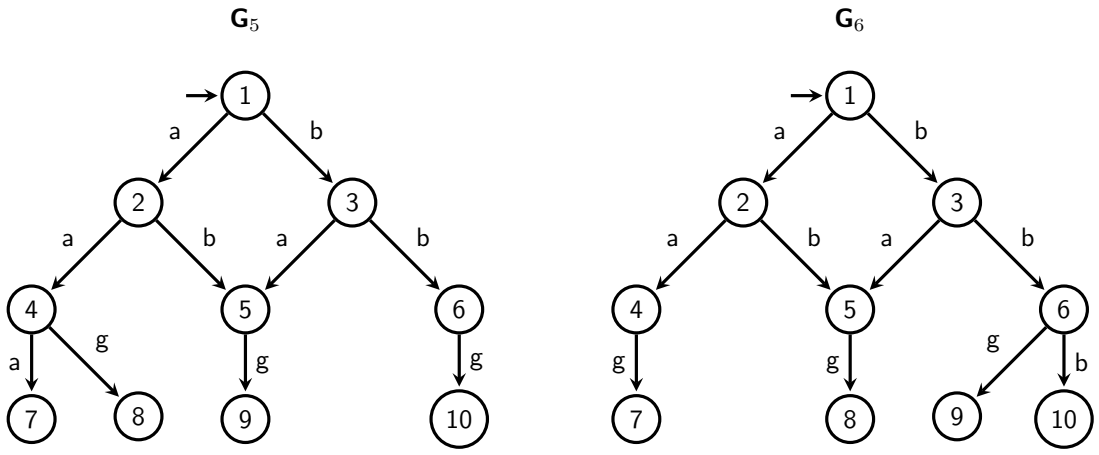


Figure 3.6: Plants G_5 and G_6.

co-observable w.r.t. $L(G_5) \cap L(G_6)$.

Table 3.3: Verification that $L(H_3)$ is co-observable w.r.t. $L(G_5)$.

$t\sigma \in L(G_5) \setminus (L(H_3) \cap L(G_5))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G_5) \cap (L(H_3) \cap L(G_5))$
aaa	*1*	$\{aaa\} \cap \overline{\{abg, bag, aag, bb\}} = \emptyset$
bbg	*1*	$\{bbg\} \cap \overline{\{abg, bag, aag, bb\}} = \emptyset$
	2	$\{bbg\} \cap \overline{\{abg, bag, aag, bb\}} = \emptyset$

The result of Proposition 3.1.3 can be extended to any system that contains an arbitrary number of specification and plant languages.

We also consider the case when the specification language is a super language of the plant language.

Proposition 3.1.4. *Let K and L be prefix-closed languages defined over Σ. If $K \supseteq L$ then K is co-observable w.r.t. L.*

Proof. Assume $K \supseteq L$.

Table 3.4: Verification that $L(H_4)$ is co-observable w.r.t. $L(G_6)$.

$t\sigma \in L(G_6) \setminus (L(H_4) \cap L(G_6))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G_6) \cap (L(H_4) \cap L(G_6))$
aag	1	$\{aag\} \cap \overline{\{abg, bag, aa, bbg\}} = \emptyset$
	2	$\{aag\} \cap \overline{\{abg, bag, aa, bbg\}} = \emptyset$
bbb	2	$\{bbb\} \cap \overline{\{abg, bag, aa, bbg\}} = \emptyset$

Then every string in L is also in $\overline{K}$.

$\Rightarrow L \backslash \overline{K} = \emptyset$.

Therefore the pre-condition of co-observability, $t\sigma \in L\backslash\overline{K}$, is always *false*.

As there does not exist $t\sigma \in L\backslash\overline{K}$ satisfying the precondition of Definition 2.8.1, the result that K is co-observable w.r.t. L is trivially *true*.

We thus conclude that K is co-observable w.r.t. L.

□

Example 5. *Consider the uncontrolled plant G_2, generating $L(G_2) = \overline{\{abg,\ bag,\ aa,\ bb\}}$, and specification H_5, with associated language $L(H_5) = \overline{\{abg,\ bag,\ aag,\ bbg\}}$ in Figure 3.7. Let $I = \{1, 2\}$ such that $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ and $\Sigma_{c,2} = \{b, g\}$. We want to ascertain whether $L(H_5)$ is co-observable w.r.t. G_2.*

In this case, $L(H_5) \supseteq L(G_2)$. Thus, by Proposition 3.1.4, we assert that $L(H_5)$ is co-observable w.r.t. $L(G_2)$. Since G_2 does not generate any sequences that require disablement actions to be taken, the system is vacuously co-observable.

Proposition 3.1.4 indicates that any language is co-observable w.r.t. all its sub-languages.

We next consider the effect that intersection has on the preservation of co-observability.

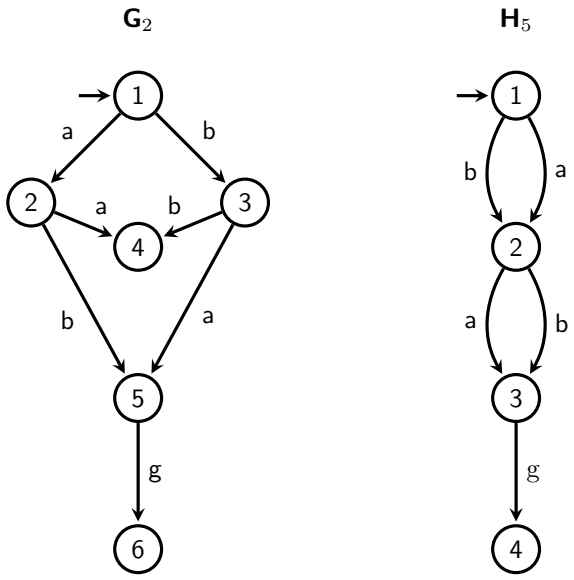


Figure 3.7: Plant G_2 and Specification H_5.

Proposition 3.1.5. *Let K_1, K_2 and M be prefix-closed languages defined over Σ. If K_1 is co-observable w.r.t. $M \cap K_2$, and K_2 is co-observable w.r.t. M, then $K_1 \cap K_2$ is co-observable w.r.t. M.*

Proof. Assume K_1 is co-observable w.r.t. $M \cap K_2$.

Assume K_2 is co-observable w.r.t. M.

Assume K_1, K_2 and M are prefix-closed.

Must show $K_1 \cap K_2$ is co-observable w.r.t. M.

Let $t \in \overline{K_1 \cap K_2} \cap M$.

Let $\sigma \in \Sigma_c$.

Assume $t\sigma \in M \backslash \overline{K_1 \cap K_2}$.

Sufficient to show $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1 \cap K_2} \cap M = \emptyset$.

As $t \in \overline{K_1 \cap K_2} \cap M$ and $\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$, for prefix-closed languages we have $t \in \overline{K_1} \cap M$ and $t \in \overline{K_2} \cap M$.

As $t\sigma \in M\backslash\overline{K_1 \cap K_2}$, we have $t\sigma \in M$ and $t\sigma \notin \overline{K_1 \cap K_2}$.

$\Rightarrow t\sigma \in M$ and ($t\sigma \notin \overline{K_1}$ or $t\sigma \notin \overline{K_2}$) for prefix-closed languages.

$\Rightarrow$ ($t\sigma \in M$ and $t\sigma \notin \overline{K_1}$) or ($t\sigma \in M$ and $t\sigma \notin \overline{K_2}$).

Case 1) $t\sigma \in M$ and $t\sigma \notin \overline{K_2}$.

Then $t\sigma \in M\backslash\overline{K_2}$.

$\Rightarrow t \in \overline{K_2} \cap M$ and $t\sigma \in M\backslash\overline{K_2}$.

As K_2 is co-observable w.r.t. M, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_2} \cap M = \emptyset$.

As $\overline{K_1 \cap K_2} \subseteq \overline{K_2}$, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1 \cap K_2} \cap M = \emptyset$.

Case 2) $t\sigma \in M$ and $t\sigma \notin \overline{K_1}$.

Under this case, there are still two cases: $t\sigma \notin \overline{K_2}$ or $t\sigma \in \overline{K_2}$.

As $t\sigma \notin \overline{K_2}$ is the same as case 1), we only consider $t\sigma \in \overline{K_2}$.

$\Rightarrow t\sigma \in M$ and $t\sigma \notin \overline{K_1}$ and $t\sigma \in \overline{K_2}$.

We will use the assumption that K_1 is co-observable w.r.t. $M \cap K_2$ to prove the result.

We will thus look on K_1 as a specification, and $M \cap K_2$ as a plant.

As $t\sigma \in M$ and $t\sigma \notin \overline{K_1}$ and $t\sigma \in \overline{K_2}$, we have: $t\sigma \in M \cap \overline{K_2}$ and $t\sigma \notin \overline{K_1}$.

$\Rightarrow t\sigma \in (M \cap \overline{K_2})\backslash\overline{K_1}$.

As $t \in \overline{K_1} \cap M$ and $t \in \overline{K_2} \cap M$, we have: $t \in \overline{K_1} \cap (M \cap \overline{K_2})$.

$\Rightarrow t \in \overline{K_1} \cap (M \cap \overline{K_2})$ and $t\sigma \in (M \cap \overline{K_2})\backslash\overline{K_1}$.

As K_1 is co-observable w.r.t. $M \cap \overline{K_2}$, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}\left[P_i(t)\right]\sigma \cap \overline{K_1} \cap (M \cap \overline{K_2}) = \emptyset$.

$\Rightarrow (\exists i \in I_c(\sigma))\ P_i^{-1}\left[P_i(t)\right]\sigma \cap (\overline{K_1} \cap \overline{K_2} \cap M) = \emptyset$.

As $t \in \overline{K_1 \cap K_2}$ and $\sigma \in \Sigma_c$ are chosen arbitrarily, we conclude by cases 1) and 2), that $K_1 \cap K_2$ is co-observable w.r.t. M. □

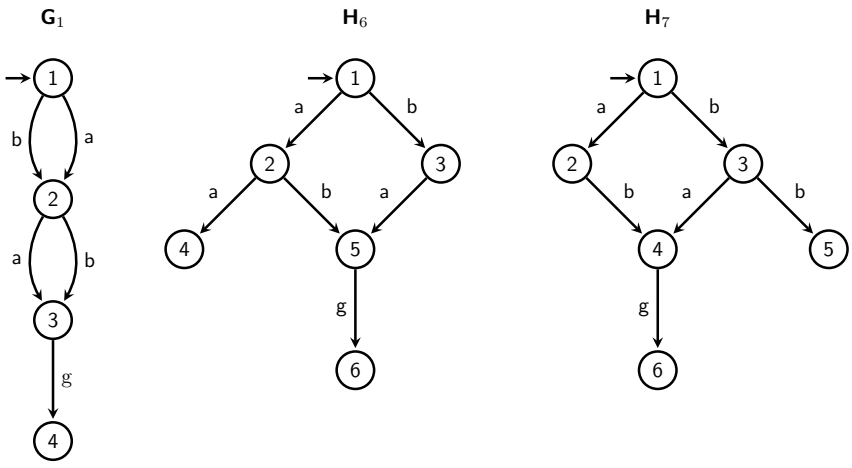


Figure 3.8: Plant Automaton G_1 and Specification Automata H_6 and H_7 for Example 6.

Example 6. *In this example, we consider two specification automata in Figure 3.8 and their corresponding languages* $L(H_6) = \overline{\{abg, bag, aa\}}$ *and* $L(H_7) = \overline{\{abg, bag, bb\}}$. *We once again visit the plant language from Example 1, namely,* $L(G_1) = \overline{\{abg, bag, aag, bbg\}}$, *and we let* $I = \{1, 2\}$ *such that* $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ *and* $\Sigma_{c,2} = \{b, g\}$.

We first want to verify that $L(H_6)$ *is co-observable w.r.t.* $L(G_1) \cap L(H_7)$. *There is only one sequence to check, since* $(L(G_1) \cap L(H_7)) \setminus (L(H_6) \cap (L(G_1) \cap L(H_7)) = \overline{\{abg, bag, bb\}} \setminus \overline{\{abg, bag\}} = \{bb\}$. *According to Table 3.5,* $L(H_6)$ *is co-observable w.r.t.* $L(G_1) \cap L(H_7)$.

We next want to verify that $L(H_7)$ *is co-observable w.r.t.* $L(G_1)$. *There are several sequences to check since* $L(G_1) \setminus (L(H_7) \cap L(G_1)) = \overline{\{abg, bag, aag, bbg\}} \setminus \overline{\{abg, bag, bb\}} = \{bbg, aag, aa\}$. *Table 3.6 confirms that* $L(H_7)$ *is co-observable w.r.t.* $L(G_1)$. *We can thus apply Proposition 3.1.5 to conclude that* $L(H_6) \cap L(H_7)$ *is co-observable w.r.t.* $L(G_1)$.

We will use Proposition 3.1.5 to show co-observability when K_2 is co-observable

Table 3.5: Verification that $L(H_6)$ is co-observable w.r.t. $L(G_1) \cap L(H_7)$. Let $L(G) = L(G_1) \cap L(H_7)$ for readability.

$t\sigma \in L(G) \setminus (L(H_6) \cap L(G))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G) \cap (L(H_6) \cap L(G))$
bb	1	$\{bb\} \cap \overline{\{abg, bag\}} = \emptyset$

Table 3.6: Verification that $L(H_7)$ is co-observable w.r.t. $L(G_1)$.

$t\sigma \in L(G_1) \setminus (L(H_7) \cap L(G_1))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G_1) \cap (L(H_7) \cap L(G_1))$
bbg	1	$\{bbg\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
	2	$\{bbg\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
aag	1	$\{aag\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
	2	$\{aag\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
aa	1	$\{aa\} \cap \overline{\{abg, bag, bb\}} = \emptyset$

w.r.t. M but K_1 is not co-observable w.r.t. M. Recall we still have that $K = K_1 \cap K_2$ is co-observable w.r.t. M if K_1 is co-observable w.r.t. the extended system $M \cap K_2$, according to Proposition 3.1.5.

Essentially, Proposition 3.1.5 allows us to treat specification K_2 as a plant component once we have shown that K_2 is co-observable w.r.t. M. To see why this is an advantage, consider the definition of co-observability (Definition 2.8.1). It states we need to check a property with respect to languages $K_1 \cap K_2$ and M for every $t\sigma \in M \setminus \overline{K_1 \cap K_2}$. If we treat K_2 as a plant, we only have to check the property for every $t\sigma \in (M \cap K_2) \setminus \overline{K_1}$. We both potentially reduce the number of strings that we have to check and that could cause the check to fail. Moreover, if we treated K_2 as a specification, we would likely have to add the required additional plant component and specifications that were needed during the incremental check of K_2 (see Algorithm 1, discussed later in the chapter, for details) to those needed to check just K_1,

resulting in a larger sized subsystem to check.

Proposition 3.1.6. *Let K_1, K_2, M_1 and M_2 be prefix-closed languages defined over Σ. If K_1 is co-observable w.r.t. $M_1 \cap K_2$, and K_2 is co-observable w.r.t. M_2, then $K_1 \cap K_2$ is co-observable w.r.t. $M_1 \cap M_2$.*

Proof. Assume K_1, K_2, M_1 and M_2 are prefix-closed.

Assume K_1 is co-observable w.r.t. $M_1 \cap K_2$.

Assume K_2 is co-observable w.r.t. M_2.

Must show $K_1 \cap K_2$ is co-observable w.r.t. $M_1 \cap M_2$.

As $M_1 \cap M_2 \subseteq M_1$, we have $M_1 \cap M_2 \cap K_2 \subseteq M_1 \cap K_2$.

As K_1 is co-observable w.r.t. $M_1 \cap K_2$, by Proposition 3.1.1, we have K_1 is co-observable w.r.t. $M_1 \cap M_2 \cap K_2$.

As K_2 is co-observable w.r.t. M_2, and $M_1 \cap M_2 \subseteq M_2$, by Proposition 3.1.1, we have K_2 is co-observable w.r.t. $M_1 \cap M_2$.

$\Rightarrow K_1$ is co-observable w.r.t. $M_1 \cap M_2 \cap K_2$, and K_2 is co-observable w.r.t. $M_1 \cap M_2$.

We thus have that $K_1 \cap K_2$ is co-observable w.r.t. $M_1 \cap M_2$, by Proposition 3.1.5.

□

Example 7. *In this example, we revisit plant automata G_5 and G_6 and examine them along with the specification automata from Example 6. This collection of automata is shown in Figure 3.9. As a reminder, the associated languages are: $L(G_5) = \overline{\{abg,\ bag,\ aag,\ aaa,\ bbg\}}$, $L(G_6) = \overline{\{abg,\ bag,\ aag,\ bbg,\ bbb\}}$, $L(H_6) = \overline{\{abg,\ bag,\ aa\}}$ and $L(H_7) = \overline{\{abg,\ bag,\ bb\}}$. As before, we suppose that $I = \{1, 2\}$ such that $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ and $\Sigma_{c,2} = \{b, g\}$.*

We first determine that $L(H_6)$ is co-observable w.r.t. $L(G_5) \cap L(H_7)$, the evidence for which can be seen in Table 3.7. We then establish that $L(H_7)$ is co-observable w.r.t.

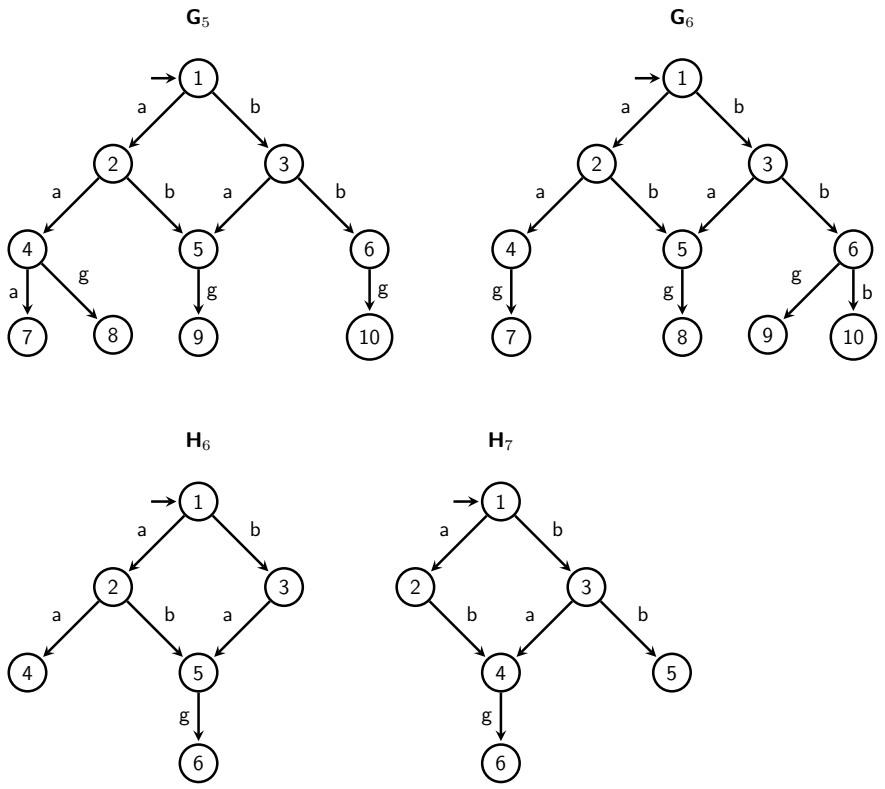


Figure 3.9: Plant Automata G_5 and G_6 from Example 4 and Specification Automata H_6 and H_7 from Example 6.

$L(G_6)$, as illustrated in Table 3.8. Since we verified that $L(H_6)$ is co-observable w.r.t. $L(G_5) \cap L(H_7)$, and $L(H_7)$ is co-observable w.r.t. $L(G_6)$, we can apply Proposition 3.1.6 and conclude that $L(H_6) \cap L(H_7)$ is co-observable w.r.t. $L(G_5) \cap L(G_6)$.

Table 3.7: Verification that $L(H_6)$ is co-observable w.r.t. $L(G_5) \cap L(H_7)$. Let $L(G) = L(G_5) \cap L(H_7)$ for readability.

$t\sigma \in L(G) \setminus (L(H_6) \cap L(G))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G) \cap (L(H_6) \cap L(G))$
bb	1	$\{bb\} \cap \overline{\{abg, bag\}} = \emptyset$

Compared to Proposition 3.1.5, Proposition 3.1.6 provides us with a more general way to incrementally verify co-observability, especially for systems composed of a

Table 3.8: Verification that $L(H_7)$ is co-observable w.r.t. $L(G_6)$.

$t\sigma \in L(G_6) \setminus (L(H_7) \cap L(G_6))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G_6) \cap (L(H_7) \cap L(G_6))$
aag	1	$\{aag\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
	2	$\{aag\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
bbg	1	$\{bbg\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
	2	$\{bbg\} \cap \overline{\{abg, bag, bb\}} = \emptyset$
bbb	2	$\{bbb\} \cap \overline{\{abg, bag, bb\}} = \emptyset$

large number of subsystems. As was the case with Proposition 3.1.3, this result can be extended to systems that have an arbitrary number of plant and/or specification components.

We now want to examine when co-observability does not hold among multiple languages.

Proposition 3.1.7. *Let K_1, K_2 and M be prefix-closed languages defined over Σ. If K_1 is not co-observable w.r.t. $M \cap K_2$, then $K_1 \cap K_2$ is not co-observable w.r.t. M.*

Proof. Assume K_1, K_2, and M are prefix-closed.

Assume K_1 is not co-observable w.r.t. $M \cap K_2$.

Must show $K_1 \cap K_2$ is not co-observable w.r.t. M.

Sufficient to show $(\exists t \in \overline{K_1 \cap K_2} \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \backslash \overline{K_1 \cap K_2}$ and $(\forall i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1 \cap K_2} \cap M \neq \emptyset$.

We will use the condition that K_1 is not co-observable w.r.t. $M \cap K_2$ to prove the result.

We thus look on K_1 as a specification, and $M \cap K_2$ as a plant.

As K_1 is not co-observable w.r.t. $M \cap K_2$, it follows that: $(\exists t \in \overline{K_1} \cap (M \cap K_2))\ (\exists \sigma \in \Sigma_c)\ t\sigma \in (M \cap K_2) \backslash \overline{K_1}$ and $(\forall i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1} \cap (M \cap K_2) \neq \emptyset$.

$\Rightarrow (\exists t \in \overline{K_1} \cap (M \cap K_2))\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \cap K_2,\ t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1} \cap (M \cap K_2) \neq \emptyset$.

$\Rightarrow (\exists t \in \overline{K_1} \cap (M \cap K_2))\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \cap K_2,\ t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap (M \cap K_2))\ P_i(t) = P_i(t_i),\ t_i\sigma \in \overline{K_1} \cap (M \cap K_2)$.

$\Rightarrow (\exists t \in \overline{K_1} \cap K_2 \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \cap K_2,\ t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap K_2 \cap M)\ P_i(t) = P_i(t_i),\ t_i\sigma \in \overline{K_1} \cap K_2 \cap M$.

$\Rightarrow (\exists t \in \overline{K_1} \cap \overline{K_2} \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \cap K_2,\ t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap \overline{K_2} \cap M)\ P_i(t) = P_i(t_i),\ t_i\sigma \in \overline{K_1} \cap \overline{K_2} \cap M$, as K_2 is prefix-closed.

$\Rightarrow (\exists t \in \overline{K_1} \cap \overline{K_2} \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \cap K_2,\ t\sigma \notin \overline{K_1} \cap \overline{K_2}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap \overline{K_2} \cap M)\ P_i(t) = P_i(t_i),\ t_i\sigma \in \overline{K_1} \cap \overline{K_2} \cap M$.

$\Rightarrow (\exists t \in \overline{K_1 \cap K_2} \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \cap K_2,\ t\sigma \notin \overline{K_1 \cap K_2}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1 \cap K_2} \cap M)\ P_i(t) = P_i(t_i),\ t_i\sigma \in \overline{K_1 \cap K_2} \cap M$, as $\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$, since K_1 and K_2 are prefix-closed.

$\Rightarrow (\exists t \in \overline{K_1 \cap K_2} \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M,\ t\sigma \notin \overline{K_1 \cap K_2}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1 \cap K_2} \cap M),\ P_i(t) = P_i(t_i),\ t_i\sigma \in \overline{K_1 \cap K_2} \cap M$, as $M \cap K_2 \subseteq M$.

$\Rightarrow (\exists t \in \overline{K_1 \cap K_2} \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \backslash \overline{K_1 \cap K_2}$ and $(\forall i \in I_c(\sigma))\ (\exists t_i \in \overline{K_1 \cap K_2} \cap M)\ P_i(t) = P_i(t_i),\ t_i\sigma \in \overline{K_1 \cap K_2} \cap M$.

$\Rightarrow (\exists t \in \overline{K_1 \cap K_2} \cap M)\ (\exists \sigma \in \Sigma_c)\ t\sigma \in M \backslash \overline{K_1 \cap K_2}$ and $(\forall i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1 \cap K_2} \cap M \neq \emptyset$.

We thus conclude that $K_1 \cap K_2$ is not co-observable w.r.t. M.

□

Proposition 3.1.7 is used for incremental verification to determine when a specification is not co-observable. If we can show that K_1 is not co-observable w.r.t. $M \cap K_2$, then we can conclude that $K = K_1 \cap K_2$ is not co-observable w.r.t. M.

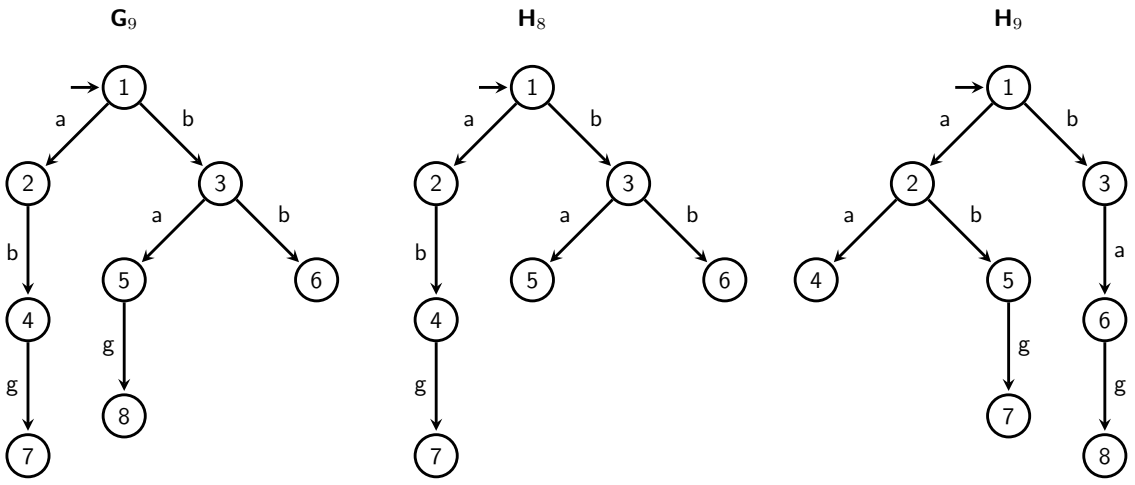


Figure 3.10: Plant Automaton G_9 and Specification Automata H_8 and H_9 for Example 8.

Example 8. *In this example, we consider the specification languages* $L(H_8) = \overline{\{abg, ba, bb\}}$ *and* $L(H_9) = \overline{\{aa, abg, bag\}}$ *and the uncontrolled plant language* $L(G_9) = \overline{\{abg, bag, bb\}}$. *The associated automata are shown in Figure 3.10. Also, suppose that* $I = \{1, 2\}$ *such that* $\Sigma_{o,1} = \{a\}$, $\Sigma_{c,1} = \{a, g\}$, $\Sigma_{o,2} = \{b\}$ *and* $\Sigma_{c,2} = \{b, g\}$.

We want to determine whether $L(H_8)$ *is co-observable w.r.t.* $L(G_9) \cap L(H_9)$. *By the results in Table 3.9, it is clear that since neither controller can take the correct control decision about* g *after its partial observation of* bag, $L(H_8)$ *is not co-observable w.r.t.* $L(G_9) \cap L(H_9)$.

By Proposition 3.1.7, we can conclude that $L(H_8) \cap L(H_9)$ *is not co-observable w.r.t.* $L(G_9)$. *Although, it is interesting to note that* $L(H_9)$ *is co-observable w.r.t.* $L(G_9)$ *(refer to Table 3.10).*

In the incremental verification of co-observability, we can use each of the above propositions independently. Furthermore, we also can combine any of the above propositions to verify co-observability in a very flexible way.

Table 3.9: Verification that $L(H_8)$ is not co-observable w.r.t. $L(G_9) \cap L(H_9)$. Let $L(G) = L(G_9) \cap L(H_9)$ for readability.

$t\sigma \in L(G) \setminus (L(H_8) \cap L(G))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G) \cap (L(H_8) \cap L(G))$
bag	1	$\{bag, abg\} \cap \overline{\{abg\}} = abg$
	2	$\{bag, abg\} \cap \overline{\{abg\}} = abg$

Table 3.10: Verification that $L(H_9)$ is co-observable w.r.t. $L(G_9)$.

$t\sigma \in L(G_9) \setminus (L(H_9) \cap L(G_9))$	$i \in I_c(\sigma)$	$P_i^{-1}[P_i(t)]\sigma \cap L(G_9) \cap (L(H_9) \cap L(G_9))$
bb	2	$\{bb\} \cap \overline{\{abg, bag\}} = \emptyset$

3.2 Algorithm

The previous section provides us with the theoretical foundations on which we can design an algorithm to verify co-observability without necessarily building the complete system. We have taken inspiration from a family of so-called *incremental verification* algorithms, e.g., [BMM04], where we choose which subsystem to examine next based on the idea of *counter examples*.

Definition 3.2.1. *Let K and L be prefix-closed languages defined over Σ. Let $I = \{1, ..., n\}$ be an index set. Let $\Sigma_{c,i} \subseteq \Sigma$ and $\Sigma_{o,i} \subseteq \Sigma$ be sets of controllable and observable events, respectively, for $i \in I$, where $I_c(\sigma) := \{i \in I \,|\sigma \in \Sigma_{c,i}\}$. Let $P_i : \Sigma^* \to \Sigma_{o,i}^*$ be natural projections. A **counter example** for the specification K and the plant L is a tuple*

$$C = (\sigma, t, t_1, \ldots, t_i, \ldots, t_n),$$

where

- $\sigma \in \Sigma_c$;
- $t \in \overline{K} \cap L$ *and* $t\sigma \in L \backslash \overline{K}$;
- $(\forall i \in I_c(\sigma))\ t_i\sigma \in \overline{K} \cap L$;
- $(\forall i \in I_c(\sigma))\ P_i(t) = P_i(t_i)$.

A counter example encodes a violation of co-observability w.r.t. controllable event σ. It includes a sequence generated by the system after which σ must be disabled as well as a set of sequences, one for each controller in $I_c(\sigma)$, whereby the controller believes that it is observing a sequence that remains in the specification after σ occurs (i.e., must be enabled). We are not concerned about the observations of those controllers $j \in I \setminus I_c(\sigma)$, and will therefore, when applicable, denote their entries in C by a "-".

For purposes of developing our algorithm, we want to define when a counter example is rejected by a plant component

Definition 3.2.2. *Let $C = (\sigma, t, t_1, \ldots, t_i, \ldots, t_n)$ be a counter example w.r.t. specification language $L(H_1)$ and plant language $L(G_1)$, defined over Σ. We say that prefix-closed* ***plant language*** *$L(G_2)$ (also defined over Σ)* ***rejects*** *C if either*

- $t\sigma \notin L(G_2)$; *or*
- $(\exists i \in I_c(\sigma))\ t_i\sigma \notin L(G_2)$.

The intention is that if we replace L in Definition 3.2.1, by $L(G_1) \cap L(G_2)$, then C is no longer a valid counter example (i.e., no longer a violation of co-observability).

Similarly, we define when a counter example is rejected by a specification component. We note that Definition 3.2.2 and 3.2.3 are essentially the same except that

a plant component can reject string $t\sigma$ while a specification can only reject string t. This demonstrates that a plant component is more powerful at rejecting a counter example and highlights the advantage of treating a specification as a plant once it has been shown to be co-observable.

Definition 3.2.3. *Let $C = (\sigma, t, t_1, \ldots, t_i, \ldots, t_n)$ be a counter example w.r.t. prefix-closed specification language $L(H_1)$ and prefix-closed plant language $L(G_1)$, defined over Σ. We say that prefix-closed* ***specification language*** *$L(H_2)$ (also defined over Σ)* ***rejects*** *C if either:*

- $t \notin L(H_2)$*; or*
- $(\exists i \in I_c(\sigma))\ t_i\sigma \notin L(H_2)$.

Analogously, the idea here is that if we replace K in Definition 3.2.1, by $L(H_1) \cap L(H_2)$, then C is no longer a valid counter example (i.e., no longer a violation of co-observability).

The input to the algorithm are two sets of automata: one consisting of the plant components $\mathcal{G} = \{G_1, G_2, \ldots, G_v\}$ and the other being the specification automata $\mathcal{H} = \{H_1, H_2, \ldots, H_w\}$. When convenient we will use the notation $L(\mathcal{G})$, respectively $L(\mathcal{H})$, to refer to $L(G_1)||L(G_2)|| \ldots ||L(G_v)$, respectively $L(H_1)||L(H_2)|| \ldots ||L(H_w)$.

The design of the algorithm is influenced by the previously-presented propositions as follows:

1. The core idea of the algorithm design revolves around Proposition 3.1.1 and Corollary 3.1.1. They state that to verify that $L(\mathcal{H})$ is co-observable w.r.t. $L(\mathcal{G})$, we just need to verify that $L(\mathcal{H})$ is co-observable w.r.t. the language generated by some subsystem of $\mathcal{G}$.

2. By Propositions 3.1.2 and 3.1.3, we know that we in order to verify that $L(\mathcal{H})$ is co-observable w.r.t. $L(\mathcal{G})$, it is sufficient to verify that for each $H_j \in \mathcal{H}$, $j \in \{1, \ldots, w\}$, $L(H_j)$ is co-observable w.r.t. $L(\mathcal{G})$.

3. To implement the above steps, we typically end up verifying that the language generated by some subset of $\mathcal{H}$, say $L(\mathcal{H}') = L(H_{j_1})||\ldots||L(H_{j_a})$, for $\{j_1, ..., j_a\} \subseteq \{1, \ldots, w\}$, is co-observable w.r.t. the language generated by some subset of $\mathcal{G}$, say $L(\mathcal{G}') = L(G_{k_1})||\ldots||L(G_{k_b})$, for $\{k_1, ..., k_b\} \subseteq \{1, \ldots, v\}$. We then simply need to find enough suitable subsets of $\mathcal{H}$ so that each $H_j \in \mathcal{H}$, $j \in \{1, \ldots, w\}$, is included in at least one subset.

4. According to Propositions 3.1.5 and 3.1.6, if we have a component H_j of $\mathcal{H}$ such that $L(H_j)$ is co-observable w.r.t. the language generated by some subset of $\mathcal{G}$, then we can treat H_j as a plant automaton to be synchronized with $\mathcal{G}$.

5. If we find a counter example C for $L(\mathcal{H})$ with respect to $L(\mathcal{G})$, we know that $L(\mathcal{H})$ is NOT co-observable w.r.t. $L(\mathcal{G})$.

6. If we find a counter example C for $L(H_j)||\cdots||L(H_w)$ with respect to $L(\mathcal{G})||L(H_1)||\cdots||L(H_{j-1})$ (i.e. we were treating specifications $H_1, \cdots, H_{j-1}$ as plant components after applying Propositions 3.1.5 and 3.1.6), we can apply Proposition 3.1.7 and conclude that $L(\mathcal{H})$ is NOT co-observable w.r.t. $L(\mathcal{G})$.

We focus now on some specific aspects of Algorithm 1. Notice that at line 3, the algorithm terminates successfully when all elements of $\mathcal{H}$ have been considered. Initially, on line 4, an element of $\mathcal{H}$ is selected.

On line 7, we begin by checking to see if $L(\mathcal{H}')$ is co-observable w.r.t. the set of all finite strings, i.e., $L(G_{\Sigma^*}) = \Sigma^*$. If this is the case, then by Proposition 3.1.1,

Algorithm 1 Incremental Co-observability Verification

1: **input** plant automata $\mathcal{G} = \{G_1, \ldots, G_v\}$,
specification automata $\mathcal{H} = \{H_1, \ldots, H_w\}$
2: **output true** if $L(\mathcal{H})$ is co-observable w.r.t. $L(\mathcal{G})$
false if $L(\mathcal{H})$ is not co-observable w.r.t. $L(\mathcal{G})$
3: **while** $\mathcal{H} \neq \emptyset$ **do**
4: Choose $H_j \in \mathcal{H}$
5: Let $\mathcal{H}' = \{H_j\}$
6: Let $\mathcal{G}' = \{G_{\Sigma^*}\}$
7: **while** $L(\mathcal{H}')$ is not co-observable w.r.t. $L(\mathcal{G}')$ **do**
8: Let C be a counter example w.r.t. $L(\mathcal{H}')$ and $L(\mathcal{G}')$;
9: Find a component $G_k \in \mathcal{G} \setminus \mathcal{G}'$ or $H_m \in \mathcal{H} \setminus \mathcal{H}'$ that rejects C
10: **if** there is no such component **then**
11: **return false**
12: **else if** the component found in line 9 is in $\mathcal{G}$ **then**
13: Let $\mathcal{G}' = \mathcal{G}' \cup \{G_k\}$
14: **else**
15: Let $\mathcal{H}' = \mathcal{H}' \cup \{H_m\}$
16: **end if**
17: **end while**
18: Let $\mathcal{H} = \mathcal{H} \setminus \mathcal{H}'$
19: Let $\mathcal{G} = \mathcal{G} \cup \mathcal{H}'$
20: **end while**
21: **return true**

$L(\mathcal{H}')$ is also co-observable w.r.t. $L(\mathcal{G})$. Otherwise, we obtain a counter example C that illustrates that $L(\mathcal{H}')$ is not co-observable w.r.t. $L(\mathcal{G}')$. In this case there are two options:

(a) Look for a plant automaton $G_k \in \mathcal{G} \setminus \mathcal{G}'$ that rejects C. If we find such an automaton, in the next iteration, we will check to see if $L(\mathcal{H}')$ is co-observable w.r.t. $L(\mathcal{G}')||L(G_k)$

(b) Look for a specification automaton $H_m \in \mathcal{H} \setminus \mathcal{H}'$ that rejects C. If we find such an automaton, in the next iteration we will check to see if $L(\mathcal{H})||L(H_m)$ is co-observable w.r.t. $L(\mathcal{G}')$.

Steps (a) and (b) are repeated until such time as co-observability is verified for $L(\mathcal{H}')$ w.r.t. $L(\mathcal{G}')$, i.e., no more counter examples are found, or all remaining components in $\mathcal{G} \setminus \mathcal{G}'$ and $\mathcal{H} \setminus \mathcal{H}'$ accept C. In this latter situation, this means that there exist disablement decisions that cannot be taken correctly by any of the relevant controllers, i.e., $L(\mathcal{H})$ is not co-observable, and the algorithm terminates with an output of false. As long as co-observability continues to hold as each successive element of $\mathcal{H}$ is examined, the algorithm eventually terminates with an output of true. Note that on Line 19, after verifying that $L(\mathcal{H}')$ is co-observable w.r.t. $L(\mathcal{G}')$, $\mathcal{H}'$ is removed from $\mathcal{H}$ and added to the set of plant automata. As such, the elements of $\mathcal{H}'$ are treated as plant components for the remainder of the algorithm, according to Proposition 3.1.5.

On line 8, we acquire a counter example $C = (\sigma, t, t_1, \ldots, t_n)$ that encodes a violation of co-observability for $L(\mathcal{H}')$ and $L(\mathcal{G}')$. The question of which counter example to choose gives rise to a set of heuristics. We are interested in two strategies. For the first strategy, we choose C such that $|t|$ is the shortest length of any of the valid counter examples that we find. For the second strategy, we choose C such that

$|t_i|$, for $i \in I_c(\sigma)$, is the longest length of any of the t_i in any of the other valid counter examples that we find.

Line 9 also introduces some heuristics for the execution of the algorithm. As this is a strategy "guided by counter examples", regardless of how we choose the counter example, we have a choice as to which component to select, should it reject the counter example $C = (\sigma, t, t_1, \ldots, t_n)$, according to Definition 3.2.2 or Definition 3.2.3. We consider the following heuristics for choosing a component:

(i) always choose a plant component over a specification component;

(ii) always choose a specification component over a plant component;

(iii) randomly choose a component;

(iv) choose a component that rejects C the "earliest";

(v) choose a component that rejects C the "latest";

(vi) choose a component that has the smallest number of states; and,

(vii) choose a component that has the smallest number of transitions.

The last four heuristics are from [BMM04], and the others are new. We examine the effect of these heuristics in the next chapter when we introduce our first case study, the data transmission problem.

If all the relevant components on line 9 accept C, then we have found a violation of co-observability for $L(\mathcal{H})$ w.r.t. $L(\mathcal{G})$ and the algorithm will terminate and return an output of **false**. If, though, we do find a component that rejects C, then we want to incorporate it into either the interim specification language $L(\mathcal{H}')$ or the interim plant language $L(\mathcal{G}')$ that we continue to verify in the loop at line 7.

Assuming that eventually line 7 is not satisfied, i.e., $L(\mathcal{H}')$ is co-observable w.r.t. $L(\mathcal{G}')$, then the algorithm continues examining the remaining specification automata until either line 3 evaluates to false and co-observability is asserted, or line 10 evaluates to true and co-observability fails.

3.2.1 Computational Complexity

We will use the following notation to describe the complexity of Algorithm 1:

(i) $|\mathcal{G}|$, respectively $|\mathcal{H}|$, is the number of plant (specification) components for the system;

(ii) $|Q|$ is the size of the largest state set amongst the plant and specification components;

(iii) n is the number of controllers of the system; and,

(iv) $|\mathcal{Q}|$ is the size of the state set of the verification structure, which best case is of size $|Q|^{n+1}$ and in the worst case is $(|Q|^{|G|+|H|})^{n+1}$, which is the value we will assume for the analysis.

The algorithm is governed by two loops (line 3 and line 7): the outer loop is executed at most $|\mathcal{H}|$ times, whereas the inner loop is executed a maximum of $|\mathcal{G}| + |\mathcal{H}|$ times. The bulk of the effort is contained in lines 7 through 9.

Line 7 involves the monolithic verification of co-observability. We have implemented the strategy described in [Ric08], where verifying co-observability requires a breadth-first search of the verification structure. The complexity of this classical search is $O(|V| + |E|)$, where V is the set of vertices and E is the set of edges in

the structure being searched. For our purposes, this corresponds to the state set and the transition function of the verification structure: $|V| = |\mathcal{Q}| \leq |Q^{|\mathcal{G}|+|\mathcal{H}|}|^{n+1}$ and $|E| = |\mathcal{Q}||\Sigma|^{n+1} \leq (|Q^{|\mathcal{G}|+|\mathcal{H}|}||\Sigma|)^{n+1}$; thus the complexity of monolithic verification of co-observability is $O((|Q^{|\mathcal{G}|+|\mathcal{H}|}||\Sigma|)^{n+1})$.

Note that the first time line 7 is executed, we require the construction of a structure that has a maximum of $|Q|^{n+1}$ states. With each iteration of the inner loop, we build a verification structure that involves consideration of one more component. Thus, on the second iteration we build a structure with a maximum of $|Q||Q|^{n+1}$ states up until, in the worst-case, the $(|\mathcal{G}| + |\mathcal{H}|)$th, i.e., last, iteration where we could have as many as $(|Q|^{|\mathcal{G}|+|\mathcal{H}|})^{n+1}$ states. We can rewrite this as $(|Q| + |Q|^2 + \ldots + |Q|^{|\mathcal{G}|+|\mathcal{H}|})^{n+1}$. The sum of the number of states is actually in the form of a well-known geometric series and can be rewritten as the $\frac{(1-|Q|^{|\mathcal{G}|+|\mathcal{H}|})}{1-|Q|} = O(|Q||Q|^{|\mathcal{G}|+|\mathcal{H}|-1}) = O(|Q|^{|\mathcal{G}|+|\mathcal{H}|})$. Thus we can use $(|Q|^{|\mathcal{G}|+|\mathcal{H}|})^{n+1}$ as our upper bound for the number of states in the verification structure, as noted in the previous paragraph.

To find a counter example $C = (\sigma, t, t_1, \ldots, t_n)$ at line 8 requires a search of the verification structure with $|\Sigma|\ |\mathcal{Q}|$ steps. Finally, at line 9 the search for a component that rejects C requires a search, in the worst case, of all the components in the system, i.e., $|\mathcal{G}| + |\mathcal{H}|$. This is followed by a search to determine acceptance or rejection of all the elements of C, i.e., $|I_c(\sigma)| + 1$, that, again in the worst case, requires a search of the length of the counter example, which could involve all states in the largest component, i.e., $|Q|$. Thus we have $(n + 1)|Q|(|\mathcal{G}| + |\mathcal{H}|)$ steps for line 9. But the complexity of these steps are insignificant in comparison to the effort at line 7.

Combining all of this information, and ignoring the minor contributions of the overhead for the set operations in lines 13, 15, 18 and 19, we have an overall (worst-case) complexity for Algorithm 1 of $O(|\mathcal{H}|(|\mathcal{G}| + |\mathcal{H}|)(|Q^{|\mathcal{G}|+|\mathcal{H}|}||\Sigma|)^{n+1})$.

In contrast, the complexity for verifying co-observability with the monolithic structure has a (worst-case) complexity of

$$O((|Q_1||Q_2|\dots|Q_v|)^{n+1} * |\Sigma|^{n+1}) = O((|Q_1||Q_2|\dots|Q_v||\Sigma|)^{n+1}) \leq O((|Q|^{|\mathcal{G}|}|\Sigma|)^{n+1}).$$

Based on worst-case complexity analysis, it is not difficult to see that if given the worst possible input, Algorithm 1 multiplies the monolithic complexity by a quadratic factor in the number of components. This is because of the possibility of entering the loop at line 7 and having to compute successive verification structures in each iteration, up to and including the computation of the monolithic structure. But it is not clear that the systems we want to test will always generate the worst possible input that will force the algorithm to perform worse than the monolithic algorithm.

To ameliorate this situation, we focus on the performance of the algorithm. We have incorporated heuristics to organize the input in such a way as to steer the algorithm towards the best possible case: when there is only one iteration of loop 7 per component in $\mathcal{H}$. We discuss the effect that heuristics have on the performance of our algorithm in the next chapter.

Chapter 4

Case Study: Co-observability of the Data Transmission Problem

The data transmission problem [BSW69] has been widely studied in the literature of communication protocols and can be stated as follows. Consider two agents communicating asynchronously across a faulty channel: a *sender* and a *receiver*. The sender is responsible for transmitting a message to the receiver in the form of a sequence of data packets, retrieved from the source host. The receiver then passes the packets on to a target host such that the packets are passed on in the correct order and no packets are accidentally sent more than once. To distinguish a new packet from a re-transmitted packet, the sender may alternate appending a control bit with a value of 0 or 1 to the data. Subsequently, the receiver can acknowledge that it received a message appended with a 0 or 1.

In the supervisory control framework, the data transmission problem has been modeled in [CDFV88, Rud92, RW92a, WvS96a]. The motivation in [Rud92] was to synthesize, as a decentralized control solution, the solution to the data transmission

problem, namely, the alternating bit protocol, and a selection of plant automata of varying levels of detail were presented to explore this problem. The model we use here is adapted from [Rud92, RW92a]. Our model differs slightly in that we have added two additional specifications, automata that were previously used as possible plant models in [Rud92]. Furthermore, our motivation is purely to illustrate our algorithm for the incremental verification of co-observability.

4.1 DTP as a Decentralized DES

The event set for the entire problem is given by

$$\begin{aligned}\Sigma \;=\; \{&\mathsf{getFrame}, \mathsf{send}_0, \mathsf{send}_1, \mathsf{rcvAck}_0, \mathsf{rcvAck}_1, \mathsf{loss}, \mathsf{sendAck}_0, \mathsf{sendAck}_1,\\ &\mathsf{rcv}_0, \mathsf{rcv}_1, \mathsf{passToHost}\},\end{aligned}$$

where the events are described as follows.

$\mathsf{getFrame}$: get new data frame from source host;

send_0: send data with control bit set to 0;

send_1: send data with control bit set to 1;

rcv_0: receive data with control bit set to 0;

rcv_1: receive data with control bit set to 1;

$\mathsf{passToHost}$: data frame passed from receiver to target host;

$\mathsf{sendAck}_0$: acknowledge data with control bit set to 0;

$\mathsf{sendAck}_1$: acknowledge data with control bit set to 1;

rcvAck_0: receive acknowledgement with control bit set to 0;

rcvAck_1: receive acknowledgement with control bit set to 1;

loss: contents in the channel are lost.

For this example, we have $I = \{1, 2\}$, representing a controller for the sender side (controller 1) and one for the receiver side (controller 2). The observable and controllable partitions of Σ are as follows:

$\Sigma_{o,1} := \{\mathsf{getFrame}, \mathsf{send}_0, \mathsf{send}_1, \mathsf{rcvAck}_0, \mathsf{rcvAck}_1, \mathsf{loss}\}$,

$\Sigma_{o,2} := \{\mathsf{sendAck}_0, \mathsf{sendAck}_1, \mathsf{rcv}_0, \mathsf{rcv}_1, \mathsf{passToHost}\}$,

$\Sigma_{c,1} := \{\mathsf{getFrame}, \mathsf{send}_0, \mathsf{send}_1\}$,

$\Sigma_{c,2} := \{\mathsf{sendAck}_0, \mathsf{sendAck}_1, \mathsf{passToHost}\}$.

Note that the controllers have no jointly-observable events, i.e., $\Sigma_{o,1} \cap \Sigma_{o,2} = \emptyset$, nor do they have jointly-controllable events, i.e., $\Sigma_{c,1} \cap \Sigma_{c,2} = \emptyset$. Also note, that at each state of the automata described in this chapter, we assume (even when not explicitly shown) self-loops of events in Σ minus the events that appear in the description of each component, e.g., see Figure 4.4 where the self-loops are present.

Figure 4.1 contains the automaton SENDER, the first of three plant components. This component captures the more general behaviour of getting new data, sending it to the receiver with either a 0 or 1 appended. Then either the data is lost, in which case the data is re-sent or an acknowledgement from the receiver arrives, which can trigger a re-sending of the data or, if the correct acknowledgement is received, repeat the transmission cycle with new data.

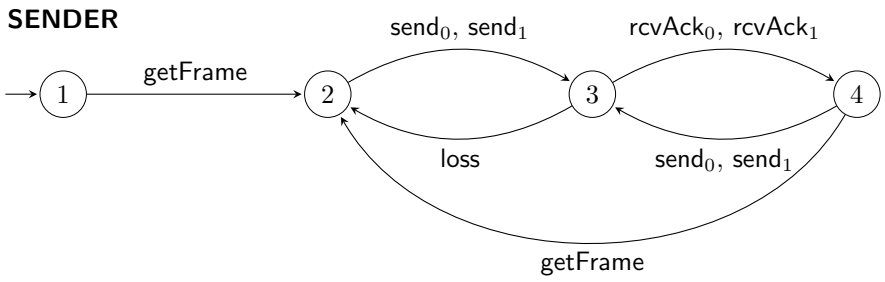


Figure 4.1: A Plant Automaton for a Sender.

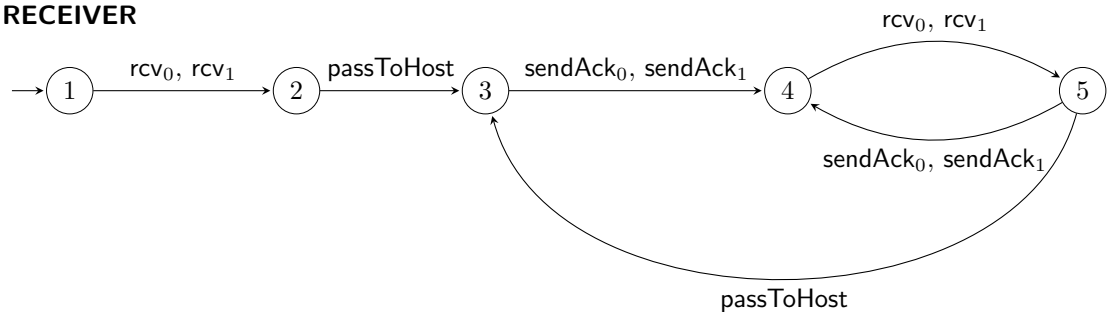


Figure 4.2: A Plant Automaton for a Receiver.

Figure 4.2 features the automaton RECEIVER, another plant component. A receiver acquires data that has either a 0 or 1 appended to it. The data is passed to the host and an acknowledgement of the type of data received is directed to the sender. The subsequent reception of data will either lead to an acknowledgement being resent or the data being passed to the host, whereby the reception cycle continues.

Figure 4.3 features the automaton CHANNEL, the final plant component. The communication channel simply describes how data sent from the sender to the receiver is either successfully delivered or lost. In the event that the data is lost, the sender retransmits the data, otherwise, the receiver accepts the data and an acknowledgement of receipt is sent. Then the transmission cycle begins again with new data.

The behaviour of the whole plant system G is represented by the synchronous product of the languages generated by the plant components: $L(G) = L(\mathsf{SENDER})||$

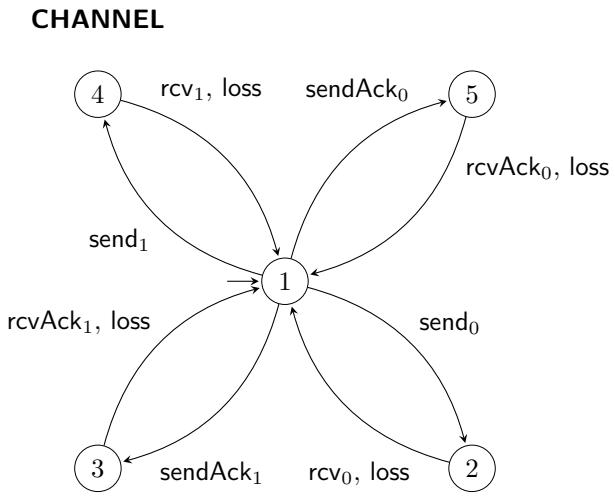


Figure 4.3: A Plant Automaton for a Communication Channel

$L(\mathsf{RECEIVER})||L(\mathsf{CHANNEL})$.

While $L(G)$ describes the language of the uncontrolled system, we also have some behaviour that we want the decentralized controllers to jointly enforce via their control decisions. In particular, we first want to ensure that a data packet is acquired from the source host by the sender before it is passed along to the target host by the receiver. We do not care about the order in which the remaining events in Σ occur, and this is captured by the selfloops at states 1 and 2 of $\mathsf{SPECIFICATION}_1$ in Figure 4.4.

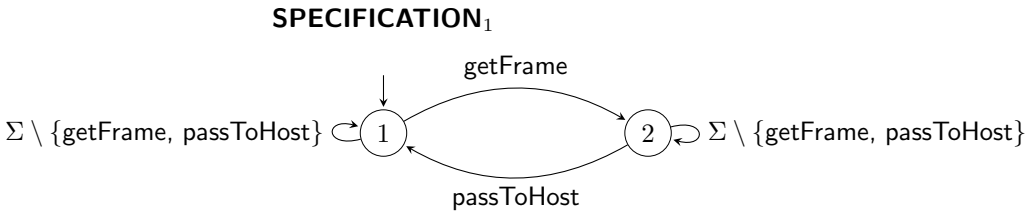


Figure 4.4: A Specification Automaton for the Delivery of Data.

The remaining two specification automata prescribe the behaviour of the sender and the receiver. Specifically, the general behaviour of the sender, as described by the plant automaton SENDER in Figure 4.1, is updated in $\mathsf{SPECIFICATION}_2$ in Figure 4.5

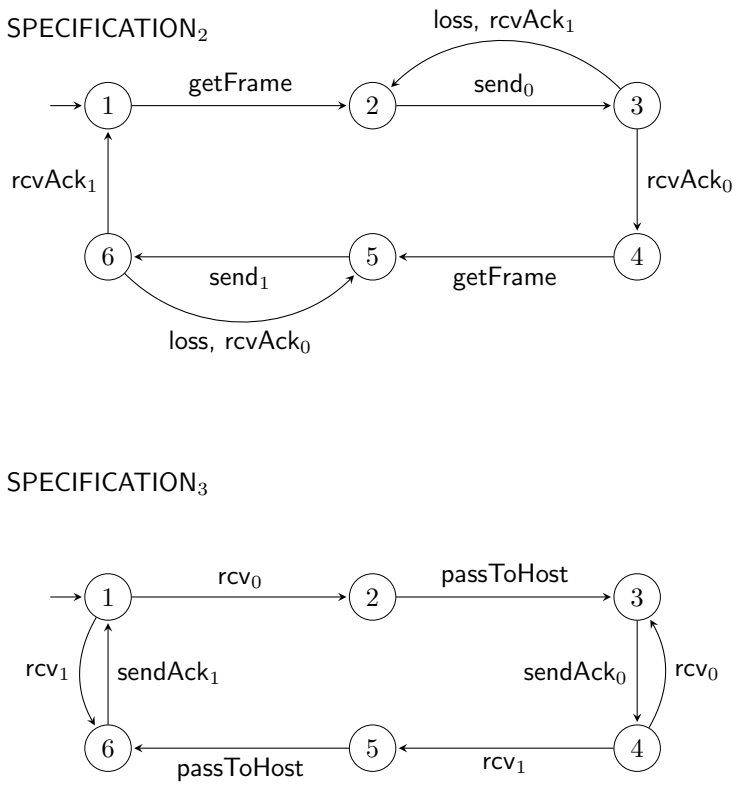


Figure 4.5: The Specification Automata for Desired Behaviour of a Sender (SPECIFICATION$_2$) and a Receiver (SPECIFICATION$_3$).

(top) by prescribing the explicit alternation of messages with attached control bits of 0 or 1. A new data packet is not retrieved from the source host until the correct sequence of acknowledgements, in the face of data loss, is acquired from the sender. Similar explicit behaviour for the receiver is described by automaton SPECIFICATION$_3$ (bottom of Figure 4.5): a data packet with a control bit of 0 (respectively, 1) is passed to the target host after which the receiver acknowledges that it received the data; if the next packet it receives is appended with the same control bit, it retransmits the acknowledgement; otherwise it passes along the new data to the target host and the process continues.

The global specification K is captured by the synchronous product of the three specification automata: $K = \mathsf{SPECIFICATION}_1||\mathsf{SPECIFICATION}_2||\mathsf{SPECIFICATION}_3$. Subsequently, $L(K) = L(\mathsf{SPECIFICATION}_1)||L(\mathsf{SPECIFICATION}_2)||L(\mathsf{SPECIFICATION}_3)$.

4.2 Incremental Verification of Co-observability of DTP

We want to verify whether the global specification $L(K)$ is co-observable w.r.t. $L(G)$. Thus, we will use Algorithm 1 with our collection of plant and specification automata, to incrementally verify co-observability of $L(K)$.

According to line 1, let $\mathcal{G} = \{\mathsf{SENDER}, \mathsf{RECEIVER}, \mathsf{CHANNEL}\}$ and $\mathcal{H} = \{\mathsf{SPECIFICATION}_1, \mathsf{SPECIFICATION}_2, \mathsf{SPECIFICATION}_3\}$.

At line 3, it is clear that $\mathcal{H} \neq \emptyset$, so we continue and (arbitrarily) choose $\mathsf{SPECIFICATION}_2$. Thus $\mathcal{H}' = \{\mathsf{SPECIFICATION}_2\}$ and $\mathcal{G}' = \{G_{\Sigma^*}\}$. In the case of $\mathsf{SPECIFICATION}_2$, since all the events in the automaton are observable to controller 1 (i.e., there are no observational ambiguities because of partial observations for controller 1) and all the controllable events in $\mathsf{SPECIFICATION}_2$ are controlled by controller 1, $L(\mathcal{H}')$ is trivially co-observable w.r.t. $L(\mathcal{G}')$. So we continue to line 18 and update $\mathcal{H} = \{\mathsf{SPECIFICATION}_1, \mathsf{SPECIFICATION}_3\}$, whereas now $\mathcal{G} = \{\mathsf{SENDER}, \mathsf{RECEIVER}, \mathsf{CHANNEL}, \mathsf{SPECIFICATION}_2\}$.

We return to line 3 again, and as $\mathcal{H}$ is still not empty, we continue by (arbitrarily) choosing $\mathsf{SPECIFICATION}_3$ from $\mathcal{H}$. Thus we now want to see if $L(\mathsf{SPECIFICATION}_3)$

is co-observable w.r.t. $L(\mathcal{G}')$, i.e., Σ^*. As before, since all events in SPECIFICATION$_3$ are observable to controller 2, and all the controllable events in this automaton are controlled by controller 2, $L(\mathcal{H}')$ is trivially co-observable w.r.t. $L(\mathcal{G}')$. At this point we now have $\mathcal{H} = \{\text{SPECIFICATION}_1\}$ and $\mathcal{G} = \{\text{SENDER, RECEIVER, CHANNEL, SPECIFICATION}_2, \text{SPECIFICATION}_3\}$

Finally, we have only one element left in $\mathcal{H}$, so we choose it: SPECIFICATION$_1$. Now we must verify that $L(\text{SPECIFICATION}_1)$ is co-observable w.r.t. Σ^*. In this case, the specification automaton involves partial observation and controllable events for both controllers, so we use the monolithic strategy to verify co-observability, which returns false.

- One counter example for this pair of languages is $C = (\text{passToHost}, \epsilon, \text{-}, \text{getFrame})$.
- Since $I_c(\text{passToHost}) = \{2\}$ and it is the case that $P_2(\epsilon) = P_2(\text{getFrame}) = \epsilon$, controller 2 will not be able to take the correct control decision about passToHost after the occurrence of either of these sequences: enable after getFrame occurs but disable otherwise.
- According to line 9, we need to find a component in $\mathcal{G} \setminus \mathcal{G}' = \{\text{SENDER, RECEIVER, CHANNEL, SPECIFICATION}_2, \text{SPECIFICATION}_3\}$ or $\mathcal{H} \setminus \mathcal{H}' = \emptyset$ that rejects C.
- C is rejected by both RECEIVER and SPECIFICATION$_3$. As we only need to choose one of these automata, we choose SPECIFICATION$_3$.
- By line 13 update $\mathcal{G}' = \{G_{\Sigma^*}\} \cup \{\text{SPECIFICATION}_3\}$.

We need to check whether $L(\text{SPECIFICATION}_1)$ is co-observable w.r.t. $L(\mathcal{G}') =$

$L(G_{\Sigma^*})||L(\mathsf{SPECIFICATION}_3) = L(\mathsf{SPECIFICATION}_3)$. Again, we perform the monolithic co-observability test which returns false.

- One counter example for this pair of languages is $C = (\mathsf{getFrame}, \mathsf{getFrame}, \mathsf{getFrame}\ \mathsf{rcv}_0\ \mathsf{passToHost}, -)$.
- Since $I_c(\mathsf{getFrame}) = \{1\}$ and $P_1(\mathsf{getFrame}) = P_1(\mathsf{getFrame}\ \mathsf{rcv}_0\ \mathsf{passToHost}) = \mathsf{getFrame}$, controller 1 cannot take the correct control decision for getFrame.
- We must find a component in $\mathcal{G} \setminus \mathcal{G}' = \{$SENDER, RECEIVER, CHANNEL, $\mathsf{SPECIFICATION}_2\}$ or $\mathcal{H} \setminus \mathcal{H}' = \emptyset$ that rejects C.
- C is rejected by both SENDER and $\mathsf{SPECIFICATION}_2$. As we only need to choose one of these automata, we choose $\mathsf{SPECIFICATION}_2$.
- Update $\mathcal{G}' = \{G_{\Sigma^*}, \mathsf{SPECIFICATION}_3\} \cup \{\mathsf{SPECIFICATION}_2\}$.

We need to check whether $L(\mathsf{SPECIFICATION}_1)$ is co-observable w.r.t. $L(\mathcal{G}') = L(G_{\Sigma^*})||L(\mathsf{SPECIFICATION}_3)||L(\mathsf{SPECIFICATION}_2) = L(\mathsf{SPECIFICATION}_3)||$ $L(\mathsf{SPECIFICATION}_2)$. We perform the monolithic co-observability test, which again returns false.

- One counter example for this pair of languages is $C =$ (getFrame, getFrame send_0 rcvAck_0, getFrame send_0 rcv_0 passToHost $\mathsf{sendAck}_0$ recAck_0, -).
- Since $I_c(\mathsf{getFrame}) = \{1\}$ and $P_1(\mathsf{getFrame}\ \mathsf{send}_0\ \mathsf{rcvAck}_0) = P_1(\mathsf{getFrame}\ \mathsf{send}_0\ \mathsf{rcv}_0\ \mathsf{passToHost}\ \mathsf{sendAck}_0\ \mathsf{recAck}_0) = \mathsf{getFrame}\ \mathsf{send}_0\ \mathsf{rcvAck}_0$, controller 1 cannot take the correct control decision for getFrame.
- We must find a component in $\mathcal{G} \setminus \mathcal{G}' =$ {SENDER, RECEIVER, CHANNEL} or $\mathcal{H} \setminus \mathcal{H}' = \emptyset$ that rejects C.

- In this case, CHANNEL is the only component that rejects C, so we update $\mathcal{G}' = \{G_{\Sigma^*}, \mathsf{SPECIFICATION}_3, \mathsf{SPECIFICATION}_2\} \cup \{\mathsf{CHANNEL}\}$.

We need to check whether $L(\mathsf{SPECIFICATION}_1)$ is co-observable w.r.t. $L(\mathcal{G}') = L(G_{\Sigma^*})||L(\mathsf{SPECIFICATION}_3)||L(\mathsf{SPECIFICATION}_2)||L(\mathsf{CHANNEL})$. Again, we perform the monolithic co-observability test which, this time, returns true.

At this point, we continue to line 18 and update $\mathcal{H} = \mathcal{H} \setminus \{\mathsf{SPECIFICATION}_1\} = \emptyset$ and update $\mathcal{G} = \mathcal{G} \cup \{\mathsf{SPECIFICATION}_1\}$. Then continuing to line 3, which evaluates now to false, we drop down to line 21 and return true, indicating that the system is overall co-observable.

4.3 Heuristics

In keeping with the incremental verification strategy introduced in [BMM04], our algorithm is directed by counter examples (line 8 of Algorithm 1). Ideally, we want to steer our algorithm towards the best-case computational scenario and avoid performing the monolithic test for the entire system. We introduce two sets of heuristics: one to guide the selection of the counter example C and the other to guide the selection of the component that rejects C. Our purpose is to investigate the effect of a number of heuristics in order to gauge their effectiveness.

(a) ShortC. Choose $C = (\sigma, t, t_1, \ldots, t_n)$ such that $|t|$ is the shortest length of any of the other valid counter examples.

(b) LongC. Choose $C = (\sigma, t, t_1, \ldots, t_n)$ such that $|t_i|$, for $i \in I_c(\sigma)$, is the longest of any of the other t_i in any additional valid counter examples.

While this is certainly not a comprehensive list, we were interested in determining whether or not the choice of the size of the counter example would affect performance.

When we have a choice as to which component rejects the counter example (line 9), it may be possible to reduce the total computation required to incrementally verify co-observability by selecting components that exhibit particular characteristics. To that end we introduce some heuristics to guide the behaviour of our algorithm.

(i) PlantOverSpec. This heuristic selects an element of $\mathcal{G} \setminus \mathcal{G}'$ over an element of $\mathcal{H} \setminus \mathcal{H}'$.

(ii) SpecOverPlant. This heuristic selects an element of $\mathcal{H} \setminus \mathcal{H}'$ over an element of $\mathcal{G} \setminus \mathcal{G}'$.

(iii) Alternating. This heuristic alternately selects a component that rejects C from $(\mathcal{G} \setminus \mathcal{G}')$ first, and then $(\mathcal{H} \setminus \mathcal{H}')$ throughout the run of the algorithm.

(iv) FirstMatch. This heuristic chooses the first component that rejects C.

(v) EarlyReject. This heuristic chooses the component that rejects C in the fewest number of steps (i.e., events).

(vi) LateReject. This heuristic chooses the component that rejects C in the most number of steps.

(vii) MinStates. This heuristic selects the component with the smallest number of states that rejects C.

(viii) MinTransitions. This heuristic selects the component with the smallest number of transitions that rejects C.

There is one final pair of heuristics that is related specifically to the Java implementation of the algorithm: when creating the list of elements in $\mathcal{G} \cup \mathcal{H}'$ (line 19), those components that were originally in $\mathcal{H}$ are either inserted at the beginning of the list or the end of the list.

- SpecsStart: $\mathcal{H}'$ components are inserted at the start of the data structure storing $\mathcal{G}$.
- SpecsEnd: $\mathcal{H}'$ components are inserted at the end of the data structure storing $\mathcal{G}$.

These heuristics have been implemented and their effect on the overall computation required to incrementally verify co-observability of the DTP case study is explored in the next section.

4.4 Experimental Results

The trials for the case study were conducted on an iMac with 2.7 GHz Intel Core i5 processor and 8 GB 1600 MHz DDR3 memory.

We first examine the time to verify co-observability of DTP using the monolithic strategy. In this case, we tested whether or not $L(\mathsf{SPECIFICATION}_3)\,||L(\mathsf{SPECIFICATION}_1)||$ $L(\mathsf{SPECIFICATION}_2)$ was co-observable w.r.t. $L(\mathsf{RECEIVER})\,||L(\mathsf{CHANNEL})||L(\mathsf{SENDER})$. The total time for the test was 132 seconds. The size of the monolithic automaton was 114 states, whereas the size of the monolithic verification structure was 7684 states.

In contrast, the best time for Algorithm 1 used component ordering $\mathcal{G} = \{\mathsf{RECEIVER}, \mathsf{CHANNEL}, \mathsf{SENDER}\}$ and $\mathcal{H} = \{\mathsf{SPECIFICATION}_3, \mathsf{SPECIFICATION}_1, \mathsf{SPECIFICATION}_2\}$.

We used LongC to select the counter example. The first heuristic used to select a rejecting component was PlantOverSpec. This heuristic produced a choice of components, so we used EarlyReject to make the final selection. Finally, the building of $\mathcal{G}$ in line 19 of the algorithm was governed by the SpecsFirst heuristic. The runtime was 3.24 seconds. This represents a speedup of 40.74 times, compared to the monolithic runtime. The $\mathcal{U}$ structure had a maximum number of 4209 states.

The worst time for Algorithm 1 used component ordering $\mathcal{G} = \{\mathsf{SENDER}, \mathsf{CHANNEL}, \mathsf{RECEIVER}\}$ and $\mathcal{H} = \{\mathsf{SPECIFICATION}_3, \mathsf{SPECIFICATION}_1, \mathsf{SPECIFICATION}_2\}$. To select the counter example, we used heuristic LongC. Then the rejecting component was chosen using SpecOverPlant. This heuristic produced a choice of components, so we used LateReject to make the final selection. Additionally, the heuristic SpecsEnd governed the operation of line 19. In this case, the runtime was 168.9 seconds. The $\mathcal{U}$ structure had a maximum number of 10323 states.

We performed exhaustive testing with respect to the ordering of $\mathcal{G}$ and $\mathcal{H}$. For each test configuration of heuristics, we performed a set of 36 tests covering the possible ordering of plants and specifications. A summary of our results is presented in Tables 4.1 to 4.3. The minimum runtime represented the ordering that performed best, and the worst runtime was the ordering that performed worst. The average runtime was the average time taken across the 36 test variations for a specific configuration of heuristics. We also track the average number of states in the computation of the larger of the two automata that generated $L(\mathcal{H}')$ and $L(\mathcal{G}')$. This affects the size of the monolithic verification structure $\mathcal{U}$ that is computed to perform the co-observability test in line 7 of the algorithm.

Tables 4.1, 4.2, and 4.3 represent the results from our suite of tests with the

primary tests concerning the primary heuristics PlantOverSpec, SpecOverPlant, and Alternating, respectively. Each table is then further broken down based on heuristics regarding counter example selection, ShortC and LongC, followed by the ordering heuristic for where specifications are added to the list of plant components, namely SpecsStart versus SpecsEnd, and, finally, the remaining five heuristics concerned with selecting a rejecting component.

Examining the results, it is immediately clear that the initial ordering of the plant and specification components plays a significant role in the runtime. For the test with heuristics PlantOverSpec, LongC, SpecsStart, and EarlyReject, we saw a minimum runtime of 3.24 seconds, and a maximum runtime of 149 seconds: the only difference between these two cases is the initial ordering of the components. A heuristic to select the initial component would be highly desirable addition to the current heuristics. That being said, we also note that the combination of heuristics SpecOverPlant and SpecsStart produced a consistent runtime of about 51-53 seconds, independent of the initial ordering. This did not produce the 40.74 times speedup of the best configuration, but it did produce a consistent speedup of about 2.5 times.

After considering the effect of varying the three primary heuristics, we see that they produce similar minimum, maximum, and average run times, with PlantOverSpec producing the fastest runtime. Overall, SpecOverPlant produced smaller maximum runtimes, while Alternating produced the largest.

We next examine the effect of choosing a counter example via ShortC versus LongC. Note that with respect to our three primary heuristics, LongC gives us our shortest runtime for all three; however, with respect to average and maximum runtime, the difference between the two heuristics is negligible.

As far as the effect of SpecsStart versus SpecsEnd is concerned, we see that PlantOverSpec tends to produce the slowest runtimes for SpecsStart while SpecOverPlant produced both its fastest and slowest runtimes with SpecsEnd.

Finally, we examine the effect of the remaining heuristics that govern how to choose a rejecting component. After the previously mentioned heuristics have been applied, it is difficult to discern a significant difference between the heuristics. Although we do note that LateReject has a tendency to produce the worst runtimes.

One thing that is clear from our results: we need to test our heuristics on more examples that consist of many more (larger) components to get a better appreciation for the effect of the various heuristics. By studying a wider variety of examples, it may be the case that certain heuristics are better suited (i.e., encourage better performance) for particular types of systems (e.g., high number of components with many observable events in common versus high number of components with few observable events in common).

We note that in Chapter 6, we present an HIDSC example that contains a high-level subsystem and three low-level subsystems. Each low level was identical up to relabeling. The state size of the high level was 3120, whereas each low level was 550. For both the high level and the low level, the monolithic algorithm ran for 5 hours without completing, so we stopped them. Our incremental algorithm ran in 424.78 seconds for the high level and 4.76 seconds for the low level. This was the fastest time of all the heuristic configurations. We only were able to run it for a single ordering of the components due to time constraints.

Table 4.1: The DTP problem using PlantOverSpec heuristic as primary rule.

heuristic 2	heuristic 3	heuristic 4	avg. iter.	avg. states in $max(\mathcal{G}', \mathcal{H}')$	avg. states in $\mathcal{U}$	min time (s)	avg time (s)	max time (s)
ShortC	SpecsStart	FirstMatch	4	25	4324.5	51.2	75.7	150
		MinTrans	4	25	4324.5	51.2	75.8	150
		MinStates	4	25	4324.5	51.2	75.7	150
		EarlyReject	4	25	4324.5	51.2	83	151
		LateReject	4	25	4324.5	51.2	68.5	151
	SpecsEnd	FirstMatch	5	26.33	5427	52.6	74.7	97.1
		MinTrans	5	26.33	5427	52.6	74.8	96.7
		MinStates	5	26.33	5427	52.6	74.8	97.3
		EarlyReject	5	26.33	5427	52.9	89.2	97.3
		LateReject	5	32.83	5427	52.6	107	161
LongC	SpecsStart	FirstMatch	4	31.83	3496.89	4.31	65.4	152
		MinTrans	4	31.83	3496.89	4.31	65.5	152
		MinStates	4	31.83	3496.89	4.31	65.5	152
		EarlyReject	4	24.5	3661.5	3.24	44.4	149
		LateReject	4	33	2936.33	4.96	68.0	152
	SpecsEnd	FirstMatch	5	26.33	3895.66	4.31	43.2	96.9
		MinTrans	5	26.33	3895.66	4.31	43.2	96.7
		MinStates	5	26.33	3895.66	4.31	43.2	96.9
		EarlyReject	5	30.17	4209	4.31	43.2	57.8
		LateReject	5	59.83	6168.67	4.31	38.1	126

Table 4.2: The DTP problem using SpecOverPlant heuristic as primary rule.

heuristic 2	heuristic 3	heuristic 4	avg. iter.	avg. states in $max(\mathcal{G}', \mathcal{H}')$	avg. states in $\mathcal{U}$	min time (s)	avg time (s)	max time (s)
ShortC	SpecsStart	FirstMatch	3	25.33	3222	51.4	51.8	52.4
		MinTrans	3	25.33	3222	51.4	51.9	52.7
		MinStates	3	25.33	3222	51.4	51.9	52.5
		EarlyReject	3	25.33	3222	51.4	51.8	52.7
		LateReject	3	25.33	3222	51.4	51.8	52.7
	SpecsEnd	FirstMatch	4	30	4324.5	51.2	75.7	151
		MinTrans	4	30	4324.5	51.2	75.7	151
		MinStates	4	30	4324.5	51.2	75.8	150
		EarlyReject	4	30	4324.5	51.2	83.0	151
		LateReject	4	34.33	4380.5	51.2	104	162
LongC	SpecsStart	FirstMatch	3	30.67	3264	51.9	52.4	53.4
		MinTrans	3	30.67	3264	51.8	52.4	53.1
		MinStates	3	30.67	3264	51.8	52.3	53.0
		EarlyReject	3	25.33	3182	51.8	52.1	52.9
		LateReject	3	30.67	3264	51.9	52.3	53.3
	SpecsEnd	FirstMatch	4	30	3410.97	5.03	65.5	151
		MinTrans	4	30	3410.97	5.01	65.5	152
		MinStates	4	30	3410.97	4.98	65.4	151
		EarlyReject	4	32.33	3547	51.2	70	150
		LateReject	4	55.67	6354.67	51.9	105	168.9

Table 4.3: The DTP problem using Alternating heuristic as primary rule.

heuristic 2	heuristic 3	heuristic 4	avg. iter.	avg. states in $max(\mathcal{G}', \mathcal{H}')$	avg. states in $\mathcal{U}$	min time (s)	avg time (s)	max time (s)
ShortC	SpecsStart	FirstMatch	4	25	4324.5	51.2	75.8	150
		MinTrans	4	25	4324.5	51.2	75.8	151
		MinStates	4	25	4324.5	51.2	75.8	151
		EarlyReject	4	25	4324.5	51.2	82.3	151
		LateReject	4	25	4324.5	51.2	68.5	150
	SpecsEnd	FirstMatch	5	26.33	5427	52.6	74.7	96.6
		MinTrans	5	26.33	5427	52.6	74.6	96.9
		MinStates	5	26.33	5427	52.6	74.8	97.2
		EarlyReject	5	26.33	5427	52.9	89.3	97.4
		LateReject	4.83	32.83	5511	52.6	115	166
LongC	SpecsStart	FirstMatch	3.38	30.67	3779.75	50.8	68.2	151
		MinTrans	3.38	30.67	3779.75	50.8	68.3	151
		MinStates	3.38	30.67	3779.75	50.8	68.2	151
		EarlyReject	3.5	24.5	3180.33	50.8	52.2	55.1
		LateReject	3.83	32.33	6062.5	52.2	87.6	125
	SpecsEnd	FirstMatch	4.56	28.17	4278.81	4.98	78.6	151
		MinTrans	4.56	28.17	4278.81	4.98	78.6	152
		MinStates	4.56	28.17	4278.81	5.01	78.6	152
		EarlyReject	4.5	30.17	3466	50.8	54	58.1
		LateReject	4.83	58.33	9592	124	132	168

Chapter 5

Hierarchical Interface-Based Decentralized Supervisory Control

The Hierarchical Interface-Based Supervisory Control (HISC) framework proposed in [Led02, LBLW05, LLW05, LLD06, Led09] can alleviate the state-space explosion problem, which is one of the main challenges in the control of Discrete Event Systems. HISC provides a set of local properties that can be used to verify global properties, such as nonblocking and controllability, so that the complete system model never needs to be constructed. The sufficient conditions of HISC allow the independent design and verification of different levels, ensuring that a change to one level of the hierarchy will not impact the others.

The current HISC framework does not support decentralized control problems, which arise naturally through the investigation of a large variety of distributed systems. These systems have many controllers that jointly control the distributed architecture. Further, these controllers at different sites in the distributed system may

see the effect of different sets of sensors and may control different sets of controllable events. The controllers must coordinate the disabling and enabling of events to realize the legal or desired behaviour. Also, the synthesis of decentralized supervisors requires that the specification satisfies co-observability [RW92b]. Nevertheless, when the system is composed of many sub-systems, checking co-observability using the existing monolithic method [RW95] requires the construction of the whole system model, which may be not possible due to the combinatorial explosion of the product state space.

To address the above problems, we propose an approach called the Hierarchical Interface-Based Decentralized Supervisory Control (HIDSC) framework that allows HISC to manage decentralized control problems. The proposed HIDSC framework is a scalable method that can mitigate the product state-space explosion problem, and make decentralized control scale better. We introduce a level-wise co-observability definition which does not require the synchronization of all the components. We then prove that if a system is level-wise co-observable, it is globally co-observable. This should allow larger problems to be solved. Further, we provide and prove the necessary and sufficient conditions for supervisor existence in HIDSC, which relies on a closed-loop system to be nonblocking, instead of the existing results that require $L_m(\mathbf{G})$-closure.

Most of the material in this chapter first appeared in a joint paper with R. Leduc and L. Ricker [LLR15].

5.1 Definitions Used for HIDSC

For *decentralized control,* there is an index set of $N > 1$ decentralized controllers, $D = \{1, ..., N\}$. These controllers have only a partial view of the system behaviour and control only a subset of the controllable events. To describe events that each *decentralized controller* $i \in D$ controls, we use the notation $\Sigma_{c,i} \subseteq \Sigma_c$, where $\cup_{i=1}^{N} \Sigma_{c,i} = \Sigma_c$. We refer to the set of controllers that control $\sigma \in \Sigma_c$ as $D_c(\sigma) := \{i \in D \,|\sigma \in \Sigma_{c,i}\}$.

To describe events that each decentralized controller $i \in D$ observes, we use the notation $\Sigma_{o,i} \subseteq \Sigma_o$, where $\cup_{i=1}^{N} \Sigma_{o,i} = \Sigma_o$. We refer to the set of controllers that observe $\sigma \in \Sigma_o$ by $D_o(\sigma) := \{i \in D \,|\sigma \in \Sigma_{o,i}\}$. Correspondingly, the natural projection describing the partial view of each controller is denoted by $P_i : \Sigma^* \to \Sigma_{o,i}^*$, for $i \in D$.

For decentralized control with a *conjunctive architecture* [RW92b], the fusion rule is the conjunction of all *local control decisions*, i.e., an event is globally enabled if not locally disabled. We use the conjunctive architecture in our work.

We now present a set of definitions for decentralized supervisory control in HIDSC. For decentralized control, we define a group of local partial-observation decentralized supervisors to exercise control over the plant. The following definition describes the decision rules of individual decentralized supervisors and the conjunction rule that combines their local control decisions into global control decisions.

Definition 5.1.1. *Let $K \subseteq \Sigma^*$ be the desired language, $i \in D$, and $t \in L(\mathbf{G})$. Then the decision rule for a* ***local partial-observation decentralized supervisor*** *is a function: $\mathcal{S}_{P_i}(t) := (\Sigma \backslash \Sigma_{c,i}) \cup \{\sigma \in \Sigma_{c,i} \mid P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{G}) \neq \emptyset\}$. The* ***conjunction*** *of $\mathcal{S}_{P_i}$, $i \in D$, denoted by $\mathcal{S}_{Con}$, is defined as: $\mathcal{S}_{Con}(t) := \cap_{i=1}^{N} \mathcal{S}_{P_i}(t) = \cap_{i=1}^{N} \mathcal{S}_{P_i}(P_i(t))$.*

We note that $\mathcal{S}_{P_i}(t) = \mathcal{S}_{P_i}(P_i(t))$ as the natural projection is idempotent, i.e., $P_i(t) = P_i(P_i(t))$.

We now define the closed behaviour for the closed-loop system of $\mathbf{G}$ under the control of $\mathcal{S}_{Con}$.

Definition 5.1.2. *Given* $\mathbf{G}$ *and* $\mathcal{S}_{Con}$*, the resulting* ***closed-loop system*** *is denoted by* $\mathcal{S}_{Con}/\mathbf{G}$*. The system's* ***closed behaviour*** $L(\mathcal{S}_{Con}/\mathbf{G})$*, is recursively defined as follows:*

I) $\epsilon \in L(\mathcal{S}_{Con}/\mathbf{G})$

II) $t \in L(\mathcal{S}_{Con}/\mathbf{G})$*,* $\sigma \in \mathcal{S}_{Con}(t)$*, and* $t\sigma \in L(\mathbf{G})$ *if and only if* $t\sigma \in L(\mathcal{S}_{Con}/\mathbf{G})$*.*

The following is the definition of decentralized supervisory control.

Definition 5.1.3. *Given* $\mathbf{G}$ *and* $\mathcal{S}_{Con}$*, we say* $\mathcal{S}_{Con}$ *is a* ***decentralized supervisory control*** *if the decision rule is defined as in Definition 5.1.1, and the resulting closed-loop system and closed behaviour is defined as in Definition 5.1.2.*

The following is the definition of nonblocking decentralized supervisory control. It states that the marked language of the closed-loop system is the set of strings marked by $\mathbf{G}$ and one still possible in the system's closed-loop behaviour.

Definition 5.1.4. *We say that* $\mathcal{S}_{Con}$ *is a* ***nonblocking decentralized supervisory control (NDSC)*** *for* $\mathbf{G}$ *if* $\overline{L_m(\mathcal{S}_{Con}/\mathbf{G})} = L(\mathcal{S}_{Con}/\mathbf{G})$ *where* $L_m(\mathcal{S}_{Con}/\mathbf{G}) := L(\mathcal{S}_{Con}/\mathbf{G}) \cap L_m(\mathbf{G})$*.*

It is useful to introduce a generalization of NDSC in which the supervisory action also includes marking as well as control, since allowing supervisors to add marking

information makes them more expressive. Now, the marked language of the closed-loop system is the set of strings marked by $K \subseteq L_m(\mathbf{G})$ that are still possible in the system's closed-loop behaviour.

Definition 5.1.5. *Let $K \subseteq L_m(\mathbf{G})$. We say that $\mathcal{S}_{con}$ is a **marking nonblocking decentralized supervisory control (MNDSC)** for $(K, \mathbf{G})$ if $\overline{L_m(\mathcal{S}_{Con}/\mathbf{G})} = L(\mathcal{S}_{Con}/\mathbf{G})$ where $L_m(\mathcal{S}_{con}/\mathbf{G}) := L(\mathcal{S}_{con}/\mathbf{G}) \cap K$.*

The next definition creates an equivalence between theoretical decentralized supervisory controls and DES supervisors. The idea is that the marking and control information of $\mathcal{S}_{Con}$ is represented by specification $\mathbf{H}$, and that the closed-loop behaviour of $\mathbf{H}||\mathbf{G}$ is equivalent to $\mathcal{S}_{con}/\mathbf{G}$.

Definition 5.1.6. *Let $\mathcal{S}_{Con}$ be a MNDSC for plant $\mathbf{G} = (Q,\ \Sigma,\ \delta,\ q_0,\ Q_m)$ and $K \subseteq L_m(\mathbf{G})$, with $L_m(\mathcal{S}_{Con}/\mathbf{G}) = L(\mathcal{S}_{con}/\mathbf{G}) \cap K$ and $L(\mathcal{S}_{Con}/\mathbf{G}) = \overline{K}$. Let $\mathbf{H} = (X,\ \Sigma,\ \xi,\ x_0,\ X_m)$ be a specification automaton. We say that $\mathbf{H}||\mathbf{G}$ has **equivalent MNDSC behaviour** with $\mathcal{S}_{Con}/\mathbf{G}$, if $K = L_m(\mathbf{H}) \cap L_m(\mathbf{G})$ and $\overline{K} = L(\mathbf{H}) \cap L(\mathbf{G})$. Alternatively, we say that $\mathbf{H}$ is an **equivalent theoretical implementation of MNDSC** $\mathcal{S}_{Con}$ for $\mathbf{G}$.*

In our work, we focus on MNDSC, which is a more expressive supervisory control paradigm. In particular, it will allow us to later introduce a decentralized supervisor existence result that relies on the closed-loop system $\mathbf{H}||\mathbf{G}$ to be nonblocking, instead of the existing results that require K to be $L_m(\mathbf{G})$-closed. This is essential to adapting decentralized control to the HISC approach as HISC provides a scalable method to verify nonblocking, but not $L_m(\mathbf{G})$-closure.

We note that in decentralized control, there is no real implementation of the centralized supervisor $\mathbf{H}$. The above MNDSC $\mathcal{S}_{Con}$, defined as the control policy of

the conjunction of a group of decentralized supervisors, is the real supervisor. We also note that for an HISC system, **H** will correspond to the theoretical flat supervisor of the system defined in Section 5.2, and will be used to determine if the flat system is nonblocking.

For reference, below is the traditional nonblocking decentralized supervisory control existence theorem that requires that K be $L_m(\mathbf{G})$-closed. We will later introduce a decentralized supervisory control existence result in Section 5.4 which does not require that K be $L_m(\mathbf{G})$-closed, but instead will be based upon marking nonblocking decentralized supervisory control.

Theorem 5.1.1 ([CL08]). *Consider DES* $\mathbf{G} = (Q,\ \Sigma,\ \delta,\ q_0,\ Q_m)$, *where* $\Sigma_{uc} \subseteq \Sigma$ *is the set of uncontrollable events,* $\Sigma_c = \Sigma \backslash \Sigma_{uc}$ *is the set of controllable events, and* $\Sigma_o \subseteq \Sigma$ *is the set of observable events. For each site* i, *where* $i = 1, ..., N$ *consider the set of controllable events* $\Sigma_{c,i}$ *and the set of observable events* $\Sigma_{o,i}$; *overall,* $\bigcup_{i=1}^{N} \Sigma_{c,i} = \Sigma_c$ *and* $\bigcup_{i=1}^{N} \Sigma_{o,i} = \Sigma_o$. *Let* P_i *be the natural projection from* Σ^* *to* $\Sigma_{o,i}^*$, *where* $i = 1, ..., N$. *Consider also the language* $K \subseteq L_m(G)$, *where* $K \neq \emptyset$. *There exists a nonblocking decentralized supervisor* S_{con} *for* $\mathbf{G}$ *such that* $L_m(S_{con}/G) = K$ *and* $L(S_{con}/G) = \overline{K}$ *if and only if the following three conditions hold:*

1. *K is controllable with respect to $L(G)$ and Σ_{uc};*
2. *K is co-observable with respect to $L(G)$, $\Sigma_{o,i}$, and $\Sigma_{c,i}$, $i = 1, ..., N$;*
3. *K is $L_m(\mathbf{G})$-closed.*

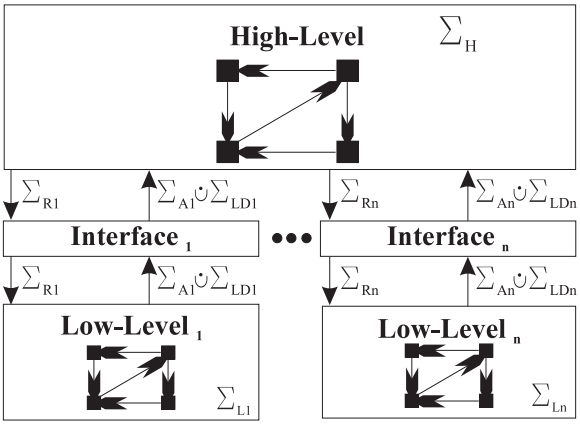


Figure 5.1: Interface Block Diagram with Low Data Events.

5.2 HISC Architecture

The HISC [Led02, LBLW05, LLW05, LLD06, Led09] approach decomposes a system into a high-level subsystem which communicates with $n \geq 1$ parallel low-level subsystems through separate *interfaces* that restrict the interaction of the subsystems. The high-level subsystem communicates with each low-level subsystem through a separate interface.

In HISC there is a master-slave relationship. A high-level subsystem sends a command to a particular low-level subsystem, which then performs the indicated task and returns a response (answer). Figure 5.1 shows conceptually the structure and information flow of the system. The overall structure of the system is shown in Figure 5.2. This style of interaction is enforced by an interface that mediates communication between the two subsystems. All system components, including the interfaces, are modeled as automata.

To restrict information flow and decouple the subsystems, the system alphabet is partitioned into pairwise disjoint alphabets:

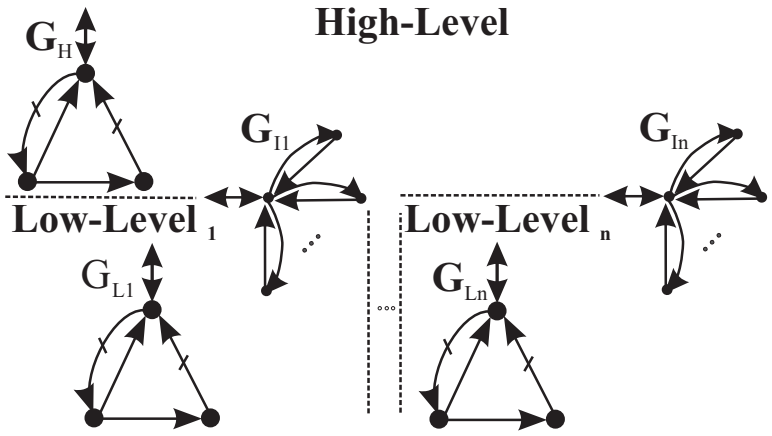


Figure 5.2: Two Tiered Structure of Parallel System

$$\Sigma := \Sigma_H \dot{\cup} \dot{\bigcup_{j=1,\ldots,n}} [\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}] \tag{5.1}$$

where we use $\dot{\cup}$ to represent disjoint union.

The events in Σ_H are called *high-level events* and the events in Σ_{L_j} are the j^{th} *low-level events* ($j = 1, \ldots, n$) as these events appear only in the high level and j^{th} low-level subsystem models, $\mathbf{G}_H$ and $\mathbf{G}_{L_j}$ respectively. We then have $\mathbf{G}_H$ defined over event set $\Sigma_H \dot{\cup} (\dot{\cup}_{j \in \{1,\ldots,n\}} [\Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}])$ and $\mathbf{G}_{L_j}$ defined over event set $\Sigma_{L_j} \dot{\cup} \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$. We model the j^{th} interface by DES $\mathbf{G}_{I_j}$, which is defined over event set $\Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}$. For the remainder of this thesis, we assume $j \in \{1, \ldots, n\}$.

The events in Σ_{R_j}, called *request events*, represent commands sent from the high-level subsystem to the j^{th} low-level subsystem. The events in Σ_{A_j} are *answer events* and represent the low-level subsystem's responses to the request events. The events in Σ_{LD_j} are called *low data events* which provide a means for a low level to send information (data) through the interface. Request, answer, and low

data events are collectively known as the set of LD *interface events*, defined as $\Sigma_I := \dot{\cup}_{k\in\{1,\ldots,n\}}[\Sigma_{R_k}\dot{\cup}\Sigma_{A_k}\dot{\cup}\Sigma_{LD_k}]$, and $\mathbf{G}_{I_j}$is an LD interface defined below. Please see [Led09] for a discussion of the various points.

Figure 5.3 shows an example of an LD interface. It could correspond to a machine at the low level with an effective internal buffer of two. In this diagram, the initial state can be recognized by a thick outline, and marked states are filled.

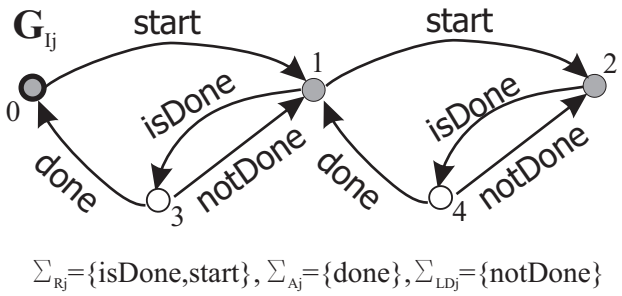


Figure 5.3: Example LD Interface

Definition 5.2.1. *The j^{th} interface DES $\mathbf{G}_{\mathbf{I_j}} = (X_j, \Sigma_{I_j}, \xi_j, x_{o_j}, X_{m_j})$ is a* LD interface *if the following properties are satisfied:*

1. $x_{o_j} \in X_{m_j}$

2. $(\forall x \in X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x,\sigma)! \Rightarrow [\sigma \in \Sigma_{R_j}] \ \vee \ [\sigma \in \Sigma_{LD_j} \wedge \xi_j(x,\sigma) \in X_{m_j}]$

3. $(\forall x \in X_j - X_{m_j})(\forall \sigma \in \Sigma_{I_j})\, \xi_j(x,\sigma)! \Rightarrow$
   $[\sigma \in \Sigma_{A_j} \wedge \xi_j(x,\sigma) \in X_{m_j}] \vee [\sigma \in \Sigma_{LD_j}]$

To simplify notation in our exposition, we bring in the following event sets, natural projections, and languages.

$$
\begin{aligned}
\Sigma_{I_j} &:= \Sigma_{R_j} \dot{\cup} \Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}, & P_{I_j} &: \Sigma^* \to \Sigma_{I_j}^* \\
\Sigma_{IL_j} &:= \Sigma_{L_j} \cup \Sigma_{I_j}, & P_{IL_j} &: \Sigma^* \to \Sigma_{IL_j}^* \\
\Sigma_{IH} &:= \Sigma_H \cup \bigcup_{k \in \{1,\ldots,n\}} \Sigma_{I_k}, & P_{IH} &: \Sigma^* \to \Sigma_{IH}^* \\
\mathcal{H} &:= P_{IH}^{-1}(L(\mathbf{G}_H)), & \mathcal{H}_m &:= P_{IH}^{-1}(L_m(\mathbf{G}_H)) \subseteq \Sigma^* \\
\mathcal{L}_j &:= P_{IL_j}^{-1}(L(\mathbf{G}_{L_j})), & \mathcal{L}_{m_j} &:= P_{IL_j}^{-1}(L_m(\mathbf{G}_{L_j})) \subseteq \Sigma^* \\
\mathcal{I}_j &:= P_{I_j}^{-1}(L(\mathbf{G}_{I_j})), & \mathcal{I}_{m_j} &:= P_{I_j}^{-1}(L_m(\mathbf{G}_{I_j})) \subseteq \Sigma^* \\
\mathcal{I} &:= \cap_{k \in \{1,\ldots,n\}} \mathcal{I}_k, & \mathcal{I}_m &:= \cap_{k \in \{1,\ldots,n\}} \mathcal{I}_{m_k} \\
\Sigma_{LD} &:= \bigcup_{k \in \{1,\ldots,n\}} \Sigma_{LD_k}
\end{aligned}
$$

We define our *flat system* to be $\mathbf{G} = \mathbf{G}_H || \mathbf{G}_{I_1} || \mathbf{G}_{L_1} || \ldots || \mathbf{G}_{I_n} || \mathbf{G}_{L_n}$. By flat system we mean the equivalent DES if we ignored the interface structure.

We now present the properties that an HISC system must satisfy to ensure that it interacts with the interfaces correctly. Please see [Led09] for a discussion of the various points.

Definition 5.2.2. *The n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, is* LD interface consistent *with respect to the alphabet partition given by (5.1), if for all $j \in \{1, \ldots, n\}$, the following conditions are satisfied:*

Multi-level Properties

1. *The event set of $\mathbf{G}_H$ is Σ_{IH}, and the event set of $\mathbf{G}_{L_j}$ is Σ_{IL_j}.*

2. $\mathbf{G}_{I_j}$ *is a LD interface.*

High-Level Property

3. $(\forall s \in \mathcal{H} \cap \mathcal{I})\ Elig_{\mathcal{I}_j}(s)\ \cap\ (\Sigma_{A_j} \dot{\cup} \Sigma_{LD_j}) \subseteq Elig_{\mathcal{H}}(s)$

Low-Level Properties

4. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)\ Elig_{\mathcal{I}_j}(s)\ \cap\ \Sigma_{R_j} \subseteq Elig_{\mathcal{L}_j}(s)$

5. $(\forall s \in \Sigma^*.\Sigma_{R_j} \cap \mathcal{L}_j \cap \mathcal{I}_j)$

$Elig_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) \cap \Sigma_{A_j} = Elig_{\mathcal{I}_j}(s) \cap \Sigma_{A_j}$ *where*

$$Elig_{\mathcal{L}_j \cap \mathcal{I}_j}(s\Sigma^*_{L_j}) := \bigcup_{l \in \Sigma^*_{L_j}} Elig_{\mathcal{L}_j \cap \mathcal{I}_j}(sl)$$

6. $(\forall s \in \mathcal{L}_j \cap \mathcal{I}_j)$

$s \in \mathcal{I}_{m_j} \Rightarrow (\exists l \in \Sigma^*_{L_j})\ sl \in \mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}$.

We now provide an additional set of properties that the system must satisfy if the flat system $\mathbf{G}$ is to be nonblocking. Please see [Led09] for a discussion of the various points.

Definition 5.2.3. *The* n^{th} *degree* ($n \geq 1$) *interface system composed of DES* $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$, *is said to be* LD level-wise nonblocking *if the following conditions are satisfied:*

(I) LD nonblocking at the high level*:*

$(\forall s \in \mathcal{H} \cap \mathcal{I})(\exists s' \in (\Sigma - \Sigma_{LD})^*)$

$ss' \in \mathcal{H}_m \cap \mathcal{I}_m$

(II) nonblocking at the low level*:*

$$(\forall j \in \{1, \ldots, n\})\, \overline{\mathcal{L}_{m_j} \cap \mathcal{I}_{m_j}} = \mathcal{L}_j \cap \mathcal{I}_j$$

The theorem bellow states that verifying the LD level-wise nonblocking and LD interface consistent conditions is sufficient to verify that our flat system is nonblocking.

Theorem 5.2.1 ([Led09])**.** *If the* n^{th} *degree* ($n \geq 1$) *interface system composed of DES* $\mathbf{G}_H, \mathbf{G}_{I_1}, \mathbf{G}_{L_1}, \ldots, \mathbf{G}_{I_n}, \mathbf{G}_{L_n}$*, is LD level-wise nonblocking and LD interface consistent with respect to the alphabet partition given by (5.1), then*

$$L(G) = \overline{L_m(G)} \text{ where } G = G_H||G_{L_1}||G_{I_1}|| \ldots ||G_{L_n}||G_{I_n}$$

Since checking that the LD level-wise nonblocking and LD interface consistent conditions only require a single component at a time, we note that we can evaluate each level independently. This means we do not need to construct the entire system model, which provides a potentially significant reduction in computational resources.

For controllability, we need to separate the subsystems into their plant and supervisor sub-components. We will do this as in Figure 5.4. We define the *high-level plant* to be $\mathbf{G}_H^p$, and the *high-level supervisor* to be $\mathbf{S}_H$ (defined over event set Σ_{IH}). We define the j^{th} *low-level plant* and *supervisor* to be $\mathbf{G}_{L_j}^p$ and $\mathbf{S}_{L_j}$ (defined over Σ_{IL_j}) respectively. We next define the high-level subsystem to be $\mathbf{G}_H := \mathbf{G}_H^p||\mathbf{S}_H$, and define the j^{th} low-level subsystem to be $\mathbf{G}_{L_j} := \mathbf{G}_{L_j}^p||\mathbf{S}_{L_j}$. We note that in HISC systems, interfaces are always supervisors.

We can now define our *flat supervisor* and *plant* as well as some other languages as follows:

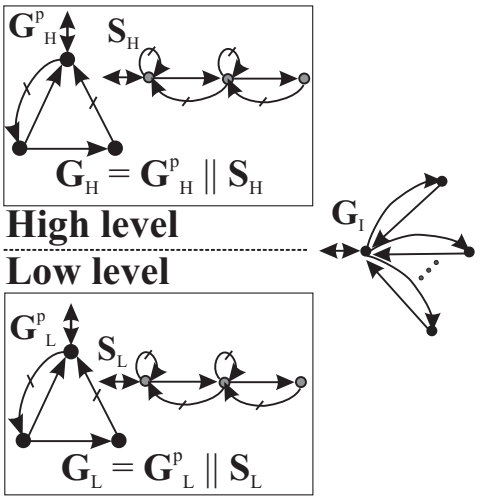


Figure 5.4: Plant and Supervisor Subplant Decomposition

$\mathbf{Plant} := \mathbf{G}^p_H||\mathbf{G}^p_{L_1}||\ldots||\mathbf{G}^p_{L_n}$,

$\mathbf{Sup} := \mathbf{S}_H||\mathbf{S}_{L_1}||\ldots||\mathbf{S}_{L_n}||\mathbf{G}_{I_1}||\ldots||\mathbf{G}_{I_n}$,

$\mathcal{H}^p := P^{-1}_{IH}L(\mathbf{G}^p_H)$, $\mathcal{S}_H := P^{-1}_{IH}L(\mathbf{S}_H) \subseteq \Sigma^*$,

$\mathcal{L}^p_j := P^{-1}_{IL_j}L(\mathbf{G}^p_{L_j})$, $\mathcal{S}_{L_j} := P^{-1}_{IL_j}L(\mathbf{S}_{L_j}) \subseteq \Sigma^*$.

The controllability requirements that each level must satisfy are given in the following definition.

Definition 5.2.4. *The n^{th} degree ($n \geq 1$) interface system composed of DES $\mathbf{G}^p_H, \mathbf{S}_H, \mathbf{G}^p_{L_1}, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}^p_{L_n}, \mathbf{S}_{L_n}, \mathbf{G}_{I_n}$, is* LD level-wise controllable *with respect to the alphabet partition given by (5.1), if for all $j \in \{1, \ldots, n\}$ the following conditions hold:*

(I) The alphabet of $\mathbf{G}^p_H$ and $\mathbf{S}_H$ is Σ_{IH}, the alphabet of $\mathbf{G}^p_{L_j}$ and $\mathbf{S}_{L_j}$ is Σ_{IL_j}, and the alphabet of $\mathbf{G}_{I_j}$ is Σ_{I_j}

(II) $(\forall s \in \mathcal{L}^p_j \cap \mathcal{S}_{L_j} \cap \mathcal{I}_j)$ $Elig_{\mathcal{L}^p_j}(s) \cap \Sigma_u \subseteq Elig_{\mathcal{S}_{L_j} \cap \mathcal{I}_j}(s)$

(III) $(\forall s \in \mathcal{H}^p \cap \mathcal{I} \cap \mathcal{S}_H)$ $Elig_{\mathcal{H}^p \cap \mathcal{I}}(s) \cap \Sigma_u \subseteq Elig_{\mathcal{S}_H}(s)$

The theorem below states that verifying LD level-wise controllable is sufficient to verify that the flat supervisor is controllable for the flat plant.

Theorem 5.2.2 ([Led09]). *If* n^{th} *degree* ($n \geq 1$) *interface system composed of DES* $\mathbf{G}^p_H, \mathbf{S}_H, \mathbf{G}^p_{L_1}, \mathbf{S}_{L_1}, \mathbf{G}_{I_1}, \ldots, \mathbf{G}^p_{L_n}, \mathbf{S}_{L_n}, \mathbf{G}_{I_n}$ *is LD level-wise controllable with respect to the alphabet partition given by (5.1), then*

$(\forall s \in L(\mathbf{Plant}) \cap L(\mathbf{Sup}))$

$Elig_{L(\mathbf{Plant})}(s) \cap \Sigma_u \subseteq Elig_{L(\mathbf{Sup})}(s)$

Since checking that the LD level-wise controllable condition only requires at most a single component at a time, we note that we can evaluate each level independently. As such we again do not need to construct the entire system model.

5.3 HIDSC Architecture

In Section 5.2, we described a system composed of plant DES $\mathbf{G}^p_H$, $\mathbf{G}^p_{L_1}, \ldots,$ $\mathbf{G}^p_{L_n}$, supervisor DES $\mathbf{S}_H, \mathbf{S}_{L_1}, \ldots, \mathbf{S}_{L_n}$, and interface DES $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$. Although the level-wise controllability condition [Led02, Led09] does effectively limit the high-level supervisor to events in Σ_{IH}, and the j^{th} low-level supervisor to events in Σ_{IL_j}, it requires the HISC structure and does not allow further restrictions outside of this structure. In order to allow decentralized supervisors within components, we need to extend the HISC structure. We will now introduce the *Hierarchical Interface-based Decentralized Supervisory Control (HIDSC) architecture.* HIDSC is an extension of HISC from centralized control to a decentralized architecture by allowing decentralized supervisors within a subsystem, but without additional HISC restrictions.

In the HIDSC framework, all the HISC supervisors are replaced by corresponding "specification" DES. In decentralized control, these specification DES represent the control behaviours we wish to implement as decentralized controllers, not specifications for synthesizing a centralized maximally permissive supervisor.

For HIDSC, we will replace supervisor $\mathbf{S}_H$ by specification DES $\mathbf{F}_H$ (defined over Σ_{IH}), and we will replace supervisor $\mathbf{S}_{L_j}$ by specification DES $\mathbf{F}_{L_j}$ (defined over Σ_{IL_j}). Typically, $\mathbf{F}_H$ will express system-wide constraints about how the components interact and what tasks the low levels should perform. $\mathbf{F}_{L_j}$ expresses how the j^{th} low level will perform the tasks (requests) given to it by the high level. For each component, there is a different index set of decentralized controllers.

We are now ready to define the structure of an HIDSC system.

Definition 5.3.1. *The* $n^{\boldsymbol{th}}$ ***degree decentralized specification interface system*** *with respect to the alphabet partition given by (5.1) is composed of plant DES* $\mathbf{G}_H^p$, $\mathbf{G}_{L_1}^p, \ldots, \mathbf{G}_{L_n}^p$, *specification DES* $\mathbf{F}_H$, $\mathbf{F}_{L_1}, \ldots, \mathbf{F}_{L_n}$, *interface DES* $\mathbf{G}_{I_1}, \ldots, \mathbf{G}_{I_n}$, *and high-level and low-level decentralized controllers. The system has the following structure.*

High level:

- *The high-level decentralized controllers have an index set* $D_H := \{N_{H,1}, \ldots, N_{H,n_0}\}$.
- *The event set for* $\mathbf{G}_H^p$, $\mathbf{F}_H$ *and the corresponding decentralized controllers is* Σ_{IH}.
- *For* $i \in D_H$, $\Sigma_{H,c,i} \subseteq \Sigma_c \cap \Sigma_{IH}$ *and* $\Sigma_{H,o,i} \subseteq \Sigma_o \cap \Sigma_{IH}$ *are the corresponding controllable and observable event subsets for the high-level decentralized controllers.*

Low level:

- *For $j \in \{1, \ldots, n\}$, the j^{th} low-level component has an index set $D_{L_j} := \{N_{L_j,1}, \ldots, N_{L_j,n_j}\}$ for its own decentralized controllers.*

- *The event set of each low-level component $\mathbf{G}^p_{L_j}$, $\mathbf{F}_{L_j}$ and the corresponding decentralized controllers is Σ_{IL_j}.*

- *For $i \in D_{L_j}$, $\Sigma_{L_j,c,i} \subseteq \Sigma_c \cap \Sigma_{IL_j}$ and $\Sigma_{L_j,o,i} \subseteq \Sigma_o \cap \Sigma_{IL_j}$ are the corresponding controllable and observable event subsets for the low-level decentralized controllers.*

Multi-level:

- *The index set for all decentralized controllers in the system is $D := D_H \dot{\cup} \dot{\bigcup}_{j=1}^{n} D_{L_j} = \{1, ..., N\}$.*

For the rest of this section, we will refer to such a system as an n^{th} degree decentralized specification interface system Ψ, or simply Ψ. Note that in Ψ, we do not specify the index of decentralized controllers by $\{1, ..., n_0\}$, $\{1, ..., n_j\}$, etc., because once combined they would overlap. We create the system index set using disjoint union.

The *flat system* $\mathbf{G}$ is the synchronization of all the plant, specification, and interface components in the system, i.e., $\mathbf{G} = \mathbf{G}^p_H \parallel \mathbf{G}^p_{L_1} \parallel \ldots \parallel \mathbf{G}^p_{L_n} \parallel \mathbf{F}_H \parallel \mathbf{F}_{L_1} \parallel \ldots \parallel \mathbf{F}_{L_n} \parallel \mathbf{G}_{I_1} \parallel \ldots \parallel \mathbf{G}_{I_n}$. We use the term *flat system* to mean the overall system ignoring the HIDSC structure.

It is important to note that for an HIDSC system, we would first design level-wise supervisors for the original HISC system while ignoring any decentralized restrictions. We would then use the HISC structure to verify that the system is nonblocking

and controllable. We would next use these level-wise supervisors (which include the system's interface DES) as "specifications" for the design of the per-component decentralized supervisors specified by the HIDSC system. The final system would not contain any of these specification DES, just the resulting decentralized controllers that would provide us with equivalent closed-loop behaviour (see Corollary 5.5.1 in Section 5.5).

5.4 HIDSC Co-observability Definition and Theorem

The main focus here is to verify co-observability in an HIDSC system Ψ without explicitly constructing the flat system. We will only perform a per-component co-observability verification, but guarantee that the whole system is co-observable.

To aid in defining our per-component co-observability definition and HIDSC co-observability theorem, we specify some decentralized notations for Ψ.

We use $D_{H,c}(\sigma) := \{i \in D_H \,|\sigma \in \Sigma_{H,c,i}\}$ to denote the set of decentralized controllers in the high level that can control the event σ. We use $D_{H,o}(\sigma) := \{i \in D_H \,|\sigma \in \Sigma_{H,o,i}\}$ to denote the set of decentralized controllers in the high level that can observe the event σ. Correspondingly, $P_{H,i} : \Sigma^* \to (\Sigma_{H,o,i})^*$ is the natural projection describing the partial view of controller $i \in D_H$.

For $j \in \{1, \ldots, n\}$, $D_{L_j,c}(\sigma) := \left\{i \in D_{L_j} \,|\sigma \in \Sigma_{L_j,c,i}\right\}$ is the set of decentralized controllers in the j^{th} low-level component that can control the event σ. We use $D_{L_j,o}(\sigma) := \left\{i \in D_{L_j} \,|\sigma \in \Sigma_{L_j,o,i}\right\}$ to represent the set of decentralized controllers in the j^{th} low-level component that can observe the event σ. Correspondingly, $P_{L_j,i}$:

$\Sigma^* \rightarrow (\Sigma_{L_j,o,i})^*$ is the natural projection describing the partial view of controller $i \in D_{L_j}$.

We use $D_c(\sigma) := \{i \in D \,|\sigma \in \Sigma_c\}$ to denote the set of decentralized controllers that can control the event σ.

Further, we introduce a few languages used for the HIDSC co-observability definition and theorem.

$\mathcal{F}_H := P_{IH}^{-1}(L(\mathbf{F}_H))$, $\mathcal{F}_{L_j} := P_{IL_j}^{-1}(L(\mathbf{F}_{L_j}))$,

$\mathcal{F} := \mathcal{F}_H \cap \mathcal{F}_{L_1} \cap \ldots \cap \mathcal{F}_{L_n}$, $\mathcal{P} := \mathcal{H}^p \cap \mathcal{L}_1^p \cap \ldots \cap \mathcal{L}_n^p$, $L(G) := \mathcal{P}$.

Language $\mathcal{F}_H$ represents the behaviour of the specification automata in the high-level subsystem, while $\mathcal{F}_{L_j}$ represents the behaviour of the specification automata for component j in the low-level subsystem. Language $\mathcal{F}$ represents the global specification of the flat system, and $\mathcal{P}$ represents the behaviour of the flat plant.

5.4.1 HIDSC Co-observability Definition

We now present the per-component level-wise co-observability definition for HIDSC system Ψ. We note that each individual condition needs at most a single subsystem for its verification.

Definition 5.4.1. *Let Ψ be an HIDSC n^{th} degree decentralized specification interface system. Then Ψ is **level-wise co-observable** if for all $j \in \{1, \ldots, n\}$ the following conditions hold:*

I) $(\forall t \in \mathcal{F}_H \cap \mathcal{H}^p \cap \mathcal{I})(\forall \sigma \in \Sigma_c)$

$$t\sigma \in (\mathcal{H}^p \cap \mathcal{I}) \backslash \mathcal{F}_H \Rightarrow (\exists i \in D_{H,c}(\sigma))\ P_{H,i}^{-1}[P_{H,i}(t)]\sigma \cap \mathcal{F}_H \cap \mathcal{H}^p \cap \mathcal{I} = \emptyset,$$

II) $(\forall t \in \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p)(\forall \sigma \in \Sigma_c)$

$$t\sigma \in \mathcal{L}_j^p \backslash (\mathcal{F}_{L_j} \cap \mathcal{I}_j) \Rightarrow (\exists i \in D_{L_j,c}(\sigma)) P_{L_j,i}^{-1}[P_{L_j,i}(t)]\sigma \cap \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p = \emptyset.$$

Definition 5.4.1 states that HIDSC system Ψ is *level-wise co-observable* if the high-level component is co-observable and each low-level component is co-observable.

We note that the interfaces are treated as specifications at the low level and treated as plants at the high level. This is done this way because interfaces represent the behaviour provided by its low level and the information needed to verify that it is co-observable is typically present at the low level but not the high level. To avoid having to repeat this information at the high level, we use the results of [LLMR14] that allow us to treat supervisors as if they are plants once we verify they are co-observable. By treating interfaces as plants at the high level, we allow the high-level supervisor to be more permissive in general as there will typically be fewer strings that can cause the co-observability verification to fail.

We now restate the co-observability definition in terms of our HIDSC system. We note that from their definition, we know that languages $\mathcal{F}$, $\mathcal{I}$, and $\mathcal{P}$ are prefix-closed. We also note that according to Definition 5.4.1, each $i \in D$ represents some $i_1 \in D_H$ as some $i_2 \in D_{L_j}$, $j \in \{1, ..., n\}$.

Definition 5.4.2. *Let Ψ be an HIDSC n^{th} degree decentralized specification interface system. Let $D = \{1, ..., N\} = D_H \dot{\cup} \dot{\bigcup}_{j=1}^{n} D_{L_j}$ be the index set for Ψ. Let $\Sigma_{c,i} \subseteq \Sigma$ and $\Sigma_{o,i} \subseteq \Sigma$ be sets of controllable and observable events, respectively, for $i \in D$, where $D_c(\sigma) = \{i \in D \,|\sigma \in \Sigma_{c,i}\}$. Let $P_i : \Sigma^* \to \Sigma_{o,i}^*$, $i \in D$, be natural projections. Then Ψ is **globally co-observable** if*

$(\forall t \in \mathcal{F} \cap \mathcal{I} \cap \mathcal{P})\ (\forall \sigma \in \Sigma_c)\ t\sigma \in \mathcal{P} \backslash (\mathcal{F} \cap \mathcal{I}) \Rightarrow (\exists i \in D_c(\sigma)) P_i^{-1}[P_i(t)]\sigma \cap \mathcal{F} \cap \mathcal{I} \cap \mathcal{P} = \emptyset$.

Note that Definition 5.4.2 is the property we want to verify but we will do so by using our per-component co-observability definition. We do not need to combine

the flat specifications, interfaces and plant components together in order to verify Definition 5.4.2, thus potentially saving computation and helping to alleviate the state-space explosion problem.

5.4.2 HIDSC Co-observability Theorem

The following is the HIDSC co-observability theorem which states that the level-wise co-observability property is sufficient to guarantee that the flat system is co-observable. This means that co-observability for the system can be verified while only constructing a single component at a time.

Theorem 5.4.1. *Let Ψ be an HIDSC n^{th} degree decentralized specification interface system. If Ψ is level-wise co-observable then Ψ is globally co-observable.*

Proof. Let Ψ be an HIDSC n^{th} degree decentralized specification interface system.

Assume Ψ is level-wise co-observable. We will now show that Ψ is globally co-observable.

Sufficient to show that:

$(\forall t \in \mathcal{F} \cap \mathcal{I} \cap \mathcal{P})\ (\forall \sigma \in \Sigma_c)\ t\sigma \in \mathcal{P} \backslash (\mathcal{F} \cap \mathcal{I}) \Rightarrow (\exists i \in D_c(\sigma)) P_i^{-1}[P_i(t)]\sigma \cap \mathcal{F} \cap \mathcal{I} \cap \mathcal{P} = \emptyset$.

Let $t \in \mathcal{F} \cap \mathcal{I} \cap \mathcal{P}$ and $\sigma \in \Sigma_c$.

Assume $t\sigma \in \mathcal{P} \backslash (\mathcal{F} \cap \mathcal{I})$.

As $\mathcal{F} = \mathcal{F}_H \cap \mathcal{F}_{L_1} \cap \ldots \cap \mathcal{F}_{L_n}$, $\mathcal{P} = \mathcal{H}^p \cap \mathcal{L}_1^p \cap \ldots \cap \mathcal{L}_n^p$, $\mathcal{I} = \mathcal{I}_1 \cap \ldots \cap \mathcal{I}_n$ and their corresponding component languages are all prefix-closed by definition, we conclude from $t \in \mathcal{F} \cap \mathcal{I} \cap \mathcal{P}$ that:

$t \in \mathcal{F}_H \cap \mathcal{H}^p \cap \mathcal{I}$, $(\forall j \in \{1, \ldots, n\})\ t \in \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p$, and $(\forall j \in \{1, \ldots, n\})$ $t \in \mathcal{I}_j \cap \mathcal{L}_j^p$ **(1)**

As $t\sigma \in \mathcal{P}\backslash(\mathcal{F} \cap \mathcal{I})$, we have: $t\sigma \in \mathcal{P}$ and $t\sigma \notin \mathcal{F} \cap \mathcal{I}$.

$\Rightarrow t\sigma \notin \mathcal{F}_H \cap \mathcal{F}_{L_1} \cap \ldots \cap \mathcal{F}_{L_n} \cap \mathcal{I}_1 \cap \ldots \cap \mathcal{I}_n$, by definition of $\mathcal{F}$ and $\mathcal{I}$.

$\Rightarrow t\sigma \in \mathcal{P}$ and $t\sigma \notin \mathcal{F}_H$, or $t\sigma \in \mathcal{P}$ and $(\exists j \in \{1, \ldots, n\})\ t\sigma \notin \mathcal{F}_{L_j} \cap \mathcal{I}_j$ **(2)**

Case 1) $(\exists j \in \{1, \ldots, n\})\ t\sigma \notin \mathcal{F}_{L_j} \cap \mathcal{I}_j$.

Let $j \in \{1, \ldots, n\}$.

We also have $t \in \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p$, by **(1)**.

$\Rightarrow t \in \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p$, $t\sigma \in \mathcal{L}_j^p$ and $t\sigma \notin \mathcal{F}_{L_j} \cap \mathcal{I}_j$ as $t\sigma \in \mathcal{P}$ and $\mathcal{P} = \mathcal{H}^p \cap \mathcal{L}_1^p \cap \ldots \cap \mathcal{L}_n^p$.

As Ψ is level-wise co-observable, we have:

$(\exists i \in D_{L_j,c}(\sigma))\ P_{L_j,i}^{-1}[P_{L_j,i}(t)]\sigma \cap \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p = \emptyset$.

$\Rightarrow (\exists i \in D_c(\sigma))\ P_{L_j,i}^{-1}[P_{L_j,i}(t)]\sigma \cap \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p = \emptyset$, as $D_{L_j,c}(\sigma) \subseteq D_c(\sigma)$.

Let $i \in D_c(\sigma)$.

$\Rightarrow P_i^{-1}[P_i(t)]\sigma \cap \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p = \emptyset$, as $P_{L_j,i}(t) : \Sigma^* \rightarrow (\Sigma_{L_j,o,i})^*$ and $P_i(t) : \Sigma^* \rightarrow (\Sigma_{L_j,o,i})^*$.

We note that the above $i \in D_{L_j,c}(\sigma)$ represents the same decentralized supervisor as the $i \in D_c(\sigma)$. They thus have the same set of observable events, thus $P_{L_j,i} = P_i$.

$\Rightarrow P_i^{-1}[P_i(t)]\sigma \cap \mathcal{F} \cap \mathcal{I} \cap \mathcal{P} = \emptyset$, as $\mathcal{F} \cap \mathcal{I} \cap \mathcal{P} \subseteq \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p$, thus $s \in \mathcal{F} \cap \mathcal{I} \cap \mathcal{P} \Rightarrow s \in \mathcal{F}_{L_j} \cap \mathcal{I}_j \cap \mathcal{L}_j^p$ which would cause a contradiction.

Case 2) $(\forall j \in \{1, \ldots, n\})\ t\sigma \in \mathcal{F}_{L_j} \cap \mathcal{I}_j$.

From earlier we have: $t\sigma \in \mathcal{P}$ and $t\sigma \notin \mathcal{F}_H \cap \mathcal{F}_{L_1} \cap \ldots \cap \mathcal{F}_{L_n} \cap \mathcal{I}_1 \cap \ldots \cap \mathcal{I}_n$.

As $(\forall j \in \{1, \ldots, n\})\ t\sigma \in \mathcal{F}_{L_j} \cap \mathcal{I}_j$, we have $t\sigma \in \mathcal{F}_{L_1} \cap \ldots \cap \mathcal{F}_{L_n} \cap \mathcal{I}_1 \cap \ldots \cap \mathcal{I}_n$.

$\Rightarrow t\sigma \notin \mathcal{F}_H$ and $t\sigma \in \mathcal{I}$.

$\Rightarrow t\sigma \notin \mathcal{F}_H$ and $t\sigma \in \mathcal{H}^p \cap \mathcal{I}$, as $\mathcal{P} = \mathcal{H}^p \cap \mathcal{L}_1^p \cap \ldots \cap \mathcal{L}_n^p \subseteq \mathcal{H}^p$.

We also have $t \in \mathcal{F}_H \cap \mathcal{H}^p \cap \mathcal{I}$ by **(1)**.

As Ψ is level-wise co-observable, we have:

$(\exists i \in D_{H,c}(\sigma))\ P_{H,i}^{-1}[P_{H,i}(t)]\sigma \cap \mathcal{F}_H \cap \mathcal{I} \cap \mathcal{H}^p = \emptyset$.

$\Rightarrow (\exists i \in D_c(\sigma))\ P_{H,i}^{-1}[P_{H,i}(t)]\sigma \cap \mathcal{F}_H \cap \mathcal{I} \cap \mathcal{H}^p = \emptyset$, as $D_{H,c}(\sigma) \subseteq D_c(\sigma)$.

Let $i \in D_c(\sigma)$ as specified in the above line and with the indicated properties.

$\Rightarrow P_i^{-1}[P_i(t)]\sigma \cap \mathcal{F}_H \cap \mathcal{I} \cap \mathcal{H}^p = \emptyset$, as $P_{H,i}(t)$: $\Sigma^* \rightarrow (\Sigma_{H,o,i})^*$ and $P_i(t)$: $\Sigma^* \rightarrow (\Sigma_{H,o,i})^*$.

We note that the above $i \in D_{H,c}(\sigma)$ represents the same decentralized supervisor as the $i \in D_c(\sigma)$. They thus have the same set of observable events, thus $P_{H,i} = P_i$.

$\Rightarrow P_i^{-1}[P_i(t)]\sigma \cap \mathcal{F} \cap \mathcal{I} \cap \mathcal{P} = \emptyset$, as $\mathcal{F} \cap \mathcal{I} \cap \mathcal{P} \subseteq \mathcal{F}_H \cap \mathcal{I} \cap \mathcal{H}^p$, thus $s \in \mathcal{F} \cap \mathcal{I} \cap \mathcal{P} \Rightarrow s \in \mathcal{F}_H \cap \mathcal{I} \cap \mathcal{H}^p$ which would cause a contradiction.

By Case 1) and 2), we have: $(\exists i \in D_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \mathcal{F} \cap \mathcal{I} \cap \mathcal{P} = \emptyset$.

As $t \in \mathcal{F} \cap \mathcal{I} \cap \mathcal{P}$ and $\sigma \in \Sigma_c$ are chosen arbitrarily, we conclude that Ψ is globally co-observable.

□

5.4.3 Complexity Analysis

In monolithic verification, the n low-level subsystems are composed directly with the high-level system without using the interface structure. The size of the state space for the monolithic method is the size of the product state space of $\mathbf{G}_H \parallel \mathbf{G}_{L_1} \parallel \ldots \parallel \mathbf{G}_{L_n}$. If the size of the state space of $\mathbf{G}_H$ is bounded by N_H, and the size of the state space for each $\mathbf{G}_{L_j}$ is bounded by N_L, then the size of the state space of the monolithic method is bounded by $N_H N_L^n$.

Our method verifies each component separately, therefore the size of the state space is bounded by the size of the component combined with its interface. For

$j = 1, ..., n$, if we assume that the size of the state space of $\mathbf{G}_{I_j}$ is bounded by N_I, then each low-level subsystem $\mathbf{G}_{L_j}||\mathbf{G}_{I_j}$ is bounded by $N_L N_I$. The high-level subsystem $\mathbf{G}_H||\mathbf{G}_{I_1}||\ldots||\mathbf{G}_{I_n}$ is bounded by $N_H N_I^n$. Therefore, our method is bounded by the larger of $N_H N_I^n$ and $N_L N_I$. Typically in an HIDSC design, the size of the high level is the limiting factor. This means that as long as $N_I \ll N_L$, we should achieve significant computational savings [LLW05].

5.5 MNDSC Supervisor Existence Theorem

We now present the marking nonblocking decentralized supervisory control (MNDSC) existence theorem, which shows that there exists an MNDSC to achieve the specification if and only if K is controllable and co-observable.

Note that in Theorem 5.5.1 bellow we do not require that K be $L_m(\mathbf{G})$-closed which is assumed by traditional decentralized control [CL08]. This will allow us to apply the result to our HIDSC system as we have an HISC nonblocking result but not an HISC $L_m(\mathbf{G})$-closed result.

Theorem 5.5.1. *Let* **Plant** $:= (Q,\ \Sigma,\ \delta,\ q_0,\ Q_m)$, $K \subseteq L_m(\mathbf{Plant})$, *and* $K \neq \emptyset$. *There exists an MNDSC* $\mathcal{S}_{Con}$ *for* (K, **Plant**) *such that* $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$ *if and only if* K *is controllable and co-observable with respect to* $L(\mathbf{Plant})$.

Proof. Let $K \subseteq L_m(\mathbf{Plant})$, $K \neq \emptyset$.

If part)

Assume K is controllable and co-observable with respect to $L(\mathbf{Plant})$.

We will show that there exists a marking nonblocking decentralized supervisory control $\mathcal{S}_{Con}$ for (K, **Plant**) such that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$.

We must first construct a suitable decentralized supervisory control $\mathcal{S}_{Con}$ for **Plant**.

For each $i \in D$ and $t \in L(\mathbf{Plant})$, we define the local decentralized supervisory control as follows:

$\mathcal{S}_{P_i}(t) := \Sigma_{uc} \cup (\Sigma_c \backslash \Sigma_{c,i}) \cup \{\sigma \in \Sigma_{c,i} \mid P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) \neq \emptyset\}$.

$\mathcal{S}_{P_i}$ enables all events in $\Sigma_{uc} \cup (\Sigma_c \backslash \Sigma_{c,i})$ and events in $\{\sigma \in \Sigma_{c,i} \mid P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) \neq \emptyset\}$. Only events in $\{\sigma \in \Sigma_{c,i} \mid P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) = \emptyset\}$ are not in $\mathcal{S}_{P_i}(t)$, for each $i \in D$.

The global decentralized supervisory control policy $\mathcal{S}_{Con}$ is defined as the conjunction of $\mathcal{S}_{P_i}$, $i \in D$, described as follows: $\mathcal{S}_{Con}(t) := \cap_{i=1}^{N} \mathcal{S}_{P_i}(t)$.

The language $L(\mathcal{S}_{Con}/\mathbf{Plant})$ is defined in Definition 5.1.2.

Clearly, $\mathcal{S}_{Con}$ is a decentralized supervisory control as defined in Definition 5.1.3. We will now show that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$ and that $\mathcal{S}_{Con}$ is nonblocking. To do this, our first step is to show that $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$.

Step 1.1) Show that $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$.

We will show that $L(\mathcal{S}_{Con}/\mathbf{Plant}) \subseteq \overline{K}$ and $\overline{K} \subseteq L(\mathcal{S}_{Con}/\mathbf{Plant})$.

Part A) Show that $L(\mathcal{S}_{Con}/\mathbf{Plant}) \subseteq \overline{K}$.

Let $t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$.

We will now prove by induction on the length of string t that $t \in \overline{K}$.

Base case: $t = \epsilon$.

We know that $\epsilon \in L(\mathcal{S}_{Con}/\mathbf{Plant})$ by definition. Further, $\epsilon \in \overline{K}$ since $K \neq \emptyset$ by assumption. We thus have $t \in \overline{K}$.

Inductive step: For $\sigma \in \Sigma$, we assume $t\sigma \in L(\mathcal{S}_{Con}/\mathbf{Plant})$ and $t \in \overline{K}$.

We will now show this implies $t\sigma \in \overline{K}$.

$\Rightarrow t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$, $(\forall i \in D)\, \sigma \in \mathcal{S}_{P_i}(t)$, and $t\sigma \in L(\mathbf{Plant})$, by definition of $L(\mathcal{S}_{Con}/\mathbf{Plant})$.

We note that we also have $t \in \overline{K}$, by inductive assumption.

We have two cases: $\sigma \in \Sigma_{uc}$ or $\sigma \in \Sigma_c$.

Case A.1) $\sigma \in \Sigma_{uc}$.

From above, we have: $t \in \overline{K}$, $\sigma \in \Sigma_{uc}$, and $t\sigma \in L(\mathbf{Plant})$.

As K is controllable, we have: $\overline{K}\Sigma_{uc} \cap L(\mathbf{Plant}) \subseteq \overline{K}$.

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

Case A.2) $\sigma \in \Sigma_c$.

From above, we have:

$t \in \overline{K}$, $\sigma \in \Sigma_c$, $(\forall i \in D)\, \sigma \in \mathcal{S}_{P_i}(t)$, and $t\sigma \in L(\mathbf{Plant})$.

We will show $t\sigma \in \overline{K}$ using proof by contradiction.

Assume $t\sigma \notin \overline{K}$.

$\Rightarrow t\sigma \in L(\mathbf{Plant}) \backslash \overline{K}$.

As K is co-observable with respect to $L(\mathbf{Plant})$, we have:

$(\exists i \in D_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) = \emptyset$.

$\Rightarrow (\exists i \in D_c(\sigma))\, \sigma \notin \mathcal{S}_{P_i}(t)$.

$\Rightarrow (\exists i \in D)\, \sigma \notin \mathcal{S}_{P_i}(t)$.

$\Rightarrow \sigma \notin \mathcal{S}_{Con}(t)$, by definition of $\mathcal{S}_{Con}(t)$.

$\Rightarrow t\sigma \notin L(\mathcal{S}_{Con}/\mathbf{Plant})$.

This is a contradiction. We thus conclude that $t\sigma \in \overline{K}$.

This completes the proof for the inductive step.

We thus conclude by induction that $L(\mathcal{S}_{Con}/\mathbf{Plant}) \subseteq \overline{K}$.

This completes the proof for part **A).**

Part B) Show that $\overline{K} \subseteq L(\mathcal{S}_{Con}/\mathbf{Plant})$.

Let $t \in \overline{K}$.

We will prove by induction on the length of string t that $t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$.

Base case: $t = \epsilon$.

We know that $\epsilon \in \overline{K}$ since $K \neq \emptyset$ by assumption. Further, we have $\epsilon \in L(\mathcal{S}_{Con}/\mathbf{Plant})$ by definition. We thus have $t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$.

Inductive step: For $\sigma \in \Sigma$, we assume $t\sigma \in \overline{K}$ and $t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$.

We will now show this implies $t\sigma \in L(\mathcal{S}_{Con}/\mathbf{Plant})$.

We thus have:

$t \in \overline{K}$ and $t\sigma \in L(\mathbf{Plant})$ by the assumption that $K \subseteq L_m(\mathbf{Plant}) \subseteq L(\mathbf{Plant})$.

We note that we also have $t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$ by the inductive assumption.

We have two cases: $\sigma \in \Sigma_{uc}$ or $\sigma \in \Sigma_c$.

Case B.1) $\sigma \in \Sigma_{uc}$.

$\Rightarrow \sigma \in \mathcal{S}_{Con}(t)$, as uncontrollable events are enabled by default for supervisor $\mathcal{S}_{Con}(t)$.

From above we have:

$t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$, $\sigma \in \mathcal{S}_{Con}(t)$ and $t\sigma \in L(\mathbf{Plant})$.

$\Rightarrow (\forall i \in D)\, \sigma \in \mathcal{S}_{P_i}(t)$ by definition of $\mathcal{S}_{Con}(t)$.

$\Rightarrow t\sigma \in L(\mathcal{S}_{Con}/\mathbf{Plant})$ by definition of $L(\mathcal{S}_{Con}/\mathbf{Plant})$.

Case B.2) $\sigma \in \Sigma_c$.

From above we have: $t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$, $t \in \overline{K}$, $t\sigma \in \overline{K}$, $\sigma \in \Sigma_c$, and $t\sigma \in L(\mathbf{Plant})$.

From the definition of $L(\mathcal{S}_{Con}/\mathbf{Plant})$, to show that $t\sigma \in L(\mathcal{S}_{Con}/\mathbf{Plant})$, it is sufficient to show that $(\forall i \in D)\, \sigma \in \mathcal{S}_{P_i}(t)$.

Let $i \in D$.

If $\sigma \notin \Sigma_{c,i}$, we immediately have: $\sigma \in \mathcal{S}_{P_i}(t)$ as $\sigma \in \Sigma_c \backslash \Sigma_{c,i}$.

We now consider $\sigma \in \Sigma_{c,i}$. It is sufficient to show that: $P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) \neq \emptyset$.

We first note that: $t \in P_i^{-1}[P_i(t)] := \{s \in \Sigma^* | P_i(s) \in \{P_i(t)\}\}$

$\Rightarrow t\sigma \in P_i^{-1}[P_i(t)]\sigma$.

As we have $t\sigma \in \overline{K}$ and $t\sigma \in L(\mathbf{Plant})$ from above, we have:

$t\sigma \in P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant})$.

$\Rightarrow P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) \neq \emptyset$.

We thus conclude $t\sigma \in L(\mathcal{S}_{Con}/\mathbf{Plant})$.

This completes the proof for the inductive step.

We thus conclude by induction that $\overline{K} \subseteq L(\mathcal{S}_{Con}/\mathbf{Plant})$.

This completes the proof for part **B).**

By part **A)** and part **B),** we have $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$.

This completes the proof for **Step 1.1)**.

Step 1.2) Show that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$.

By the definition of marking nonblocking decentralized supervisory control, we have: $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = L(\mathcal{S}_{Con}/\mathbf{Plant}) \cap K$.

Substituting $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$ (by the result of **Step 1.1)**, we have:

$L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K} \cap K = K$.

Step 1.3) Show that $\mathcal{S}_{Con}$ is nonblocking.

It is sufficient to show that $\overline{L_m(\mathcal{S}_{Con}/\mathbf{Plant})} = L(\mathcal{S}_{Con}/\mathbf{Plant})$.

The result is automatic since by **Step 1.1)** $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$ and by **Step 1.2)** $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$.

By **Steps 1.1)**, **1.2)** and **1.3)**, we conclude that there exists a marking nonblocking decentralized supervisory control $\mathcal{S}_{Con}$ for **Plant** such that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$.

If part is complete.

Only if part)

Assume there exists a marking nonblocking decentralized supervisory control $\mathcal{S}_{Con}$ for (K, **Plant**) such that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$.

We will now show this implies that K is controllable and co-observable with respect to $L(\mathbf{Plant})$.

We first note that as $\mathcal{S}_{Con}$ is nonblocking, $\overline{K} = \overline{L_m(\mathcal{S}_{Con}/\mathbf{Plant})} = L(\mathcal{S}_{Con}/\mathbf{Plant})$.

Step 2.1) Show that K is controllable with respect to $L(\mathbf{Plant})$.

Sufficient to show that $\overline{K}\Sigma_{uc} \cap L(\mathbf{Plant}) \subseteq \overline{K}$.

Let $t \in \overline{K}$, $\sigma \in \Sigma_{uc}$ and $t\sigma \in L(\mathbf{Plant})$.

$\Rightarrow t \in L(\mathcal{S}_{Con}/\mathbf{Plant})$ and $\sigma \in \mathcal{S}_{Con}(t)$, as $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$ and by the definition of $\mathcal{S}_{Con}(t)$.

$\Rightarrow t\sigma \in L(\mathcal{S}_{Con}/\mathbf{Plant})$, by definition of $L(\mathcal{S}_{Con}/\mathbf{Plant})$.

$\Rightarrow t\sigma \in \overline{K}$, as $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$.

$\Rightarrow \overline{K}\Sigma_u \cap L(\mathbf{Plant}) \subseteq \overline{K}$.

This completes the proof that K is controllable with respect to $L(\mathbf{Plant})$.

Step 2.2) Show that K is co-observable with respect to $L(\mathbf{Plant})$.

Sufficient to show that:

$$(\forall t \in \overline{K} \cap L(\mathbf{Plant}))\ (\forall \sigma \in \Sigma_c)\ t\sigma \in L(\mathbf{Plant}) \setminus \overline{K} \Rightarrow (\exists i \in D_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) = \emptyset.$$

Let $t \in \overline{K} \cap L(\mathbf{Plant})$, $\sigma \in \Sigma_c$ and $t\sigma \in L(\mathbf{Plant}) \setminus \overline{K}$.

$\Rightarrow t\sigma \in L(\mathbf{Plant})$ and $t\sigma \notin \overline{K}$.

$\Rightarrow t\sigma \notin L(\mathcal{S}_{Con}/\mathbf{Plant})$ as $L(\mathcal{S}_{Con}/\mathbf{Plant}) = \overline{K}$.

$\Rightarrow (\exists i \in D)\, \sigma \notin \mathcal{S}_{P_i}(t))$, by the definition of $L(\mathcal{S}_{Con}/\mathbf{Plant})$.

$\Rightarrow (\exists i \in D)\, (\sigma \in \Sigma_{c,i})$, $(P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) = \emptyset)$, by the definition of $\mathcal{S}_{P_i}(t)$, as only events in $\{\sigma \in \Sigma_{c,i} \mid P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) = \emptyset\}$ are not in $\mathcal{S}_{P_i}(t)$.

$\Rightarrow (\exists i \in D_c\,(\sigma))\; P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L(\mathbf{Plant}) = \emptyset$, by the definition of $D_c\,(\sigma)$.

This completes the proof that K is co-observable with respect to $L(\mathbf{Plant})$.

By **Step 2.1** and **Step 2.2** we conclude that K is controllable and co-observable with respect to $L(\mathbf{Plant})$.

Only if part is complete**.**

By **If part** and **Only if part**, we conclude that there exists a marking nonblocking decentralized supervisory control $\mathcal{S}_{Con}$ for (K, **Plant**) such that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = K$ if and only if K is controllable and co-observable with respect to $L(\mathbf{Plant})$.

□

We will now relate Theorem 5.5.1 to our HIDSC system and nonblocking. In essence, we are requiring Ψ to have equivalent MNDSC behaviour with $\mathcal{S}_{Con}/\mathbf{Plant}$, which ensures our HIDSC system implementation will be nonblocking.

Corollary 5.5.1. *Let Ψ be an HIDSC n^{th} degree decentralized specification interface system. Let* $\mathbf{Plant} := \mathbf{G}_H^p||\mathbf{G}_{L_1}^p||\ldots||\mathbf{G}_{L_n}^p$, $\mathbf{Spec} := \mathbf{F}_H||\mathbf{F}_{L_1}||\ldots||\mathbf{F}_{L_n}||\mathbf{G}_{I_1}||\ldots||\mathbf{G}_{I_n}$. *Let $L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant}) \neq \emptyset$. There exists an MNDSC $\mathcal{S}_{Con}$ for ($L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$, $\mathbf{Plant}$) such that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$, and $L(\mathcal{S}_{Con}/\mathbf{Plant}) = L(\mathbf{Spec}) \cap L(\mathbf{Plant})$, if and only if $L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$ is controllable and co-observable with respect to $L(\mathbf{Plant})$, and $\overline{L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})} = L(\mathbf{Spec}) \cap L(\mathbf{Plant})$.*

Proof.

If part)

Assume $L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$ is nonblocking, controllable and co-observable with respect to $L(\mathbf{Plant})$, and $\overline{L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})} = L(\mathbf{Spec}) \cap L(\mathbf{Plant})$.

Take $K = L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$ and we have by Theorem 5.5.1 there exists an MNDSC $\mathcal{S}_{Con}$ for $(L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant}), \mathbf{Plant})$ such that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$.

As $\mathcal{S}_{Con}$ is nonblocking by Theorem 5.5.1, we have:

$\overline{L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})} = \overline{L_m(\mathcal{S}_{Con}/\mathbf{Plant})} = L(\mathcal{S}_{Con}/\mathbf{Plant})$.

As $\overline{L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})} = L(\mathbf{Spec}) \cap L(\mathbf{Plant})$ by assumption, we have:

$L(\mathcal{S}_{Con}/\mathbf{Plant}) = L(\mathbf{Spec}) \cap L(\mathbf{Plant})$.

Only if part)

Assume There exists an MNDSC $\mathcal{S}_{Con}$ for $(L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant}), \mathbf{Plant})$ such that $L_m(\mathcal{S}_{Con}/\mathbf{Plant}) = L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$.

Take $K = L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$ and we have by Theorem 5.5.1 that $L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$ is controllable and co-observable with respect to $L(\mathbf{Plant})$.

As $\mathcal{S}_{Con}$ is nonblocking, we have:

$\overline{L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})} = \overline{L_m(\mathcal{S}_{Con}/\mathbf{Plant})} = L(\mathcal{S}_{Con}/\mathbf{Plant}) = L(\mathbf{Spec}) \cap L(\mathbf{Plant})$.

□

For HIDSC system Ψ, Corollary 5.5.1 tells us that the marked behaviour of our MNDSC and flat plant is equal to $L_m(\mathbf{Spec}) \cap L_m(\mathbf{Plant})$ and their closed behaviour is equal to $L(\mathbf{Spec}) \cap L(\mathbf{Plant})$. To apply Corollary 5.5.1, we need to first show that Ψ is co-observable, nonblocking, and controllable. For scalability, we want to verify all these global properties using only per-component properties. Theorem 5.4.1 shows

us that level-wise co-observability gives us global co-observability. Theorem 5.2.1 and 5.2.2 tell us that the HISC LD level-wise nonblocking, LD interface consistent, and level-wise controllability properties together imply that our flat system is nonblocking and controllable. We can thus verify all needed global properties using per-component checks. As we never need to construct the full system model, this offers potentially great computational savings.

Chapter 6

Example

To demonstrate the HIDSC method, we adapt a small manufacturing system from [Led02], that was originally modeled as an HISC system. See Figure 6.1. The system is composed of three manufacturing units running in parallel, a testing unit, material feedback, a packaging unit, plus three buffers to insure the proper flow of material.

Figure 6.2 shows which DES belong to the high-level subsystem ($\mathbf{G}_H$), high-level plant ($\mathcal{G}_H$), the high-level specification automata ($\mathcal{S}_H$), the j^{th} low-level subsystem ($\mathbf{G}_{L_j}$), the j^{th} low-level plant ($\mathcal{G}_{L_j}$), the j^{th} low-level specification automata ($\mathcal{S}_{L_j}$), and the j^{th} interface DES ($\mathbf{G}_{I_j}$), $j = \mathrm{I}, \mathrm{II}, \mathrm{III}$. We note that the three low-level subsystems shown in Figure 6.2 are identical up to relabeling. Figure 6.3 shows the low-level subsystems in detail.

Controllable events are those with a slash on the transition arrow, marked states are states with an unlabeled incoming arrow, and initial states are states with an unlabeled outgoing arrow.

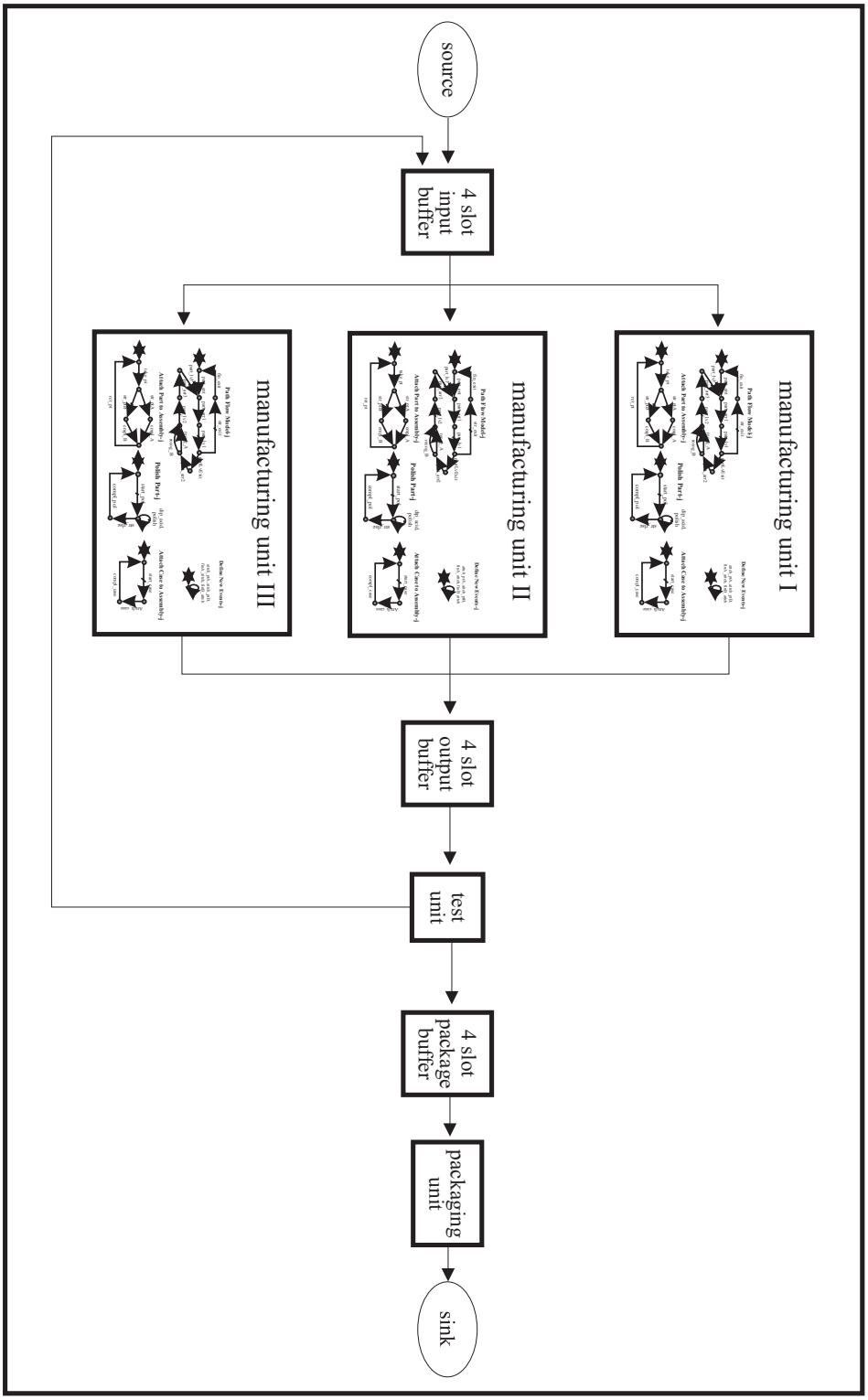
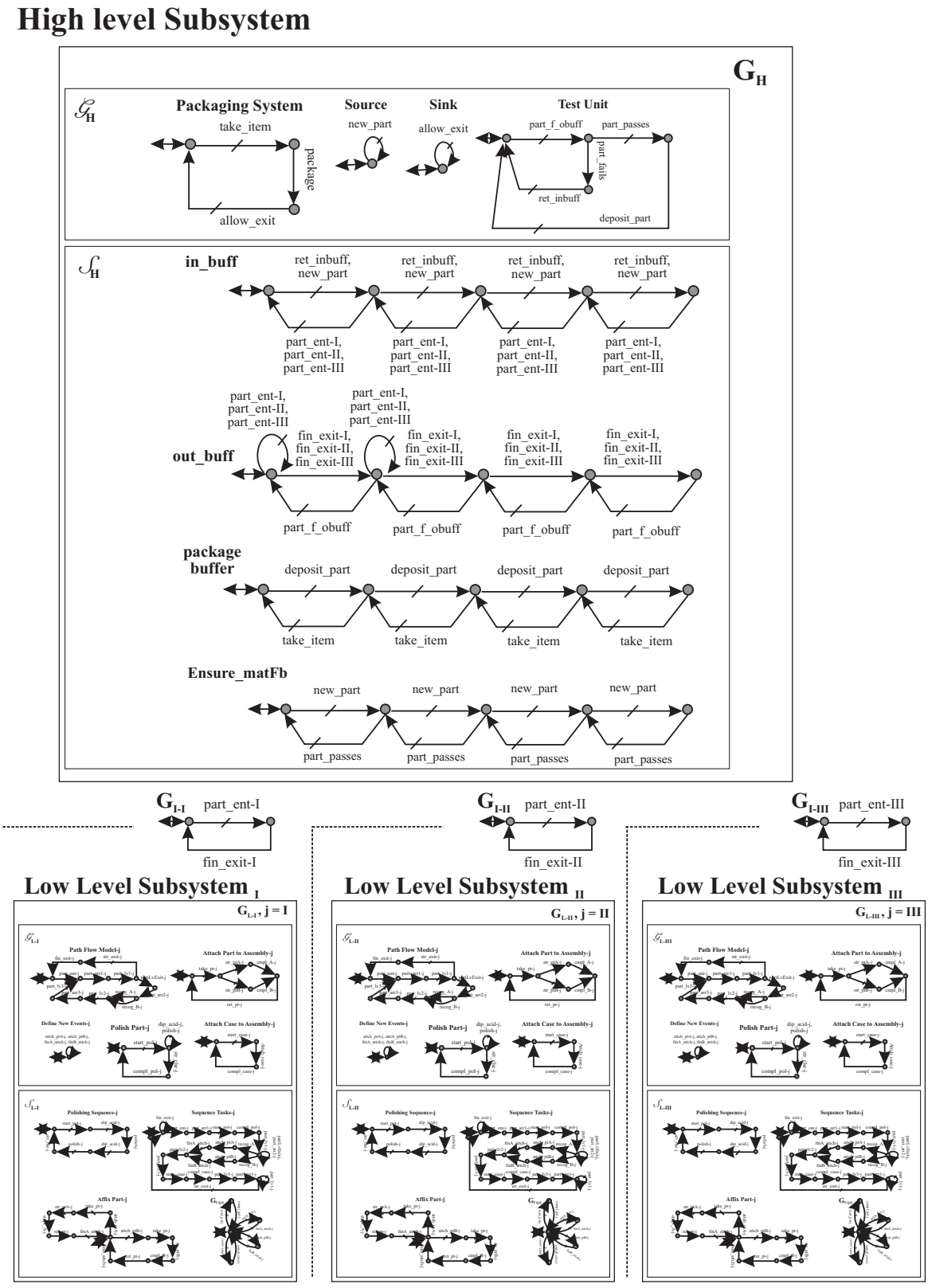


Figure 6.1: Block Diagram of Parallel Plant System.



108

Figure 6.2: Complete Parallel System.

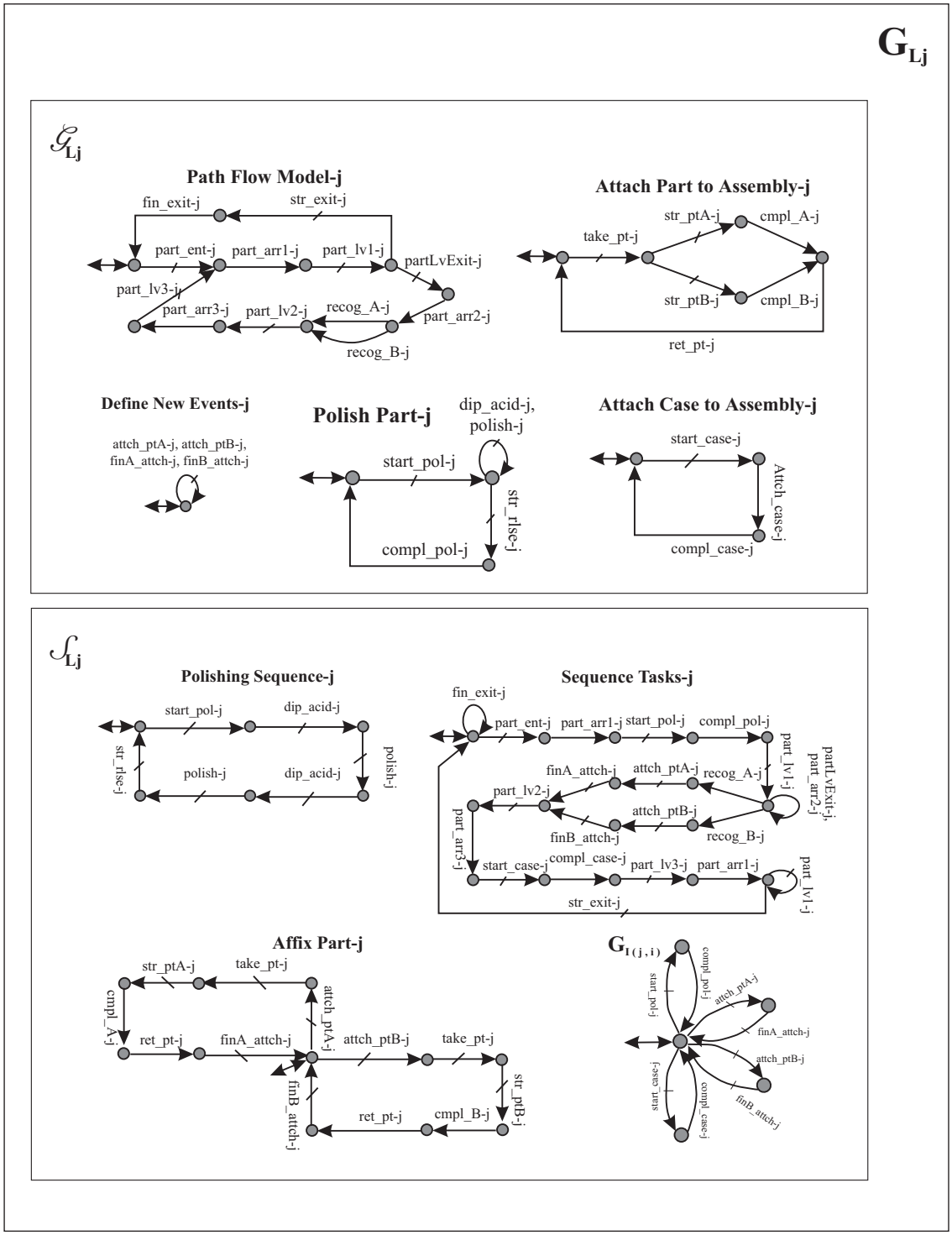


Figure 6.3: Low-Level Subsystem j.

6.1 Manufacturing System as an HIDSC

Originally this example was modeled as an HISC system. We will now adapt it to an HIDSC system. Typically, we would only do this if the system had an inherent distributed nature forcing us to implement supervisors with partial observations and partial controllability beyond the compartmentalized limitations imposed by the HISC structure.

We define the alphabet partition $\Sigma := [\dot{\cup}_{j\in\{\mathrm{I,II,III}\}}(\Sigma_{L_j}\dot{\cup}\Sigma_{R_j}\dot{\cup}\Sigma_{A_j})] \dot{\cup} \Sigma_H$ below:

$$\begin{aligned}\Sigma_H &= \{\textit{take\_item}, \textit{package}, \textit{allow\_exit},\\ &\quad \textit{new\_part}, \textit{part\_f\_obuff}, \textit{part\_passes},\\ &\quad \textit{part\_fails}, \textit{ret\_inbuff}, \textit{deposit\_part}\}\end{aligned}$$

$$\Sigma_{R_j} = \{\textit{part\_ent-j}\}$$

$$\Sigma_{A_j} = \{\textit{fin\_exit-j}\}$$

$$\begin{aligned}\Sigma_{L_j} \quad = \quad \{&\textit{start\_pol-j},\ \textit{attch\_ptA-j},\ \textit{attch\_ptB-j},\\ &\textit{start\_case-j},\ \textit{comp\_pol-j},\ \textit{finA\_attch-j},\\ &\textit{finB\_attch-j},\ \textit{compl\_case-j},\ \textit{part\_arr1-j},\\ &\textit{part\_lv1-j},\ \textit{partLvExit-j},\ \textit{str\_exit-j},\\ &\textit{part\_arr2-j},\ \textit{recog\_A-j},\ \textit{recog\_B-j},\\ &\textit{part\_lv2-j},\ \textit{part\_arr3-j},\ \textit{part\_lv3-j},\\ &\textit{take\_pt-j},\ \textit{str\_ptA-j},\ \textit{str\_ptB-j},\\ &\textit{compl\_A-j},\ \textit{compl\_B-j},\ \textit{ret\_pt-j},\\ &\textit{dip\_acid-j},\ \textit{polish-j},\ \textit{str\_rlse-j},\ \textit{Attch\_case-j}\}\end{aligned}$$

Our first step is to replace the existing supervisors with specification automata; thus let $\mathbf{F}_H = \mathcal{S}_H$ and $\mathbf{F}_{L_j} = \mathcal{S}_{L_j}$, $j = \mathrm{I}, \mathrm{II}, \mathrm{III}$.

Now we only consider a sample run for the low-level controllers for a single low-level subsystem, as the other low-level subsystems are identical up to relabel.

We next design decentralized controllers ($H1$, $H2$, L_{I_1}, L_{I_2}, L_{II_1}, L_{II_2}, L_{III_1}, L_{III_2}) to define our HIDSC problem.

For the high-level subsystem, the observable and controllable alphabet for controller $H1$ is specified as:

$\Sigma_{H,o,1} = \Sigma_I \cup \{\textit{new\_part}, \textit{ret\_inbuff}, \textit{part\_f\_obuff}\}$,

$\Sigma_{H,c,1} = (\Sigma_I \cap \Sigma_c) \cup \{\textit{part\_f\_obuff}\}$.

The observable and controllable alphabet for controller $H2$ is specified as:

$\Sigma_{H,o,2} = (\Sigma_I \cap \Sigma_{uc}) \cup \{\textit{take\_item}, \textit{package}, \textit{allow\_exit}, \textit{new\_part}, \textit{part\_passes},$

part_fails, *ret_inbuff*, *deposit_part*}

$\Sigma_{H,c,2} = \{take\_item, allow\_exit, new\_part, part\_f\_obuff, part\_passes, ret\_inbuff,$ *deposit_part*$\}$.

For the j^{th} low-level subsystem ($j = \mathrm{I}, \mathrm{II}, \mathrm{III}$), the observable alphabet for controllers L_{j_1} and L_{j_2} is specified as:

$\Sigma_{L,o,j1} = \{$*part_ent-j*, *fin_exit-j*, *start_pol-j*, *part_arr1-j*$\}$

$\Sigma_{L,o,j2} = \Sigma_{L_j}$

The controllable alphabet for controllers L_{j_1} and L_{j_2} is specified as:

$\Sigma_{L,c,j1} = \{$*part_ent-j*, *start_pol-j*$\}$

$\Sigma_{L,c,j2} = (\Sigma_{L_j} \cap \Sigma_c) \setminus \{$*start_pol-j*$\}$

The index sets of decentralized controllers for each component are: D_H=$\{H1$, $H2\}$, D_{L_I}=$\{L_{I_1}, L_{I_2}\}$, $D_{L_{II}}$=$\{L_{II_1}, L_{II_2}\}$, and $D_{L_{III}}$=$\{L_{III_1}, L_{III_2}\}$.

We now define the *flat plant,* and the *flat specification automata* as follows:

Plant := $\mathcal{G}_H||\mathcal{G}_{L_{\mathrm{I}}}||\mathcal{G}_{L_{\mathrm{II}}}||\mathcal{G}_{L_{\mathrm{III}}}$

Spec := $\mathbf{F}_H||\mathbf{F}_{L_I}||\mathbf{F}_{L_{II}}||\mathbf{F}_{L_{III}}||\mathbf{G}_{I_{\mathrm{I}}}||\mathbf{G}_{I_{\mathrm{II}}}||\mathbf{G}_{I_{\mathrm{III}}}$

6.2 Co-observability Verification for the Decentralized System

We now need to verify whether L_m(**Spec**) is co-observable w.r.t. L(**Plant**). We can then conclude, in combination with checking controllability and nonblocking, by Corollary 5.5.1 that there exists an MNDSC decentralized supervisory control and that its resulting closed-loop behaviour is the same as that of the flat system of our HIDSC system. By Theorem 5.4.1, we know that to check co-observability of the

HIDSC system, it is sufficient to verify level-wise co-observability.

The following steps for level-wise co-observability verification are:

Step 1. Verify whether the first low-level subsystem satisfies its portion of the level-wise co-observable definition, i.e., whether $L(\mathbf{F}_{L_\mathrm{I}}||\mathbf{G}_{I_\mathrm{I}})$ is co-observable w.r.t. $L(\mathbf{G}_{L_\mathrm{I}})$, $\Sigma_{L,c,i}$, $\Sigma_{L,o,i}$ for $i \in D_{L_I}$.

Using our research software tool, we verified that the first low-level component satisfies its portion of the level-wise co-observable definition. The monolithic verification ran for 5 hours without finishing, so we stopped it. The best run time of the incremental verification algorithm was 4.76 seconds. The state size for the low level was 550.

Step 2. Step 1 is sufficient to verify all three low levels as they are identical up to relabeling.

Step 3. Verify whether the high-level subsystem satisfies its portion of the level-wise co-observable definition, i.e., verifying whether $L(\mathbf{F}_H)$ is co-observable w.r.t. $L(\mathcal{G}_H||\mathbf{G}_{I_\mathrm{I}})$, $\Sigma_{H,c,i}$, $\Sigma_{H,o,i}$, for $i \in D_H$.

Using our research software tool, we verified that the high-level component satisfies its portion of the level-wise co-observable definition. The monolithic verification ran for 5 hours without finishing, so we stopped it. The best run time of the incremental verification algorithm was 424.78 seconds. The state size for the high level was 3120.

By completing Steps 1-3, we conclude that the decentralized system is level-wise co-observable, thus globally co-observable by Theorem 5.4.1. The total verification

run time was thus 429.54 seconds for a system whose complete system model has a state size of 2.78×10^{10}.

We applied our incremental verification algorithm to the entire system model (i.e. to the flat system), but our software still had not completed after 5 hours, so we stopped it.

Using our software tool DESpot [DES14], we verified that the system is level-wise controllable, LD level-wise nonblocking, and LD interface consistent. We can thus conclude by Theorem 5.2.1 and Theorem 5.2.2 that our flat system is nonblocking and controllable. We conclude by Corollary 5.5.1 that there exists a marking nonblocking decentralized supervisory control $\mathcal{S}_{Con}$ for **Plant**, and that **Spec**||**Plant** has equivalent MNDSC behaviour with $\mathcal{S}_{Con}$/**Plant**. This means that since **Spec**||**Plant** is nonblocking, $\mathcal{S}_{Con}$/**Plant** is also nonblocking.

6.3 Complexity Analysis for the Decentralized System

Applying DESpot to the small manufacturing system example, we found that the state size of the entire system was 2.78×10^{10}. However, the high-level state size was 3120 and the low-level state size was 550. As an HIDSC check only requires constructing a single component at a time, this is a potential savings of about seven orders of magnitude.

The computational complexity to verify co-observability using the monolithic approach in [RW95] is $O(|\Sigma||Y|^{2(N+2)})$, where Σ is the event set, Y is the state space,

and N is the number of decentralized controllers. Substituting in for the small manufacturing system example, verifying co-observability using the above method gives a computation bounded by $|42||2.78 \times 10^{10}|^{2(8+2)} = 3.19 \times 10^{210}$. Using our method, the computation is bounded by $|15||3120|^{2(2+2)} = 1.35 \times 10^{29}$. The potential computational saving is a 180 order of magnitude reduction.

Chapter 7

Conclusions and Future Work

In this chapter, we present our conclusions and discuss future work.

7.1 Conclusions

In decentralized control, agents have only a partial view and partial control of the system and must cooperate to achieve the control objective. In order to synthesize a decentralized control solution, a specification must satisfy the co-observability property. Existing co-observability verification methods require the possibly intractable construction of the complete system. To address this issue, we introduced an incremental verification of co-observability approach. Selected subgroups of the system are evaluated individually, until verification is complete. The new method is potentially much more efficient than the existing monolithic approaches, in particular for systems composed of many subsystems.

We applied our incremental algorithm to a small example of 114 states, made up of three plant components and 3 specifications. Each run used a different component

ordering and a different configuration of our selection heuristics. The fastest version had a speedup of 40.74 times over the monolithic algorithm, while the slowest was 20% slower. It was clear that we need to apply our software to a wider range of examples, and that we need to develop better heuristics, particularly with respect to the initial ordering of the components.

To further increase the scalability of decentralized control, we adapted the existing Hierarchical Interface-Based Supervisory Control (HISC) to support it. We introduced the Hierarchical Interface-Based Decentralized Supervisory Control (HIDSC) framework that included a per-component definition of co-observability that allows co-observability to be evaluated using only a single component at a time. As a result, the entire system model never needs to be constructed which potentially provides significant savings. Finally, we provided and proved the necessary and sufficient conditions for supervisory control existence in the HIDSC framework.

We applied our approach to a small manufacturing example. It contained a high level with 3120 states, three low levels with 550 states each, and a flat model with 2.78×10^{10} states. We verified the per-component co-observability property in 429.54 seconds. The incremental algorithm was used since the monolithic version failed to complete after five hours of trying to verify a low level. We applied the incremental algorithm to verify the HIDSC system as a flat model but our software still had not completed after 5 hours, so we stopped it.

7.2 Future Work

For incremental verification of co-observability, we need to apply our software to many more examples, in particular examples with more components, and larger components.

We also need to investigate better heuristics, in particular with respect to the initial ordering of the components. We would also like to extend our incremental verification method to state-based co-observability [Aga14].

For HIDSC framework, we recommend the following items. First, we recommend extending HIDSC from the current two level approach to a multi-level method to make HIDSC handle even larger systems in a flexible manner.

Second, if a component is not co-observable, then we could introduce communication to "allow" certain events to be observable [WVS96b, BL00, RC11], in order to make the whole system co-observable. We would call this HIDSC co-observability with communication.

Third, we would look into incorporating stronger properties such as strong decomposability [RW92b] in order to add synthesis of maximally permissive supervisors to HIDSC. Because strong decomposability implies co-observability, systems satisfying the strong decomposable property must also satisfy the co-observable property. As strong decomposability is closed under arbitrary union, it is thus readily adaptable to synthesis. It would thus be possible to automatically synthesize the supremal controllable, nonblocking and strongly decomposable supervisors for the HIDSC system.

Bibliography

[Aga14] Urvashi Agarwal. Symbolic decentralized supervisory control. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont., 2014. [ONLINE] Available: http://www.cas.mcmaster.ca/~leduc.

[AM96] M Antoniotti and B Mishra. NP-completeness of the supervisor synthesis problem for unrestricted CTL specifications. In *Proceedings of the third International Workshop on Discrete Event Systems*, Edinburgh, UK, Aug 1996.

[BL00] George Barrett and Stephane Lafortune. Decentralized supervisory control with communicating controllers. *IEEE Transactions on Automatic Control*, 45(9):1620–1638, 2000.

[BMM04] Bertil A Brandin, Robi Malik, and Petra Malik. Incremental verification and synthesis of discrete-event systems guided by counter examples. *IEEE Transactions on Control Systems Technology*, 12(3):387–401, 2004.

[BSW69] Keith A Bartlett, Roger A Scantlebury, and Peter T Wilkinson. A note

on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.

[CCdB93] M Courvoisier, M Combacau, and A de Bonneval. Control and monitoring of large discrete event systems: a generic approach (to EMS). In *Proceedings of IEEE International Symposium on Industrial Electronics*, pages 571–576, Budapest, Hungary, Jun 1993.

[CDFV88] Randy Cieslak, C Desclaux, Ayman S Fawaz, and Pravin Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988.

[CH00] Haoxun Chen and H-M Hansich. Model aggregation for hierarchical control synthesis of discrete event systems. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 1, pages 418–423, Sydney, Australia, December 2000.

[CL01] Yi-Liang Chen and Feng Lin. Hierarchical modeling and abstraction of discrete event systems using finite state machines with parameters. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 5, pages 4110–4115, Orlando, FL, USA, December 2001.

[CL08] Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems (second edition)*. Springer, 2008.

[DES14] DESpot. The official website for the DESpot project. [Online]. Available: http://www.cas.mcmaster.ca/~leduc/DESpot.html, 2014.

[DQC00] Max H De Queiroz and José ER Cury. Modular supervisory control of large scale discrete event systems. In *Proceedings of 5th International Workshop on Discrete Event Systems*, pages 103–110, Ghent, Belgium, Aug 2000.

[EC01] Jose M Eyzell and Jose ER Cury. Exploiting symmetry in the synthesis of supervisors for discrete event systems. *IEEE Transactions on Automatic Control*, 46(9):1500–1505, 2001.

[FCW09] Lei Feng, Kai Cai, and WM Wonham. A structural approach to the non-blocking supervisory control of discrete-event systems. *The International Journal of Advanced Manufacturing Technology*, 41(11-12):1152–1168, 2009.

[FM06a] Hugo Flordal and Robi Malik. Modular nonblocking verification using conflict equivalence. In *Proceedings 8th International Workshop on Discrete Event Systems*, pages 100–106, Ann Arbor, Michigan, USA, July 2006.

[FM06b] Hugo Flordal and Robi Malik. Supervision equivalence. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 155–160, Ann Arbor, Michigan, USA, July 2006.

[FM09] Hugo Flordal and Robi Malik. Compositional verification in supervisory control. *SIAM Journal on Control and Optimization*, 48(3):1914–1938, 2009.

[FMFÅ07] Hugo Flordal, Robi Malik, Martin Fabian, and Knut Åkesson. Compositional synthesis of maximally permissive supervisors using supervision equivalence. *Discrete Event Dynamic Systems*, 17(4):475–504, 2007.

[FW06a] Lei Feng and W Murray Wonham. Computationally efficient supervisor design: Abstraction and modularity. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 3–8, Ann Arbor, Michigan, USA, July 2006.

[FW06b] Lei Feng and WM Wonham. Computationally efficient supervisor design: control flow decomposition. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 9–14, Ann Arbor, Michigan, USA, July 2006.

[FW08] Lei Feng and WM Wonham. Supervisory control architecture for discrete-event systems. *IEEE Transactions on Automatic Control*, 53(6):1449–1461, 2008.

[GM04] B Gaudin and H Marchand. Modular supervisory control of a class of concurrent discrete event systems. In *Proceedings of 7th International Workshop on Discrete Event Systems*, pages 181–186, Reims, France, Sep 2004.

[GW00] Peyman Gohari and W. Murray Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 30(5):643–652, 2000.

[HC98] Paul Hubbard and Peter E Caines. Trace-dc hierarchical supervisory

control with applications to transfer-lines. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 3, pages 3293–3298, Tampa FL, USA, December 1998.

[HT06] Richard C Hill and Dawn M Tilbury. Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 399–406, Ann Arbor, Michigan, USA, July 2006.

[HT08] RC Hill and DM Tilbury. Incremental hierarchical construction of modular supervisors for discrete-event systems. *International Journal of Control*, 81(9):1364–1381, 2008.

[KvS06] Jan Komenda and Jan H van Schuppen. Optimal solutions of modular supervisory control problems with indecomposable specification languages. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 143–148, Ann Arbor, Michigan, USA, July 2006.

[KvSGM05] Jan Komenda, Jan H van Schuppen, Benoit Gaudin, and Hervé Marchand. Modular supervisory control with general indecomposable specification languages. In *Proceedings of 44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference*, pages 3474–3479, Seville, Spain, December 2005.

[KvSGM08] Jan Komenda, JH van Schuppen, Benoit Gaudin, and Hervé Marchand. Supervisory control of modular systems with global specification languages. *Automatica*, 44(4):1127–1134, 2008.

[KW95] P Kozak and WM Wonham. Fully decentralized solutions of supervisory control problems. *IEEE Transactions on Automatic Control*, 40(12):2094–2097, 1995.

[LBLW05] Ryan J. Leduc, Bertil A Brandin, Mark Lawford, and WM Wonham. Hierarchical interface-based supervisory control-part I: serial case. *IEEE Transactions on Automatic Control*, 50(9):1322–1335, 2005.

[LDS09] Ryan J. Leduc, Pengcheng Dai, and Raoguang Song. Synthesis method for hierarchical interface-based supervisory control. *IEEE Transactions on Automatic Control*, 54(7):1548–1560, 2009.

[Led02] R. J. Leduc. *Hierarchical Interface-based Supervisory Control*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont., 2002.

[Led09] Ryan J. Leduc. Hierarchical interface-based supervisory control with data events. *International Journal of Control*, 82(5):783–800, 2009.

[LLD06] Ryan J. Leduc, Mark Lawford, and Pengcheng Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Transactions on Control Systems Technology*, 14(4):654–668, 2006.

[LLMR14] Huailiang Liu, Ryan J. Leduc, Robi Malik, and S. L. Ricker. Incremental verification of co-observability in discrete-event systems. In *Proc. of 2014 American Control Conference*, pages 5446–5452, Portland, Oregon, USA, June 2014.

[LLR15] Huailiang Liu, Ryan J. Leduc, and S. L. Ricker. Hierarchical interface-based decentralized supervisory control. In *Proceedings of 54th IEEE Conference on Decision and Control, to appear*, Osaka, Japan, December 2015.

[LLW05] Ryan J. Leduc, Mark Lawford, and W Murray Wonham. Hierarchical interface-based supervisory control-part II: parallel case. *IEEE Transactions on Automatic Control*, 50(9):1336–1348, 2005.

[LW88a] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Infomation Sciience*, 44:173–198, 1988.

[LW88b] Feng Lin and W Murray Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44(3):199–224, 1988.

[LW90] Feng Lin and W Murray Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions on Automatic Control*, 35(12):1330–1337, 1990.

[MSR04] Robi Malik, David Streader, and Steve Reeves. Fair testing revisited: A process-algebraic characterisation of conflicts. In *Automated Technology for Verification and Analysis*, volume 3299, pages 120–134. Springer, 2004.

[OvS00] Ard Overkamp and Jan H van Schuppen. Maximal solutions in decentralized supervisory control. *SIAM Journal on Control and Optimization*, 39(2):492–511, 2000.

[PCL06] Patrıcia N Pena, José ER Cury, and Stéphane Lafortune. Testing modularity of local supervisors: An approach based on abstractions. In *Proceedings of 8th International Workshop on Discrete Event Systems*, pages 107–112, Ann Arbor, Michigan, USA, July 2006.

[PKK97] Joseph H Prosser, Moshe Kam, and Harry G Kwatny. Decision fusion and supervisor synthesis in decentralized discrete-event systems. In *Proceedings of the 1997 American Control Conference*, volume 4, pages 2251–2255, Albuquerque, New Mexico, USA, June 1997.

[QJ99] Robin G Qiu and Sanjay B Joshi. A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 29(6):573–586, 1999.

[RC11] Laurie Ricker and Benoit Caillaud. Mind the gap: Expanding communication options in decentralized discrete-event control. *Automatica*, 47(11):2364–2372, 2011.

[Ric08] SL Ricker. Asymptotic minimal communication for decentralized discrete-event control. In *9th International Workshop on Discrete Event Systems, 2008. WODES 2008.*, pages 486–491, 2008.

[Rud92] Karen Rudie. *Decentralized Control of Discrete-event Systems*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont., 1992.

[RW92a] K. Rudie and W. M. Wonham. Protocol verification using discrete-event systems. In *Proc. 31st Conf. Decision Contr.*, pages 3770–3777, 1992.

[RW92b] Karen Rudie and W Murray Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, 1992.

[RW95] Karen Rudie and Jan C Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40(7):1313–1319, 1995.

[RYL03] Kurt Rohloff, Tae-Sic Yoo, and Stéphane Lafortune. Deciding coobservability is PSPACE-complete. *IEEE Transactions on Automatic Control*, 48(11):1995–1999, 2003.

[SB01] Geert Stremersch and RK Boel. Decomposition of the supervisory control problem for Petri nets under preservation of maximal permissiveness. *Automatic Control, IEEE Transactions on*, 46(9):1490–1496, 2001.

[SB11] Klaus Schmidt and Christian Breindl. Maximally permissive hierarchical control of decentralized discrete event systems. *IEEE Transactions on Automatic Control*, 56(4):723–737, 2011.

[SC02] Gang Shen and Peter E Caines. Hierarchically accelerated dynamic programming for finite-state machines. *IEEE Transactions on Automatic Control*, 47(2):271–283, 2002.

[SM06] Klaus Schmidt and Thomas Moor. Marked-string accepting observers for the hierarchical and decentralized control of discrete event systems. In

Proceedings of 8th International Workshop on Discrete Event Systems, pages 413–418, Ann Arbor, Michigan, USA, July 2006.

[SMP08] Klaus Schmidt, Thomas Moor, and Sebastian Perk. Nonblocking hierarchical control of decentralized discrete event systems. *IEEE Transactions on Automatic Control*, 53(10):2252–2265, 2008.

[SW04] Rong Su and W Murray Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, 14(1):31–53, 2004.

[TKU94] Shigemasa Takai, Shinzo Kodama, and Toshimitsu Ushio. Decentralized state feedback control of discrete event systems. *Systems & control letters*, 22(5):369–375, 1994.

[TKU05] Shigemasa Takai, Ratnesh Kumar, and Toshimitsu Ushio. Characterization of co-observable languages and formulas for their super/sublanguages. *IEEE Transactions on Automatic Control*, 50(4):434–447, 2005.

[TL09] JG Thistle and HM Lamouchi. Effective control synthesis for partially observed discrete-event systems. *SIAM Journal on Control and Optimization*, 48(3):1858–1887, 2009.

[Tri04] Stavros Tripakis. Undecidable problems of decentralized observation and control on regular languages. *Information Processing Letters*, 90(1):21–28, 2004.

[Tsi89] John N Tsitsiklis. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals and Systems*, 2(2):95–107, 1989.

[TU00] Shigemasa Takai and Toshimitsu Ushio. Reliable decentralized supervisory control of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 30(5):661–667, 2000.

[Won14] W.M. Wonham. Supervisory control of discrete-event systems. Department of Electrical and Computer Engineering, University of Toronto, July 2014. [Online] Available: http://www.control.toronto.edu/DES/.

[WR88] W Murray Wonham and Peter J Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of control, Signals and Systems*, 1(1):13–30, 1988.

[WvS96a] K. C. Wong and J. H. van Schuppen. Decentralized supervisory control of discrete-event systems with communication. In *Proc. Int. Workshop on Discrete Event Systems*, pages 284–289, 1996.

[WVS96b] KC Wong and JH Van Schuppen. Decentralized supervisory control of discrete-event systems with communication. *Report-Department of Operations Research, Statistics, and System Theory*, (6):1–10, 1996.

[WW96] Kai C Wong and W Murray Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems*, 6(3):241–273, 1996.

[WW98] Kai C Wong and W Murray Wonham. Modular control and coordination of discrete-event systems. *Discrete Event Dynamic Systems*, 8(3):247–297, 1998.

[YL02a] T-S Yoo and Stéphane Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems*, 12(3):335–377, 2002.

[YL02b] Tae-Sic Yoo and Stephane Lafortune. NP-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1495–1499, 2002.

[YL04] Tae-Sic Yoo and Stéphane Lafortune. Decentralized supervisory control with conditional decisions: Supervisor existence. *IEEE Transactions on Automatic Control*, 49(11):1886–1904, 2004.

[ZW90] Hao Zhong and W Murray Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, 35(10):1125–1134, 1990.