Abstract

Efficient Abstractions in Hierarchical Supervisory Control of Discrete-Event Systems

Xinquan Qu
Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto
2013

In this thesis we study two problems in hierarchical supervisory control of discrete-event systems. The first problem is controller reduction, which is to find smaller abstractions of hierarchical controllers for easier understanding. The second problem is controller design, which is to find smaller abstractions of systems for high-level controller design. We employ natural projections to generate abstractions, thus incorporate the two problems into one optimization problem in which we try to minimize the size of the observable event set of the abstractions, subject to the constraints that the abstractions still achieve maximally permissive and nonblocking control. An optimal solution is proposed for the controller reduction problem, and a suboptimal solution is proposed for the controller design problem.
Acknowledgement

First of all I would like to express my sincere gratitude to my advisor Professor Wonham, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. Without his tenacious support, I would not have accomplished my research. I could not have imagined having a better supervisor for my graduate study.

Secondly, I would like to express my thanks to Kai Cai. In the May of Year 2013, he spent lots of time to guide my research. When I had problems of writing my thesis, he generously helped me to revise my paper. Furthermore, Kai shared his computer with me when my laptop was stolen. Without his help, I would not have accomplished my work in time.

Thirdly I would like to thank Jimmy Tsai, who helped me a lot in my study. He helped me to revise my thesis in the first several rounds to make it better. When my laptop was stolen, he generously shared his laptop with me for my presentation. He also invited me to lots of activities, which made my campus life much more interesting.

Fourthly, I would like to thank my colleagues in the lab. You make this lab such a perfect place for research. I am a lucky one to be able to become a member of this great group.
Finally, I would like to express my gratitude to my parents. It is your unconditional support to make me finally accomplish this much.
Contents

1 Introduction .................................................. 1

1.1 Hierarchical Supervisory Control ....................... 2

1.2 Preliminaries on Discrete-Event Systems ............... 4

1.3 Notation .................................................... 7

2 Problem Formulation ........................................... 9

2.1 Literature Review of Hierarchical Supervisory Control . 10

2.2 Hierarchical Supervisory Control Architectures .......... 12

2.2.1 Hierarchical Supervisory Control ..................... 13

2.2.2 Vertical and Decentralized Supervisory Control .... 15

2.3 Problem Formulation ........................................ 17
3 Building Efficient Abstractions

3.1 Monolithic Controller Is Known

3.1.1 Theoretical Analysis

3.1.2 Computation Procedure and Example

3.2 Monolithic Controller Is Unknown

3.2.1 Problem Analysis

3.2.2 Closed Specification Solution

3.2.3 General Specification Solution

3.2.4 Comparison with Previous Results

3.2.5 Computational Procedure and Examples

3.3 Chapter Summary and Conclusions

4 Application

4.1 Abstractions of Hierarchical Controllers

4.2 A Triple Loop Example
4.3 Chapter Summary and Conclusions .......................... 68

5 Conclusions and Future Work ............................ 69

5.1 Main Contributions ........................................ 70

5.2 Future Work ............................................. 71

5.3 Final Remark ............................................. 72

Bibliography .................................................... 73

Appendices ...................................................... 79

A Proofs ........................................................ 80

A.1 Proofs for Section 3.1 ..................................... 80

A.1.1 Proposition 2 ........................................... 80

A.1.2 Lemma 1 ................................................ 81

A.1.3 Lemma 2 ................................................ 81

A.1.4 Theorem 1 .............................................. 81

A.2 Proofs for Section 3.2 ..................................... 82
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.2.17 Proposition 9</td>
<td>90</td>
</tr>
<tr>
<td>A.2.18 Proposition 10</td>
<td>91</td>
</tr>
<tr>
<td>A.3 Proofs for Section 4.1</td>
<td>91</td>
</tr>
<tr>
<td>A.3.1 Theorem 9</td>
<td>91</td>
</tr>
</tbody>
</table>
List of Tables

3.1 Algorithm 1 .................................................. 24
3.2 Algorithm 2 .................................................. 45
3.3 Algorithm 3 .................................................. 46
4.1 Three Types of Controllers ............................... 66
4.2 Comparison of Abstraction Reduction Methods ....... 67
List of Figures

1.1 Hierarchical Control ............................................. 2

1.2 Two Systems ....................................................... 7

1.3 Notation for Marked States ................................. 8

2.1 Hierarchical Control ............................................. 14

2.2 Vertical and Decentralized Supervisory Control .......... 15

2.3 Two Scenarios for Problem 2.1 ............................... 18

3.1 $E$ is Normal w.r.t. $(G, P_o)$ ............................. 22

3.2 Transfer-Line System ........................................... 25

3.3 Machines and Buffers .......................................... 25

3.4 Local Plants and Local Specifications ..................... 26

3.5 Local Controllers ............................................... 27
3.6 \( M = L(G), E = L_m(G) \) ........................................ 28

3.7 \( G(K) \) ................................................................. 28

3.8 \( G(K^{hi}) \) ............................................................ 29

3.9 Boundary Strings and Events of \( L \) .............................. 33

3.10 Natural Observer ...................................................... 34

3.11 Events of Boundaries of \( E, K \) w.r.t. \( M \) ...................... 35

3.12 Generators for \( K_1 \) and \( K_2 \) ................................. 47

3.13 \( M = L(G), E = L_m(G) \) ........................................ 48

3.14 \( M^{hi} = L(G), E^{hi} = L_m(G) \) ............................... 49

3.15 \( G(K^{hi}) \) ............................................................ 49

3.16 \( G(K) \) ................................................................. 50

3.17 Transfer-Line System ................................................ 51

3.18 Machines and Buffers ............................................... 51

3.19 Local Controllers .................................................... 52

3.20 \( M = L(G), E = L_m(G) \) ........................................ 52

3.21 \( M^{hi} = L(G), E^{hi} = L_m(G) \) ............................... 53
3.22 $G(K^{hi})$ .................................................. 53

4.1 Hierarchical Controllers ................................. 56
4.2 Triple Loop System ........................................ 59
4.3 Machines ..................................................... 60
4.4 Buffers ....................................................... 60
4.5 Three-Level Hierarchical Architecture ............... 62
4.6 Abstractions of Hierarchical Controllers ............ 66
## List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Σ</td>
<td>Event set</td>
<td>4</td>
</tr>
<tr>
<td>Σ*</td>
<td>All finite strings of Σ</td>
<td>4</td>
</tr>
<tr>
<td>ε</td>
<td>Empty string</td>
<td>4</td>
</tr>
<tr>
<td>s₁ ≤ s₂</td>
<td>String s₁ is a prefix of string s₂</td>
<td>4</td>
</tr>
<tr>
<td>·</td>
<td>Prefix closure</td>
<td>4</td>
</tr>
<tr>
<td>Σ₀</td>
<td>Observable subset of Σ</td>
<td>4</td>
</tr>
<tr>
<td>P</td>
<td>Natural projection</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≡ₘ</td>
<td>Nerode equivalence</td>
<td>5</td>
</tr>
<tr>
<td>A</td>
<td>Automaton</td>
<td>5</td>
</tr>
<tr>
<td>X</td>
<td>States of an automaton</td>
<td>5</td>
</tr>
<tr>
<td>ξ</td>
<td>Transition function of an automaton</td>
<td>5</td>
</tr>
<tr>
<td>x₀</td>
<td>Initial state of an automaton</td>
<td>5</td>
</tr>
<tr>
<td>Xₘ</td>
<td>Marked states of an automaton</td>
<td>5</td>
</tr>
<tr>
<td>G</td>
<td>Generator</td>
<td>5</td>
</tr>
<tr>
<td>L(G)</td>
<td>Language generated by G</td>
<td>6</td>
</tr>
<tr>
<td>Lₘ(G)</td>
<td>Marked language generated by G</td>
<td>6</td>
</tr>
<tr>
<td>Σₖ</td>
<td>Controllable event set</td>
<td>6</td>
</tr>
<tr>
<td>Σₙ</td>
<td>Uncontrollable event set</td>
<td>6</td>
</tr>
<tr>
<td>E</td>
<td>Language of a specification</td>
<td>6</td>
</tr>
<tr>
<td>M</td>
<td>Language of a plant</td>
<td>6</td>
</tr>
<tr>
<td>C(·)</td>
<td>Controllable sublanguages</td>
<td>6</td>
</tr>
<tr>
<td>supC(·, ·, ·)</td>
<td>Supremal controllable sublanguage</td>
<td>6</td>
</tr>
<tr>
<td>Symbol</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Controllable event</td>
<td>8</td>
</tr>
<tr>
<td>$\beta$</td>
<td>Uncontrollable event</td>
<td>8</td>
</tr>
<tr>
<td>$Q$</td>
<td>Natural projection</td>
<td>16</td>
</tr>
<tr>
<td>$</td>
<td>\cdot</td>
<td>$</td>
</tr>
<tr>
<td>$G(\cdot)$</td>
<td>Generator of a regular language</td>
<td>25</td>
</tr>
<tr>
<td>$\mathcal{N}$</td>
<td>Natural number set</td>
<td>41</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This thesis provides a thorough analysis of hierarchical supervisory control. Specifically the problems of hierarchical controller reduction and hierarchical controller design are incorporated into one optimization problem. A complete solution to the projection-based controller reduction problem is proposed, and a suboptimal solution to the controller design is provided. To the best of our knowledge, a complete solution to controller reduction has not yet been reported elsewhere. Although our solution is suboptimal for the design problem, the conditions are weaker than the conditions given by other recent work.

Section 1.1 introduces hierarchical supervisory control. Section 1.2 presents basic information on discrete-event systems. Section 1.3 provides the notation we will use in this thesis.
1.1 Hierarchical Supervisory Control

A realistic discrete-event system is often composed of multiple local systems with interacting plant components subject to local specifications [2, 7, 11]. If the number of plant components and specifications is large, monolithic control with the Ramadge-Wonham approach [42] may well demand an infeasible amount of computational resources, though it guarantees the maximally permissive and nonblocking control. Furthermore, even if it is feasible to accomplish monolithic supervisory control, the controller in a realistic application may be too complicated to be readily understood.

On the other hand, hierarchical supervisory control organizes the system [34] by dividing the system into multiple levels where each level consists of several small subsystems. In each level, controllers (or coordinators) are synthesized to supervise the local systems, and their abstractions are further used to represent the subsystems in high-level supervisory control. Therefore the supervisory control of a large system can be achieved by multiple small hierarchical controllers. An illustration of hierarchical control is shown in Fig. 1.1, where three hierarchical controllers $K_{sub_1}, K_{sub_2}, K_{sub_3}$...
are designed in two levels for five local systems. Specifically $K_{\text{sub}_1}, K_{\text{sub}_2}$
is designed in the first level to supervise systems 1, 2 and 4, 5 respectively. Furthermore the two controllers are further used in the second level supervisory control, where the systems 1, 2, 4, 5 are not needed; thus the supervisory control problem may be significantly smaller than monolithic supervisory control.

There have been some sufficient conditions [12, 29] proposed to build the hierarchical controllers for maximally permissive and nonblocking control. Nevertheless it is still an open problem to find weaker conditions for hierarchical supervisory control. Furthermore, although the hierarchical controllers are less complex than the monolithic controller, they may still be prohibitively complicated for understanding. Various methods have been proposed either to decompose the hierarchical controllers into multiple local ones [3, 4] or to minimise the size of generators of the hierarchical controllers [38, 39]. However, the decomposition approach demands a great deal of computational time, while the reduction approach may be limited in simplifying the controller.

The two problems, namely the hierarchical supervisory controller reduction (comprehension) problem and the design problem, are closely related. In each level of the hierarchical architecture, controllers are synthesized to supervise the subsystems, and their abstractions further represent the subsystems in the high-level supervisory control. In the end, when all hierarchical controllers are designed, then smaller abstractions can be computed for understanding. Therefore efficient abstractions of controllers are the critical part of hierarchical controller reduction and design.

In this thesis, we focus on efficient abstractions in hierarchical supervisory control. The controller reduction problem is to find smaller abstractions of the hierarchical controllers for easier understanding. The controller design problem is to find smaller abstractions of systems for the high-level supervisory control. We incorporate the two problems into
one optimization problem to find a smaller abstraction by projecting out as many unnecessary events as possible. We propose an optimal solution for the controller reduction problem, and a suboptimal solution for the controller design problem.

1.2 Preliminaries on Discrete-Event Systems

Discrete-Event Systems are based on languages. Particularly a special type of languages called regular languages are interesting since they can be modeled by finite automata for computation. The concepts of language, automaton, generator and controllability will be introduced below based on [26],[42],[6].

For a finite alphabet $\Sigma$, the set of all nonempty finite strings over $\Sigma$ is denoted as $\Sigma^+$. The empty string is written as $\epsilon$, thus the set of all finite strings over $\Sigma$ is denoted as $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$. A language over $\Sigma$ is any subset of $\Sigma^*$. The operation concatenation $\text{cat} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ is defined as $s_1 = \sigma_1 \ldots \sigma_n, s_2 = \sigma_{n+1} \ldots \sigma_{n+m}$, $\text{cat}(s_1, s_2) = \sigma_1 \ldots \sigma_n\sigma_{n+1} \ldots, \sigma_{n+m}$.

We simply write the concatenation as $\text{cat}(s_1, s_2) = s_1s_2$. A partial order is defined on strings as $s_1 \leq s_2 \iff (\exists s_3 \in \Sigma^*, s_2 = s_1s_3)$. We also call $s_1$ a prefix of $s_2$. The prefix closure of a language $L$ is $\overline{L} = \{s \in \Sigma^*|\exists t \in L$ such that $s \leq t\}$. If a language equals its prefix closure, then we call it a (prefix) closed language.

Let $\Sigma_o \subseteq \Sigma$ be a subset of observable events. The natural projection $P : \Sigma \rightarrow \Sigma_o^*, \Sigma_o \subseteq \Sigma$ is defined inductively as

$$P(\epsilon) = \epsilon;$$
$$P(s\sigma) = \begin{cases} P(s) & \text{if } \sigma \notin \Sigma_o; \\ P(s)\sigma & \text{if } \sigma \in \Sigma_o. \end{cases}$$
The set-valued inversion of natural projection $P^{-1} : \Sigma^* \rightarrow 2^{\Sigma^*}$ is defined as $P^{-1}(t) = \{s | P(s) = t\}$.

Let $P_i : (\bigcup_{i=1}^n \Sigma_i)^* \rightarrow \Sigma_i^*$, $i = 1, \ldots, n$. The set-valued synchronous product of multiple languages $L_1 \subseteq \Sigma_1^*, \ldots, L_n \subseteq \Sigma_n^*$ is defined as $\big\|_{i=1}^n L_i = \{s | P_i(s) \in L_i, i = 1, \ldots, n\}$. Two languages $L_1, L_2$ are called synchronously nonconflicting if $L_1 \big\| L_2 = L_1 \big\| L_2$.

The Nerode equivalence relation is defined over $(\Sigma, L)$ as: given $s, t \in \Sigma^*$ if $(\forall u \in \Sigma^*$), $su \in L \iff tu \in L$, then $s \equiv_L t$. As an equivalence relation generates a partition over $\Sigma^*$, thus Nerode equivalence inherently induces cells(equivalence classes) of strings in $\Sigma^*$. If the number of cells induced by Nerode equivalence on $(\Sigma, L)$ is finite, then we call $L$ a regular language.

A recognizer is a 5-tuple $(X, \Sigma, \xi, x_0, X_m)$, where $X = \Sigma^*/\equiv_L$ is the set of cells of the partition over $\Sigma^*$ induced by Nerode equivalence, $x_o \in X$ is the cell to which $\epsilon$ belongs, $X_m \subseteq X$ is called the marked states. Corresponding to the strings $s \in L$, since $X$ is a set of cells of a partition over $\Sigma^*$, thus a function $P_L$ is defined as $\forall s \in \Sigma^*$ that $\exists! x \in X^1$ such that $P_L(s) = x$. Additionally we define the identity function by $id_\Sigma : \Sigma \rightarrow \Sigma$, such that $id_\Sigma(\sigma) = \sigma$. It has been proved that the transition function $\xi : X \times \Sigma \rightarrow X$ is uniquely defined by $\xi \circ (P_L \times id_\Sigma) = P_L \times \text{cat}$, $\xi(\epsilon) = x_0$ [42]. If $X$ is finite, then the 5-tuple is called a finite automaton. We use $L_m(A)$ to refer to the marked language of automaton $A$, which is the set of strings that reach the marked states $X_m$ from $x_0$.

An automaton $A$ can be represented as a directed graph, where $X$ is the set of nodes and $\xi$ is the set of edges. The two concepts of reachable state and coreachable state are introduced now. For a state $x$ to be reachable, means that there is a path from $x_0$ to $x$. For a state $x$ to be coreachable, means there is a path from $x$ to $X_m$. Suppose $L$ is the marked language of $A$. If we only keep the reachable and coreachable states of $X$, then the

$^1\exists!$ means uniquely exists
remaining part of the automaton is called a generator, which satisfies the following equations.

\[ L = L_m(G) \]
\[ \overline{L} = L(G) \]

Therefore we see that a generator completely characterizes the language \( L \) and its prefix closure \( \overline{L} \).

Now we introduce the concepts of controllability. Controllable events \( \Sigma_c \) is a subset of \( \Sigma \), which can be disabled by a supervisor. The complement of the controllable event set \( \Sigma_c \) is defined as uncontrollable event set \( \Sigma_u \). Given a closed language \( M \subseteq \Sigma^* \), the definition of controllability of a language \( K \) w.r.t. \( (M, \Sigma_u) \) is given below:

**Definition 1.** A language \( K \) is controllable w.r.t. \( (M, \Sigma_u) \) if

\[ \overline{K}\Sigma_u \cap M \subseteq \overline{K} \]

If \( K \subseteq E \) is controllable w.r.t. \( (M, \Sigma_u) \), and \( s \in \overline{K} \), while \( s\sigma \in \overline{E} - \overline{K} \), then it’s said that \( \sigma \) is disabled at \( s \). In the above equations, \( \overline{K}\Sigma_u := \{s\sigma_u|s \in \overline{K} \text{ and } \sigma_u \in \Sigma_u\} \). For any \( s \in \overline{K} \) and any uncontrollable event \( \sigma_u \in \Sigma_u \), if \( s\sigma_u \in M \), then the string \( s\sigma_u \) can be generated by the plant. As no supervisor can disable an uncontrollable event, thus the string pertaining to \( s\sigma_u \) cannot be prevented. Therefore, \( \overline{K} \) must contain \( s\sigma_u \) in order that the strings of \( K \) can be ensured by a supervisor.

In a DES, all feasible strings will be modeled by plant \( M \), and the tasks or constraints of the system are modeled by specification \( E \). The subfamily of languages \( C(E) = \{K'|K' \subseteq E, \overline{K'}\Sigma_u \cap M \subseteq \overline{K'}\} \) is the collection of all controllable sublanguages of \( E \) w.r.t. \( M \). The supremal language \( supC(E) \) is also a controllable sublanguage of \( E \) [42]. We denote the supremal controllable sublanguage of \( E \) as \( supC(M, E, \Sigma_u) \).
1.3 Notation

In this section, we introduce notation which will be used throughout this thesis.

A typical system consists of many local plant components and specifications. In this thesis we form a local plant and local specification for each local system, instead of using multiple plant components. For example, in Fig. 1.2, there are two local systems which are encircled by the dashed ovals. Their events are shown above the transitions, e.g. $\alpha_1, \beta_1$ are the events of $G_1$. In the first subsystem we incorporate the components $G_1, G_2$ into the plant $M_1$, and the marked language of $G_1, G_2, B_1$ into the specification $E_1$. Similarly we incorporate the components $G_2, G_3, B_2$ into $M_2, E_2$ in the second subsystem. The formal expressions are given below.

\[
M_1 := L(G_1) \parallel L(G_2), \quad E_1 := L_m(G_1) \parallel L_m(G_2) \parallel L_m(B_1)
\]
\[
M_2 := L(G_2) \parallel L(G_3), \quad E_2 := L_m(G_2) \parallel L_m(G_3) \parallel L_m(B_2)
\]

Figure 1.2: Two Systems
A local controller is built w.r.t. each local plant and its specification such that

\[ K_1 := \text{supC}(M_1, E_1, \Sigma_{1,u}) \]
\[ K_2 := \text{supC}(M_2, E_2, \Sigma_{2,u}) \]

where \( \Sigma_{i,u}, i = 1, 2 \) is the uncontrollable event set of the \( i \)th local system.

![Figure 1.3: Notation for Marked States](image)

Throughout this thesis, we use \( K \) to refer to the supremal controllable sublanguage of \((M, E)\). The components of generators contributing to the plant \( M \) are denoted as \( G_i \). The components of generators exclusively contributing to the specification \( E \) are denoted as \( B_j \).

The controllable events are denoted as \( \alpha_i \). The uncontrollable events are denoted as \( \beta_j \). Fig. 1.2 shows the events written above the transitions. Furthermore, we also adopt the representation of software package TCT\(^2\) that controllable events are denoted by odd numbers, while uncontrollable events are denoted as even numbers.

Finally we show the notation for the marked states in a generator. We have two ways to denote the marked states. The first is to have one arrow pointing away from the state [42]. The other is by two concentric circles, which is used by TCT. The states of \( X \) are denoted by natural numbers. The initial state is denoted as 0. Furthermore we use \( G(L) \) to refer to the generator of a regular language \( L \).

\(^2\)http://www.control.utoronto.ca/cgi-bin/dltctdos.cgi
Chapter 2

Problem Formulation

The reduction and design of hierarchical controllers are intertwined problems for hierarchical supervisory control. We incorporate the two problems into one optimization problem, which aims to find small abstractions in hierarchical supervisory control for maximally permissive and nonblocking control. The problem is solved in two scenarios. If the monolithic controller is known then the high-level controller is defined based on the monolithic controller. If the monolithic controller is unknown, then the high-level controller is synthesized based on the high-level plant and specification.

Section 2.1 reviews the literature on hierarchical supervisory control and reduction of controllers. Section 2.2 introduces various hierarchical architectures. Section 2.3 formulates the optimization problem of building abstractions for maximally permissive and nonblocking control. Finally, Section 2.4 summarizes the content of this chapter.
2.1 Literature Review of Hierarchical Supervisory Control

Hierarchical supervisory control divides the system into groups of subsystems, where a controller is designed to supervise each group of subsystems. Furthermore, the abstraction of the controller is used to represent the subsystem for higher level supervisory control. Therefore, a large-scale supervisory control can be accomplished by a batch of small hierarchical controllers. We first review the literature on hierarchical controller design, and then introduce the work on hierarchical controller reduction.

While early research focused on vertical system structure [46, 41, 8], most recent research has shifted attention to horizontal system structure [9, 10, 27, 28, 31, 43, 20, 19, 22, 15, 29], which doesn’t require calculating the pair of global plant and specification, thus avoids the modeling in the global state space which can lead to the combinatorial explosion of states. Early work [19, 22, 15] was designed for hierarchies with only two levels, whereas recent techniques [13, 30, 25, 12, 16, 29] can be applied for multiple levels.

In the horizontal control framework, the approaches of [9, 10, 27, 28, 31, 43, 20] focus on a special system as modular structure, and assume that the subsystems are nonblocking. On the other hand, more recent research [12, 29, 30] introduces a coordinator to the system for nonblocking control.

In respect of the controller abstraction techniques, the methods in [36, 37, 14, 18, 32] utilize nondeterministic automata whereas [17, 22, 21, 12, 29, 30] employ natural projection. Since projection-based approaches are based on languages instead of automata, they are simpler to analyze, and can be applied beyond the conventional automaton model.

In projection-based abstraction methods, the approaches of [17, 22, 21] use a concept of interface, which defines the interactions between levels,
for abstraction. Although those methods function well in achieving the controllability and nonblocking properties, they don’t seek maximal permissiveness so the controllers designed may be strictly weaker than the monolithic controller. By contrast, the methods of [12, 29, 30] employ a type of projection called natural observer for the nonblocking control. Furthermore, with additional conditions they also achieve maximally permissive control. Specifically, [12] adopts output control consistency (occ), while [29] uses a weaker condition called local control consistency (lcc).

There have been various results on controller abstractions [38, 39, 24, 40, 3, 4, 35, 45, 44]. They take different approaches towards building simpler abstractions of controllers. The supervisor reduction (SupReduce) of [38, 39, 24, 40] simplifies the controllers by minimizing the generator of a controller. In words, states of the controllers are merged in such a way that they can be recovered given the automaton of the plant. On the other hand, localization [3, 4, 35, 45, 44] aims to decompose a controller to multiple small ones. In words, if the plant consists of $n$ components, then the controller can be decomposed into $n$ controllers, where the controllers can recover the original controller collaboratively. However, SupReduce focuses on minimizing the generator of controllers by merging equivalent states in a so-called control congruence, but not on simplifying the language of the controller. Localization may generate multiple simpler controllers, but it is time consuming to compute them. Furthermore it may be difficult to see the global information as it is scattered among the multiple controllers.

We incorporate the hierarchical controller reduction (comprehension) problem and controller design problem into the problem of building small abstractions for hierarchical supervisory control. In words, if hierarchical controllers are known, then the abstractions are the image of those controllers under projection. Thus the controller reduction problem is to find a minimal number of observable events for the projection. If the hierarchical controllers are unknown, then the controller design problem is to find
small abstractions of the plant and specification for maximally permissive and nonblocking control. We provide a complete solution for the controller reduction problem. A suboptimal solution is proposed for the controller design problem. Although it is not optimal, the conditions are weaker than the conditions given by recent work [12, 29].

2.2 Hierarchical Supervisory Control Architectures

In this section, we introduce hierarchical architecture, decentralized architecture and vertical architecture. The definitions are given below.

Let \((M_1, E_1), \ldots, (M_n, E_n)\) be the local systems, such that \(M_i \subseteq \Sigma_i^*\) is closed and \(E_i \subseteq M_i\). Let \(\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,u}\), where \(\Sigma_{i,u}\) is the local uncontrollable event set and \(\Sigma_{i,c}\) is the controllable event set, \(\Sigma_u = \bigcup_{i=1}^n \Sigma_{i,u}\) is the global uncontrollable events and \(\Sigma_c = \bigcup_{i=1}^n \Sigma_{i,c}\) is the global controllable events. \(\Sigma_{i,\cap} = \bigcup_{j \neq i} \Sigma_j \cap \Sigma_i\) is the local shared event set and \(\Sigma_{\cap} = \bigcup_{i=1}^n \Sigma_{i,\cap}\) is the global shared event set. The local controllers are designed as \(K_i := supC(M_i, E_i, \Sigma_{i,u})\), \(i = 1, \ldots, n\).

The monolithic controller is designed as \(K_{sup} := supC(\|M_i\|_{i=1}^n, \|E_i\|_{i=1}^n, \Sigma_u)\). Although the monolithic controller guarantees maximally permissive and nonblocking control, it demands the greatest computational effort (time and memory). The following definition can bridge the hierarchical supervisory control and monolithic supervisory control.

**Definition 2.** ([27]) Let \(P_{i,j} : (\Sigma_i \cup \Sigma_j)^* \rightarrow \Sigma_j^*\), \(P_{j,i} : (\Sigma_i \cup \Sigma_j)^* \rightarrow \Sigma_i^*\). If

\[
M_i(\Sigma_i \cap \Sigma_{j,u}) \cap P_{j,i}(P_{i,j}^{-1}(M_j)) \subseteq M_i \\
M_j(\Sigma_j \cap \Sigma_{i,u}) \cap P_{i,j}(P_{j,i}^{-1}(M_i)) \subseteq M_j
\]

then \(M_i, M_j\) are called mutually controllable.
Chapter 2. Problem Formulation

Namely if a local $M_j$ is projected to $\Sigma_i^*$ to play the role of plant, and $M_i$ is controllable w.r.t. the $P_j,i(P_{i,j}^{-1}(M_j))$, furthermore if the statement is true vice versa, then $M_i, M_j$ are mutually controllable. In summary, mutual controllability measures the coupling effects of two plants.

The following proposition provides a feasible basis that it is feasible to build supervisory control based on local controllers, in order that various hierarchical supervisory control techniques can be applied.

**Proposition 1.** ([27]) If $M_1, \ldots, M_n$ are pairwise mutually controllable, then

$$supC(||i=1^n M_i, ||i=1^n E_i, \Sigma_u) = supC(||i=1^n K_i, ||i=1^n K_i, \Sigma_u)$$

Henceforth, we build the hierarchical supervisory control based on local controllers. The global plant and specification are defined as $M := ||i=1^n M_i, E := || K_i$. The monolithic controller is $K := supC(M, E, \Sigma_u)$, which is identical to $K_{sup}$ given that local plants are pairwise mutually controllable.

Although it may be computationally prohibitive to design the monolithic controller $K$, it achieves maximally permissive and nonblocking control. Therefore it provides a universal benchmark for the hierarchical controllers to accomplish maximally permissive and nonblocking control.

### 2.2.1 Hierarchical Supervisory Control

A typical hierarchical supervisory control example is shown in Fig. 2.1, where 3 hierarchical controllers $K_{sub_1}, K_{sub_2}, K_{sub_3}$ are designed to supervise 4 local controllers $K_1, K_2, K_3, K_4$.

\(^{1}\)The conventional global plant is $||i=1^n M_i$, but we abuse notation for consistency in calling ‘plant’ the system to be controlled.
Figure 2.1: Hierarchical Control

The hierarchical supervisory control procedure consists of two steps:

Hierarchical Architecture
1. Group the local subsystems to form a hierarchy
2. Design controllers for each level of subsystems

The hierarchy of the system is a hierarchical grouping of local controllers. For example, in the first level, the 4 local controllers in Fig. 2.1 are grouped as (1, 2), (3, 4). In the second level, all the abstractions of the subsystems are grouped together. Some results [1], [5], [33] have been reported for hierarchy design, where subsystems are grouped according to graph theory analysis.

Since the monolithic controller achieves maximal permissiveness, thus if hierarchical controllers can collaboratively be equivalent to the monolithic controller, then they collaboratively achieve maximal permissiveness. We formulate the statement in the following equation. Furthermore, non-blocking control can be achieved if the hierarchical controllers and local controllers are synchronously nonconflicting. We formulate the statements
into the following equations.

\[
\begin{align*}
\langle \| n_{i=1}^{n} \bar{K}_{i} \| \rangle \parallel \langle \| m_{i=1}^{m} K_{sub_{i}} \| \rangle &= K \quad \text{(Maximally Permissive)} \\
\langle \| n_{i=1}^{n} \bar{K}_{i} \| \rangle \parallel \langle \| m_{i=1}^{m} K_{sub_{i}} \| \rangle &= \bar{K} \quad \text{(Nonblocking)}
\end{align*}
\]

For each level of the subsystems, the controller \( K_{sub_{i}} \) can be designed with methods of decentralized or vertical supervisory control approaches. We will introduce them in the next section.

### 2.2.2 Vertical and Decentralized Supervisory Control

In this section, we introduce the vertical supervisory control and decentralized supervisory control, which can solve a two-level hierarchical supervisory control problem.

![Vertical and Decentralized Supervisory Control](image)  

**Figure 2.2:** Vertical and Decentralized Supervisory Control

The vertical control procedure consists of 4 steps.

**Vertical Architecture**
1. Generate the plant \( M = \|_{i=1}^{n} K_i^2 \) and specification \( E = \|_{i=1}^{n} K_i \).

2. Find a suitable global projection \( P_o : \Sigma^* \to \Sigma_o^* \), where \( \Sigma_o \subseteq \Sigma \).

3. Generate the abstracted plant \( M^{hi} := P_o(\|_{i=1}^{n} K_i) \) and specification \( E^{hi} := P_o(E) \).

4. Design the controller \( K^{hi} = supC(M^{hi}, E^{hi}, \Sigma_u \cap \Sigma_o) \).

The high-level controller \( K^{hi} \) is required to achieve the maximally permissive and nonblocking control such that

\[
M \| K^{hi} = K \quad \text{(Maximally Permissive)}
\]
\[
M \| \overline{K^{hi}} = \overline{K} \quad \text{(Nonblocking)}
\]

The decentralized control procedure consists of 4 steps.

Decentralized Architecture

1. Design suitable local projections \( Q_1, \ldots, Q_n \), where \( Q_i : \Sigma^* \to \Sigma_{i,o}^* \), \( \Sigma_{i,o} \subseteq \Sigma_i \).

2. Generate local abstractions \( K^{hi}_1, \ldots, K^{hi}_n \) according to \( K^{hi}_i = Q_i(K_i) \).

3. Synthesize the abstracted plant \( M^{hi'} := \|_{i=1}^{n} \overline{K^{hi}_i} \) and specification \( E^{hi'} := \|_{i=1}^{n} \overline{K^{hi}_i} \).

4. Design the controller \( K^{hi'} = supC(M^{hi'}, E^{hi'}, \Sigma_u \cap \Sigma_o) \), where \( \Sigma_o = \bigcup_{i=1}^{n} \Sigma_{i,o} \).

The high-level controller \( K^{hi'} \) is required to achieve the maximally permissive and nonblocking control such that

---

\(^*\)Conventional plant is \( \|_{i=1}^{n} M_i \)
Decentralized control builds the high-level plant and specification with local abstractions $K_{hi}^1, \ldots, K_{hi}^n$ which are small, while vertical control needs the global plant and specification $M, E$. In this regard, decentralized control corresponds to “act locally”, and vertical control corresponds to “think globally”. In this thesis, we will “think globally” to focus on the vertical architecture.

### 2.3 Problem Formulation

For the reader’s convenience, we repeat the definitions of Section 2.2 here.

Let $(M_1, E_1), \ldots, (M_n, E_n)$ be the local systems, such that $M_i \subseteq \Sigma_i^*$ is closed and $E_i \subseteq M_i$. Let $\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,u}$, where $\Sigma_{i,u}$ is the local uncontrollable event set and $\Sigma_{i,c}$ is the controllable event set, $\Sigma_u = \bigcup_{i=1}^n \Sigma_{i,u}$ is the global uncontrollable events and $\Sigma_c = \bigcup_{i=1}^n \Sigma_{i,c}$ is the global controllable event set. The set $\Sigma_{i,\cap} = \bigcup_{j \neq i} \Sigma_j \cap \Sigma_i$ is the local shared event set and $\Sigma_{\cap} = \bigcup_{i=1}^n \Sigma_{i,\cap}$ is the global shared event set. The local controllers are designed as $K_i := supC(M_i, E_i; \Sigma_{i,u}), i = 1, \ldots, n$. The global plant and specification are defined as $M := \|_{i=1}^n K_i$, $E := \| K_i$. The monolithic controller is $K := supC(M, E; \Sigma_u)$.

We have to find a natural projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$, where $\Sigma_o \subseteq \Sigma$. It is equivalent to find the observable event set $\Sigma_o$ for $P_o$. The high-level plant and specification are generated as $M^{hi} = P_o(M), E^{hi} = P_o(E)$. The high-level controller $K^{hi} \subseteq \Sigma_o^*$ is the desired controller designed by two
scenarios:

1. Monolithic controller $K$ is known, then $K^{hi} = P_o(K)$;

2. Monolithic controller $K$ is not known, then $K^{hi} = \text{supC}(M^{hi}, E^{hi}, \Sigma_o \cap \Sigma_u)$.

The two scenarios correspond to the controller reduction problem and controller design problem respectively. The controller reduction problem is to find smaller abstractions of controllers for understanding. The control design problem is to find small abstractions for the plant and specification for high-level supervisory control. In this thesis we minimize the abstractions of controllers by projecting out as many unnecessary events as possible, so the two problems are to find a minimal observable event set.

The optimization problem is formulated below.
minimize $|\Sigma_o|$ subject to
\[
\|\prod_{i=1}^{n} K_i \| K^{hi} = K \quad \text{(Maximally Permissive)}
\]
\[
\|\prod_{i=1}^{n} K_i \| K^{hi} = \|\prod_{i=1}^{n} K_i \| K^{hi} \quad \text{(Nonblocking)}
\]

We provide a full solution to Problem (2.1) in the first scenario, and a suboptimal solution in the second scenario.

### 2.4 Chapter Summary and Conclusions

Hierarchical supervisory control and controller abstraction are two major challenges in supervisory control of DES. In this chapter we incorporate the two problems into one optimization problem, where the task is to find a minimally observable event set for maximal permissive and nonblocking control.

The controller abstraction problem is when the monolithic controller is known, then the high-level controller(abstraction) is built based on the monolithic one. The controller design problem is when the monolithic controller is unknown, then the high-level controller is designed based on the high-level plant and specification. Therefore the two problems are closely related in that the critical part is to find the observable event set to build abstractions of controller/plant and specification.

We will propose solutions to both problems in Chapter 3, where an optimal solution is given for the controller reduction problem, and a suboptimal solution is given for the controller design problem.
Chapter 3

Building Efficient Abstractions

Problem (2.1) is solved in two scenarios, 1. Monolithic controller $K$ is known, 2. Monolithic controller $K$ is not known. A full solution to the first scenario is proposed, which paves the way to understand $K$. In the second scenario, we provide sufficient conditions for $\Sigma_o$ to be a feasible solution to Problem (2.1).

Section 3.1 solves Problem (2.1) when $K$ is known. Section 3.2 obtains feasible solutions to Problem (2.1) when $K$ is not known.

3.1 Monolithic Controller Is Known

In the supervisory control problem, constructing a simpler controller is a big challenge since the size of a controller may be too large for human comprehension. Therefore devising a small high-level controller, which works with all local controllers to agree with the monolithic controller is important. The problem is expressed in Problem (2.1) as $K^{hi} = P_o(K)$, where the high-level controller is designed based on $K$. 
3.1.1 Theoretical Analysis

The nature of the constraints of Problem (2.1) can be characterized by normality, which is defined as

**Definition 3. (Normality [23])** Given a closed language $M \subseteq \Sigma^*$, and a natural projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$, where $\Sigma_o \subseteq \Sigma$. A language $N \subseteq M$ is called $(M, P_o)$-normal if

$$P_o^{-1}(P_o(N)) \cap M = N$$

Normality says that the abstraction of $N$ under $P_o$ as $P_o(N)$ can recover $N$ with the help of $M$. It can be written in the following equation.

$$M \parallel P_o(N) = N$$

We further discuss the property of normality in the following proposition.

**Proposition 2.** Let $M \subseteq \Sigma^*$ be a closed language. Let $P_o : \Sigma^* \rightarrow \Sigma_o^*$ be a natural projection, where $\Sigma_o \subseteq \Sigma$. A language $N \subseteq M$ is $(M, P_o)$-normal iff

$$(\forall s \in M) P_o(s) \in P_o(N) \Rightarrow s \in N$$

The critical feature of the definition is that membership of the string $s \in N$, $s' \in M - N$ are preserved by projection $P_o$. Namely that the projections of strings in $N$ are in $P_o(N)$, and the projections of strings in $M - N$ are in $P_o(M) - P_o(N)$.

Fig. 3.1 shows that the membership of strings that $s \in N$, $s' \in M - N$ are preserved by projection $P_o$. Namely if $s \in N$, then $P_o(s) \in P_o(N)$, if $s' \in M - N$, then $P_o(s') \in P_o(M) - P_o(N)$.

The next lemma says that the first constraint of Problem (2.1) as (1)
Maximally Permissive control is equivalent to the requirement that $K$ be $(M, P_o)$-normal.

**Lemma 1.** The Maximally Permissive constraint in Problem (2.1) is satisfied iff $K$ is $(M, P_o)$-normal.

Given that $\Sigma_o$ satisfies (1) the maximally permissive constraint, the next lemma says that it satisfies (2) the nonblocking constraint if and only if $\overline{K}$ is $(M, P_o)$-normal.

**Lemma 2.** Suppose the Maximally Permissive constraint in problem (2.1) is satisfied. The Nonblocking constraint is satisfied iff $\overline{K}$ is $(M, P_o)$-normal.

Notice that the Lemma 1 is a strong result in that we don’t have any other premised conditions, while Lemma 2 characterizes the nonblocking control when the maximally permissive constraint is satisfied. Therefore Lemma 2 is conditionally necessary and sufficient for the nonblocking constraint, e.g. there might be other conditions for nonblocking con-
straint without the maximally permissive constraint, such as the trivial case $\Sigma_o = \emptyset$.

Combining the two lemmas, we can provide a necessary and sufficient condition for the (1) Maximally Permissive, (2) Nonblocking constraints simultaneously.

**Theorem 1.** The two constraints (1) Maximally Permissive, (2) Nonblocking control of Problem (2.1) are satisfied iff

1. $K$ is $(M, P_o)$-normal;

2. $\overline{K}$ is $(M, P_o)$-normal.

This is a strong result in that it doesn’t have any premised conditions. Therefore it universally characterizes the nature of the constraints of Problem (2.1). In this regard we can transform the Problem (2.1) to an equivalent problem, e.g. the solution sets are the same.

\[
\begin{align*}
\text{minimize } &|\Sigma_o| \text{ subject to} \\
K & \text{ is } (M, P_o)\text{-normal} \\
\overline{K} & \text{ is } (M, P_o)\text{-normal}
\end{align*}
\]

Due to Theorem 1, Problem (3.1) is equivalent to Problem 2.1, as $\Sigma_o$ is feasible in Problem (3.1) if and only if it is feasible in Problem (2.1), and the objective functions in both problems are to minimize the size of $\Sigma_o$. 
3.1.2 Computation Procedure and Example

Although we can verify whether or not $\Sigma_o$ satisfies that $K$ is $(M, P_o)$-normal, it is still an open problem to find a minimal $\Sigma_o$. A brute force solution is to test all subsets of $\Sigma_o$, but we prefer to devise a heuristic procedure. The algorithm is shown below.

**Algorithm 1:** Calculate $\Sigma_o$

**Data:** Generators $G(M), G(E)$

**Result:** feasible $\Sigma_o$ for Problem (3.1)

1. Construct a generator $G$ such that $L_m(G) = M \parallel E, L(G) = \overline{M} \parallel \overline{E}$. $X_m$ is the marked states set;

2. Initialize $\Sigma_o = \emptyset$;

3. Find all events pointing from the marked states to the unmarked states of $G$, add them to $\Sigma_o$;

4. Find all events pointing from the unmarked states to the marked states of $G$, add them to $\Sigma_o$;

5. Find all events pointing from a coreachable state to an uncoreachable state, add them to $\Sigma_o$;

6. If $K, \overline{K}$ are $(M, P_o)$-normal, go to step 8;

7. Select a $\sigma \in \Sigma - \Sigma_o$, add it to $\Sigma_o$, then go to step 6;

8. Return $\Sigma_o$.

Table 3.1: Algorithm 1
Chapter 3. Building Efficient Abstractions

We need to point out that Algorithm 1 is only a heuristic approach to Problem 3.1, and that it may not find a minimal solution for Problem (3.1).

We use the Transfer-Line system [42] to show how to find a smallest $\Sigma_o$. All computations are done with TCT software\footnote{http://www.control.utoronto.ca/DES/}. A regular language $L$ is represented as generator $G(L)$ for the reader’s convenience.

![Figure 3.2: Transfer-Line System](image)

There are three machines $G_1, G_2, G_3$ and two buffers $B_1, B_2$ in the Transfer-Line system. Their generators are shown in Fig. 3.3.

![Figure 3.3: Machines and Buffers](image)

According to Section 2.3, we first prepare the local plants and local specifications.

\[
M_1 := L(G_1) \parallel L(G_2) \parallel L(G_3) \\
E_1 := L_m(G_1) \parallel L_m(G_2) \parallel L_m(G_3) \parallel L_m(B_1) \\
M_2 := L(G_2) \parallel L(G_3) \\
E_2 := L_m(G_2) \parallel L_m(G_3) \parallel L_m(B_2)
\]
The DES are shown in Fig. 3.4

![Diagram](image)

Figure 3.4: Local Plants and Local Specifications

The event sets are:

\[ \Sigma_1 = \{1, 2, 3, 4, 5, 6, 8\} \]
\[ \Sigma_2 = \{3, 4, 5, 6, 8\} \]
\[ \Sigma = \{1, 2, 3, 4, 5, 6, 8\} \]
\[ \Sigma_{\cap} = \Sigma_1 \cap \Sigma_2 = \{3, 4, 5, 6, 8\} \]
\[ \Sigma_{1,u} = \{2, 4, 6, 8\} \]
\[ \Sigma_{2,u} = \{4, 6, 8\} \]
\[ \Sigma_u = \{2, 4, 6, 8\} \]
The local controllers are

\[ K_1 := supC(M_1, E_1, \Sigma_{1,u}) \]
\[ K_2 := supC(M_2, E_2, \Sigma_{2,u}) \]

The DES are shown in Fig. 3.5.

![Diagram of Local Controllers](image)

**Figure 3.5: Local Controllers**

The global plant and specification are

\[ M := \overline{K_1} \parallel \overline{K_2} \]
\[ E := K_1 \parallel K_2 \]

The DES is shown in Fig. 3.6.

The controller \( K \) is designed w.r.t. \((M, E)\) as

\[ K = supC(M, E, \Sigma_u) \]

which is shown in Fig. 3.7.

With \( K \) at hand, we can now check whether a given \( \Sigma_o \) satisfies the constraints of Problem (3.1). Since the set \( \Sigma \) contains only 7 events, we
can enumerate the choices of $\Sigma_o$ to satisfy the two constraints:

1. $K$ is $(M, P_o)$-normal;
2. $\overline{K}$ is $(M, P_o)$-normal.

Algorithm 1 finds \{1\} at step 3, and \{6\} at step 4. Event set \{1, 6\} is already a feasible solution to Problem (3.1). Indeed in the Transfer-Line system, \{1, 6\} is the minimal solution to Problem (3.1).

The high-level controller is designed as

$$K^{hi} = P_o(K)$$

which is shown in Fig. 3.8.

Fig.3.7 and Fig. 3.8 express the critical point of the controller, that
event 1 can only occur once before the occurrence of event 6, or else blocking will happen as is shown in Fig. 3.6, considering the string \( s = 1231 \). However, our abstraction \( K^{hi} \) is simpler than \( K \) for the reader to understand, since \( K \) contains too many redundant events while \( K^{hi} \) only preserves the important ones.

3.2 Monolithic Controller Is Unknown

In various hierarchical control approaches, the major task is to build a high-level controller \( K^{hi} \), which agrees with monolithic controller \( K \) in the sense that (1) Maximally Permissive, (2) Nonblocking control are achieved. Therefore it is important to analyze the properties of the observable events, without information on \( K \). It is formulated in Problem (2.1) that the high-level controller is designed as \( K^{hi} := supC(M^{hi}, E^{hi}, \Sigma_u \cap \Sigma_o) \). Furthermore, it is desired to have conditions without information on \( K \) such that the results can be informative to analyze decentralized control.

A feasible solution to Problem (2.1) is proposed based on boundary analysis of languages. Although the solution may not be a minimal one, it is theoretically no larger than in recent work [12],[29].

We first analyze the properties of Problem (2.1) in Section 3.2.1. A feasible solution to the special case that \( E \) is closed is given in Section 3.2.2. We generalize the solution in Section 3.2.3. In Section 3.2.5 we give a procedure to compute the observable events, and illustrate it in two examples.
3.2.1 Problem Analysis

The problem we aim to solve in this section is different from that of the previous section in that now $K^{hi}$ is dependent on the high-level plant and specification, while $K^{hi}$ is designed based on $K$ in Section 3.1. Therefore not only does the projection $P_o$ have to preserve information on $K$, but it also has to preserve information on $E$.

In this regard, it is very difficult to propose a universal minimal solution for Problem (2.1). Henceforth, we take a trade-off that focuses on finding a feasible solution to Problem (2.1).

The following lemma reveals an important relationship between normality and controllability.

**Lemma 3.** Suppose $\bar{L}$ is $(M, P_o)$-normal. $L$ is controllable w.r.t. $M$ iff $P_o(L)$ is controllable w.r.t. $P_o(M)$.

With the help of Lemma 3, we now introduce a theorem which helps to analyze the problem in the rest of the thesis.

**Theorem 2.** Suppose $E$ is $(M, P_o)$-normal. An event set $\Sigma_o$ satisfies the constraints of Problem 2.1 iff

1. $K$ is $(M, P_o)$-normal,

2. $\bar{K}$ is $(M, P_o)$-normal,

With Theorem 2, we now modify the problem to the following form:
minimize $|\Sigma_o|$ subject to
\[ K \text{ is } (M, P_o)\text{-normal} \]
\[ \overline{K} \text{ is } (M, P_o)\text{-normal} \]
\[ E \text{ is } (M, P_o)\text{-normal} \]

Problem (2.1) and Problem (3.2) are not equivalent. In words, if $\Sigma_o$ is feasible for Problem (3.2), then it is feasible for Problem 2.1, while the reverse statement is not true. However, since we will not seek the minimal solution to Problem (2.1) but only a feasible solution, thus solving Problem (3.2) is a sufficient condition.

Since $K$ is unknown, the constraint that $K$ is $(M, P_o)$-normal is unverifiable, hence we have to find alternative conditions to substitute for the constraints based on $K$. Feasible solutions for Problem (3.2) without information on $K$ are proposed in the following sections.

A feasible solution for Problem (3.2) is proposed given that $E$ is closed in Section 3.2.2. The general solution is proposed in Section 3.2.3.

### 3.2.2 Closed Specification Solution

In this subsection we provide a method to find a feasible $\Sigma_o$ without building $K$ given that $E$ is closed. Throughout this section, we modify the definitions in Section 2.3 in that the specification is defined as $E' = \overline{E} = \bigcap_{i=1}^{n} K_i$, the monolithic controller is modified as $K' = supC(M, E', \Sigma_o)$. The other definitions are not changed. Thus all conclusions of Section 3.1 can be applied here without modification. Finally we will write $E, K$ instead of $E', K'$ for convenience.
The reason that we are interested in a closed specification $E$ is because it significantly reduces the complexity of problem (3.2).

**Proposition 3.** Let $M \subseteq \Sigma^*$ be closed, let $E \subseteq M$, let $\Sigma = \Sigma_c \cup \Sigma_u$, where $\Sigma_u, \Sigma_c$ are the uncontrollable and controllable events respectively. If $E$ is closed, then $K = \sup C(M, E, \Sigma_u)$ is closed.

The three constraints in Problem (3.2) require that $K$ and $\overline{K}$ be $(M, P_o)$-normal. With Proposition 3 the Problem (3.2) can thus be transformed to another equivalent one, i.e. its solution set equals that of Problem (3.2).

\[
\begin{align*}
\text{minimize } |\Sigma_o| & \quad \text{subject to} \\
K & \text{ is } (M, P_o)\text{-normal} \\
E & \text{ is } (M, P_o)\text{-normal}
\end{align*}
\] (3.3)

We first analyze the normality thoroughly. In this regard, a definition to characterize the boundary of a language $L$ is given below.

**Definition 4.** Let a closed language $M \subseteq \Sigma^*$, let a closed language $L \subseteq M$. A pair $(s, \sigma)$, where $s \in M, \sigma \in \Sigma$ is called a piece of the boundary of $L$ w.r.t. $M$ if $s \in L \land s\sigma \in M - L$.

**Definition 5.** Let a closed language $M \subseteq \Sigma^*$, let a closed language $L \subseteq M$. An event $\sigma$ is called an event of boundary of $L$ w.r.t. $M$ if

\[(\exists s \in L)s\sigma \in M - L\]

Fig. 3.9 shows two pairs $(s_1, \sigma_c), (s_2, \sigma_u)$ as pieces of the boundary of $L$ w.r.t. $M$. Thus $\sigma_c, \sigma_u$ are events of the boundary of $L$. We now show that the pieces of the boundary characterize the normality of $L$ w.r.t. $(M, P_o)$.

**Proposition 4.** Let $M \subseteq \Sigma^*$ be closed, let $L \subseteq M$, let $P_o : \Sigma \to \Sigma_o^*$ be a
Chapter 3. Building Efficient Abstractions

Figure 3.9: Boundary Strings and Events of $L$

Proposition 4 says that $L$ is $(M, P_o)$-normal if and only if $P_o$ preserves the boundary information of $L$ completely. In the next proposition, we characterize the relation between events of the boundary of $L$ and its controllability.

**Proposition 5.** Let $M \subseteq \Sigma^*$ be closed, let $L \subseteq M$, let $\Sigma = \Sigma_u \cup \Sigma_c$ where $\Sigma_u, \Sigma_c$ are the uncontrollable and controllable events. Let $\Sigma_b$ be the events of the boundary of $\overline{L}$ w.r.t. $M$. Then $L$ is controllable w.r.t. $(M, \Sigma_u)$ iff

$$\Sigma_b \subseteq \Sigma_c$$

Proposition 5 says that the controllability of a language $L$ is completely characterized by its events of the boundary w.r.t. $M$. We can thus summarize the properties of $K$ as the largest sublanguage of $E$ such that its events of the boundary w.r.t. $M$ are controllable.

Propositions 3 and 4 connect normality of $L$ and controllability of $L$ with the pieces of the boundary of $L$ w.r.t. $M$. Normality is about the...
whole pieces of the boundary of $L$, while controllability is about the events of the boundary of $L$.

\[
\begin{tikzpicture}
  \node (L) at (0,0) {$L$};
  \node (Lbar) at (2,0) {$\bar{L}$};
  \node (proj) at (1,-1) {$P_o$};
  \node (projL) at (0,-2) {$P_o(L)$};
  \node (projLbar) at (2,-2) {$\bar{P}_o(L)$};
  \node (projLbarL) at (0,-3) {$P_o(L)$};
  \node (projLbarLbar) at (2,-3) {$\bar{P}_o(L)$};
  \draw[->] (L) to node[anchor=west] {$u$} (proj);
  \draw[->] (Lbar) to node[anchor=west] {$s$} (proj);
  \draw[->] (proj) to node[anchor=south] {$P_o(L)$} (projL);
  \draw[->] (proj) to node[anchor=south] {$\bar{P}_o(L)$} (projLbar);
  \draw[->] (projL) to (projLbarL);
  \draw[->] (projLbar) to (projLbarLbar);
  \draw[->] (projLbarL) to (projLbarLbar);
\end{tikzpicture}
\]

Figure 3.10: Natural Observer

To find a feasible $\Sigma_o$ without building $K$, we need a tool called natural observer, which is defined below.

**Definition 6. (Natural Observer [41])** Let $L \subseteq \Sigma^*$, let $\Sigma_o \subseteq \Sigma$. A natural projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$ is called an $L$-observer if

\[
(\forall s \in \bar{L}), (\forall t \in \Sigma_o^*), P_o(s)t \in P_o(L) \Rightarrow (\exists u \in \Sigma^*), P_o(u) = t \land su \in L
\]

Natural observer says that if the image of $s \in \bar{L}$ under $P_o$ can be extended to $P_o(L)$ by a string $t \in \Sigma_o^*$, then there exists a string $u \in \Sigma^*$ to extend $s$ into $L$, and $t$ is the image of $u$ under $P_o$. One illustration of natural observer is shown in Fig. 3.10, where the projection $P_o$ is an $L$-observer. A string $s \in \bar{L}$ can be extended to $L$ by $u$ if its projection $P_o(s)$ is extended to $P_o(L)$ by $t$, where $P_o(u) = t$.

Now we build a feasible solution to Problem (3.3).

**Theorem 3.** Suppose $P_o$ is an $M$-observer, $E$ is $(M, P_o)$-normal, and $\Sigma_{b,K}$ are the events of the boundary of $\bar{K}$ w.r.t. $M$. The constraints of Problem
(3.3) are satisfied iff
\[ \Sigma_{b,K} \subseteq \Sigma_o \]

Theorem 3 provides a weaker condition about \( K \) in that we don’t need to know all pieces of the boundary of \( K \), but only the events of the boundary of \( K \). Furthermore, it says that the missing boundary information of \( K \) can be recovered from the boundary information of \( E \), by augmenting \( P_o \) to be \( M \)-observer. Nevertheless it still requires \( \Sigma_{b,K} \), which may not be known until \( K \) is built.

Fortunately, when \( E \) is closed, there is a strong relation between the events of the boundary of \( E \) and the events of the boundary of \( K \). We show the connection in Fig. 3.11

![Figure 3.11: Events of Boundaries of \( E \), \( K \) w.r.t. \( M \)](image)

There are three events \( \sigma_u \in \Sigma_u, \sigma_c, \sigma'_c \in \Sigma_c \), such that \( \sigma_c, \sigma'_c \) are the events of the boundary of \( K \) and \( \sigma'_c, \sigma_u \) are the events of the boundary of \( E \). Therefore \( \sigma'_c \) is an event of both boundaries of \( E \) and \( K \), while \( \sigma_c \) is exclusively an event of the boundary of \( K \). Due to definition 4, there exists \( s \in K \) such that \( s\sigma_c \in M - K \). Since \( \sigma_c \) is not an event of the
boundary of \( E \), thus \( s\sigma_c \in E - K \). We see in Fig. 3.11 that there is an uncontrollable string \( u \in \Sigma_u^* \) such that \( s\sigma_c u \in E \land s\sigma_c u \sigma_u \in M - E \). It is not a coincidence, where the deep connection is implied by Propositions 4 and 5. The previous description is formalized in the following proposition.

**Proposition 6.** Let \( \Sigma_{b,K} \) be the set of all the events of the boundary of \( K \) w.r.t. \( M \), and \( \Sigma_{b,E} \) be the set of all the events of the boundary of \( E \) w.r.t. \( M \). Then

\[
\forall \sigma_c \in \Sigma_{b,K} \sigma_c \notin \Sigma_{b,E} \\
\Rightarrow \exists s \in K \exists u \in \Sigma_u^* (\exists \sigma_u \in \Sigma_{b,E} \cap \Sigma_u) s\sigma_c u \in E \land s\sigma_c u \sigma_u \in M - E
\]

Now we summarize the current results. Theorem 3.3 says that when (1) \( E \) is \((M, P_o)\)-normal, (2) \( P_o \) is \( M \)-observer, then \( \Sigma_{b,K} \subseteq \Sigma_o \) is a necessary and sufficient condition for \( \Sigma_o \) to be a feasible solution for Problem (3.3). Proposition 4 says that condition (1) implies \( \Sigma_{b,E} \subseteq \Sigma_o \). Therefore the missing part is now \( \Sigma_{b,K} - \Sigma_{b,E} \), the exclusive events of the boundary of \( K \). We need more tools to find them. Locally consistent control(lcc) is defined below.

**Definition 7.** *(Locally Control Consistent [29])* Let \( L \) be a language over \( \Sigma \), let \( \Sigma_o \subseteq \Sigma \), and \( \Sigma_u \subseteq \Sigma \) be the uncontrollable events. A natural projection \( P_o \) is \( L \)-lcc if

\[
(\forall s \in L)(\forall \sigma_u \in \Sigma_u) P_o(s) \sigma_u \in P_o(L) \\
\Rightarrow (\exists u \in (\Sigma_u - \Sigma_o)^*, su\sigma_u \in L) \lor (\forall u \in (\Sigma - \Sigma_o)^*, su\sigma_u \in L)
\]

The lcc definition says that if the image of \( s \in L \) under \( P_o \) stays in \( P_o(L) \) with an extension of a controllable observable event \( \sigma_u \), then either there is an uncontrollable unobservable string between \( s \) and \( \sigma_u \), or there is no unobservable string at all.

The next lemma says that the observer and normality combines with
Lemma 4. Suppose $P_o$ is $M$-observer, $E$ is $(M, P_o)$-normal, and $\Sigma_{b,K}$ are the events of the boundary of $K$ w.r.t. $M$. If $P_o$ is $M$-lcc, then

$$\Sigma_{b,K} \subseteq \Sigma_o$$

Finally we can provide a solution to find a feasible $\Sigma_o$ for Problem (3.3) without any dependencies on $K$. The following theorem is the major result of this section.

Theorem 4. If $P_o$ satisfies following conditions

1. $E$ is $(M, P_o)$-normal;
2. $P_o$ is $M$-observer;
3. $P_o$ is $M$-lcc,

then $\Sigma_o$ satisfies the constraint in Problem (3.3).

The conditions of Theorem 4 are all verifiable. Conditions 1,2 can be checked by TCT.

There are less restrictive conditions than Theorem 4 to find a feasible $\Sigma_o$ for Problem (3.3). Recall that the critical property of $\Sigma_o$ to satisfy the constraint of Problem (3.3) is $\Sigma_{b,K} \subseteq \Sigma_o$. With condition (1) $E$ is $(M, P_o)$-normal, we have that $\Sigma_{b,E} \cap \Sigma_{b,K} \subseteq \Sigma_o$. Therefore the missing part is $\Sigma_{b,K} - \Sigma_{b,E}$. Proposition 6 says that there are connections between $\Sigma_{b,K} - \Sigma_{b,E}$ and $\Sigma_{b,E} \cap \Sigma_u$.

Based on those observations, we provide alternative weaker sufficient conditions to Theorem 4. The critical idea is to devise partial natural observer, and partial lcc properties since only $\Sigma_{b,E} \cap \Sigma_u$ is useful to find $\Sigma_{b,K}$,
while the conventional natural observer/lcc conditions are too conservative in the sense that \( P_o \) has to satisfy these conditions for all \( \Sigma_o/\Sigma_o \cap \Sigma_u \).

**Definition 8.** (Partial Natural Observer) Let \( L \subseteq \Sigma^* \). A natural projection \( P_o : \Sigma^* \rightarrow \Sigma_o^* \) is called an \((L, \Sigma'_o)\)-observer, where \( \Sigma'_o \subseteq \Sigma_o \), if

\[
(\forall s \in \overline{L})(\forall \sigma_o \in \Sigma'_o)(\exists t \in \Sigma_o^*)P_o(s)t\sigma_o \in P_o(L)
\]

\[
\Rightarrow (\exists w \in \Sigma^*)P_o(w) = t \land sw\sigma_o \in L
\]

**Definition 9.** (Partial LCC) Let language \( L \subseteq \Sigma^* \). Let \( \Sigma = \Sigma_u \cup \Sigma_c \), where \( \Sigma_u, \Sigma_c \) are the uncontrollable and controllable events, let \( \Sigma'_u \subseteq \Sigma_u \cap \Sigma_o \). \( P_o \) is called \((L, \Sigma'_u)\)-locally control consistent or \((L, \Sigma'_u)\)-lcc if

\[
(\forall s \in \Sigma^*)(\forall u \in (\Sigma_u \cap \Sigma_o)^*)(\forall \sigma_u \in \Sigma'_u)P_o(s)u\sigma_u \in P_o(\overline{L})
\]

\[
\Rightarrow (\exists w \in \Sigma^*)sw\sigma_u \in \overline{L} \land P_o(w) = u \lor (\forall w \in \Sigma^*)sw\sigma_u \in \overline{L} \land P_o(w) = u
\]

The definitions of partial natural observer, partial lcc are similar to the definitions of natural observer, lcc. The major difference is that the partial natural observer/lcc require \( P_o \) to satisfy conditions for a subset of \( \Sigma_o/\Sigma_o \cap \Sigma_u \), while natural observer/lcc require \( P_o \) to satisfy conditions for all \( \Sigma_o/\Sigma_o \cap \Sigma_u \). Now we weaken the conditions of Theorem 4 by employing the partial natural observer, partial lcc.

**Theorem 5.** If \( P_o \) satisfies the following conditions:

1. \( E \) is \((M, P_o)\)-normal;

2. \( P_o \) is an \((M, \Sigma_{b,E} \cap \Sigma_u)\)-observer;

3. \( P_o \) is an \((M, \Sigma_{b,E} \cap \Sigma_u)\)-lcc,

then \( \Sigma_o \) satisfies the constraint of Problem (3.3).

The essential idea of Theorem 5 is as same as Theorem 4, but we trim
the part of conditions which don’t play any role in the proof of Theorem 4.

In contrast to the conditions of Theorem 4, we haven’t developed computational tools for partial natural observer/ partial lcc. However, it is possible to check the conditions in small examples, as we will confirm in Section 3.2.5.

3.2.3 General Specification Solution

In the previous subsection a feasible solution for Problem 3.2 is proposed given that \( E \) is closed. In this subsection a general feasible solution will be devised without that assumption. Similar to Section 3.2.2, we don’t seek to find a minimal solution, but focus on finding some feasible ones, e.g. satisfying the condition of Problem (3.2).

First of all, we simplify Problem 3.2 by the following Proposition.

**Proposition 7.** Let \( M \subseteq \Sigma^* \) be closed. Let \( P_o : \Sigma^* \rightarrow \Sigma_o \) be the natural projection, where \( \Sigma_o \subseteq \Sigma \). Suppose \( P_o \) is an \( M \)-observer. If \( L \subseteq M \) is \((M, P_o)\)-normal, then \( \overline{L} \) is \((M, P_o)\)-normal.

This proposition bridges the normality of a language \( L \) to its closure \( \overline{L} \) using the observer concept. Therefore it bridges the maximally permissive control to nonblocking control.

**Proposition 8.** Suppose \( P_o \) is \( M \)-observer. If \( \Sigma_o \) satisfies the maximally permissive constraint of Problem (3.2), then it satisfies the nonblocking constraint.
We thus transform the Problem 3.2 to the following one:

\[
\begin{align*}
\text{minimize } |\Sigma_o| \text{ subject to } \\
K & \text{ is } (M, P_o)\text{-normal} \\
E & \text{ is } (M, P_o)\text{-normal} \\
P_o & \text{ is } M\text{-observer}
\end{align*}
\] (3.4)

Although Problem (3.4) reduces the reliance on \( K \) of the constraints, it is still required that \( K \) be \((M, P_o)\)-normal. Thus we would like to propose sufficient conditions to satisfy this constraint without condition on \( K \).

Now we summarize the content of Section 3.2.2. Given a specification \( E \subseteq M \), the monolithic controller is designed as \( K' := \sup C(M, \bar{E}, \Sigma_u) \). The high-level controller is \( K'_{hi} := \sup C(M^{hi}, \bar{E}^{hi}, \Sigma_u \cap \Sigma_o) \), where \( M^{hi}, E^{hi} \) are the high-level plant and specifications such that \( M^{hi} := P_o(M), E^{hi} := P_o(E) \). The problem is formulated in Section 3.2.2 as below.

\[
\begin{align*}
\text{minimize } |\Sigma_o| \text{ subject to } \\
\| \|_{i=1}^{n} K_i \| K'_{hi} = K' \\
\| \|_{i=1}^{n} \bar{K}_i \| \bar{K}'_{hi} = \| \|_{i=1}^{n} \bar{K}_i \| \bar{K}'_{hi}
\end{align*}
\] (3.5)

Recall that in the previous section \( E \) was closed. Modified for the present section where \( E \) is general, Theorem 4 says that if

1. \( \bar{E} \) is \((M, P_o)\)-normal,
2. \( P_o \) is \( M\)-observer,
3. \( P_o \) is \( M\)-lcc,

then \( \Sigma_o \) satisfies the constraints of Problem (3.5).
First of all, conditions 2 and 3 are properties of $P_o$ w.r.t. $M$, while condition 1 is about $E$ and $M$. In words, with conditions 2 and 3, $P_o$ preserves the boundary of $K'$ when it preserves the boundary of $\overline{E}$.

**Lemma 5.** If $K' = \text{supC}(M, \overline{E}, \Sigma_u)$, then

$$\text{supC}(M, K' \cap E, \Sigma_u) = K$$

Lemma 5 says that there is a close relationship between $K'$ and $K$. Namely $K' \cap E$ captures the critical information of $E$ for $K$. In this regard, $K'$ helps to trim $E$ to a smaller language which contains all information relevant to $K$. Particularly, if $K' \cap E$ is $(M, P_o)$-normal, then we can inductively apply Theorem 4. The next lemma says that $K' \cap E$ is $(M, P_o)$-normal.

**Lemma 6.** If

1. $E$ is $(M, P_o)$-normal;
2. $P_o$ is an $M$-observer;
3. $P_o$ is an $M$-lcc;

then, $K' \cap E$ and $\overline{K'} \cap \overline{E}$ are $(M, P_o)$-normal.

Therefore Lemma 6 says that we can use Theorem 4 inductively with three conditions. Namely we define a series of languages as $E^1 = E, E^{n+1} = E^n \cap \text{supC}(M, \overline{E^n}, \Sigma_u)$ inductively, then with three conditions (1) $E$ is $(M, P_o)$-normal; (2) $P_o$ is $M$-observer; (3) $P_o$ is $M$-lcc, we have that $E^m$ and $\overline{E^m}$ are $(M, P_o)$-normal, where $m \in \mathcal{N}$. The series satisfies that $E^1 \supseteq E^2 \supseteq \ldots$, and every $E_m, m \in \mathcal{N}$, still preserves the critical information for $K$. The next lemma says that the series will converge to $K$ in finite steps.
Lemma 7. If $E, M$ are regular languages, then $\exists N \in \mathbb{N}$ such that

$$E^N = K$$

Now we wrap up the previous discussion into the major result of this section.

Theorem 6. If

1. $E$ is $(M, P_o)$-normal,
2. $P_o$ is $M$-observer,
3. $P_o$ is $M$-lcc,

then $\Sigma_o$ is feasible for Problem (3.4), and thus Problem (3.2).

The conditions for Theorem 6 are essentially the same as in Theorem 4. The major differences are the conclusions, where Theorem 4 is for the basis case in the inductive proof, while Theorem 6 extends the conclusion to the whole series $E^1, E^2, \ldots$ inductively. The final part is due to Lemma 7 that the series will converge to $K$ in finite steps, thus $K$ is $(M, P_o)$-normal, which satisfies the constraint of Problem (3.4).

Similar to Section 3.2.2, we can weaken the conditions of Theorem 6 by using the partial observer/lcc. Suppose $N$ is the index such that $E^N = K$; we denote $\Sigma_{b,E}$ as the boundary events of $E^i$, $\Sigma_{b,E} := \bigcup_{i=1}^{N} \Sigma_{b,E^i} \cap \Sigma_u$. We can use the partial natural observer/lcc to weaken the conditions of our theorem, as follows.

Theorem 7. If

1. $E$ is $(M, P_o)$-normal,
2. \( P_o \) is \((M, \Sigma_{b,u})\)-observer,

3. \( P_o \) is \((M, \Sigma_{b,u})\)-lcc,

then \( \Sigma_o \) is feasible for Problem (3.4), and thus Problem (3.2).

### 3.2.4 Comparison with Previous Results

In this section we compare our results to the recent work [29], as it is stronger than other results. Their approaches are proposed in decentralized architecture; we will give the vertical architecture version here for comparison.

The following theorem is the main result of [29].

**Theorem 8.** ([29]) If

1. \( \Sigma \cap \subseteq \Sigma_o \),
2. \( P_o \) is an \( M \)-observer,
3. \( P_o \) is an \( M \)-lcc,

then \( \Sigma_o \) satisfies the constraints of Problem (3.2).

Recall that \( \Sigma_{i,\cap} = \cup_{j \neq i} \Sigma_j \cap \Sigma_i \) is the local shared event set, and \( \Sigma_{\cap} = \cup_{i=1}^n \Sigma_{i,\cap} \) is the global shared events. Now we compare Theorem 8 to Theorem 6.

1. Theorem 6 requires that \( E \) be \((M, P_o)\)-normal;
2. Theorem 8 requires that \( \Sigma_{\cap} \subseteq \Sigma_o \).
Now we show that our condition is weaker than [29] by the next proposition.

**Proposition 9.** Suppose $P_o$ is an $M$-observer. If

$$\Sigma \cap \subseteq \Sigma_o$$

then $E$ is $(M, P_o)$-normal.

Therefore the conditions of [29] are stronger than ours.

### 3.2.5 Computational Procedure and Examples

Theorem 6 requires $\Sigma_o$ to satisfy that if

1. $E$ is $(M, P_o)$-normal;
2. $P_o$ is an $M$-observer;
3. $P_o$ is an $M$-lcc;

then the vertical supervisory control is maximally permissive and non-blocking. The next proposition says that if condition 1 can be satisfied first, then additional events can be added to $\Sigma_o$ without violating condition 1.

**Proposition 10.** Let $\Sigma_o \subseteq \Sigma'_o \subseteq \Sigma$, let $P_o : \Sigma^* \rightarrow \Sigma_o^*$ and $P'_o : \Sigma^* \rightarrow \Sigma_{o'}^*$. If $E$ is $(M, P_o)$-normal, then $E$ is $(M, P'_o)$-normal.

Therefore we can find $\Sigma_o$ in two steps, where the procedure is given below.

Now we provide an algorithm for Theorem 7.
Algorithm 2: Calculate $\Sigma_o$

**Data**: Generators $G(M), G(E)$

**Result**: feasible $\Sigma_o$ for Problem (3.2)

1. Initialize $\Sigma_o$ using Algorithm 1, with input $(G(M), G(E))$;

2. Augment $\Sigma_o$ that $P_o$ is $M$-observer, lcc with algorithm of [29].

We can use the Algorithm in [29], a procedure has been proposed to extend $\Sigma_o$ to be $M$-observer, lcc simultaneously.

The following example is to show that the conditions of Theorem 5 are weaker than Theorem 6. We don’t use TCT to calculate but analyze it to find a solution.

There are two local controllers $K_1, K_2$, which are shown in Fig. 3.12.

The event sets are below.

$\Sigma_1 = \{\alpha_1, \alpha_3, \beta_1, \beta_2, \beta_3, \beta_4\}$

$\Sigma_2 = \{\alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \beta_4\}$

$\Sigma = \{\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \beta_4\}$

$\Sigma_{\cap} = \Sigma_1 \cap \Sigma_2 = \{\alpha_3, \beta_1, \beta_2, \beta_3, \beta_4\}$

$\Sigma_{1,u} = \{\beta_1, \beta_2, \beta_3, \beta_4\}$

$\Sigma_{2,u} = \{\beta_1, \beta_2, \beta_3, \beta_4\}$

$\Sigma_u = \{\beta_1, \beta_2, \beta_3, \beta_4\}$
Algorithm 3: Calculate $\Sigma_o$

**Data:** Generators $G(M), G(E)$

**Result:** feasible $\Sigma_o$ for Problem (3.2)

1. Initialize $E' = E$, $\Sigma_o = \emptyset$;

2. Find events with Algorithm 1, with input $(G(M), G(E))$. Initialize $\Sigma_o$ with those events, name the uncontrollable events of the boundary of $E'$ to be $\Sigma_{u,c}$;

3. Augment $\Sigma_o$ to be $(M, \Sigma_{u,c})$-observer;

4. Augment $\Sigma_o$ to be $(M, \Sigma_{u,c})$-lcc;

5. Construct $M^{hi} = P_o(M), E^{hi'} = P_o(E')$;

6. If $P_o$ is not $(M, \Sigma_{u,c})$-observer, lcc, go to step 3;

7. If $E^{hi'} = E^{hi'} \cap \text{supC}(M^{hi}, E^{hi'}, \Sigma_u \cap \Sigma_o)$, go to step 11;

8. Set $E^{hi'} = E^{hi'} \cap \text{supC}(M^{hi}, E^{hi'}, \Sigma_o \cap \Sigma_u)$;

9. Find events with Algorithm 1(page 25), the input are $(G(M), G(E'))$. Name the uncontrollable events of the boundary of $E'$ as $\Sigma_{u,c}$.

10. If $P_o$ is $(M, \Sigma_{u,c})$-observer, lcc, then go to step 7. Else go to step 3;

11. Return $\Sigma_o$;

Table 3.3: Algorithm 3

The plant and specification are defined as

$$M = \overline{K_1} \parallel \overline{K_2}$$

$$E = K_1 \parallel K_2$$
The DES are shown in Fig. 3.13.

From Fig. 3.13, we see that $\Sigma_{b,E} = \beta_3$. Now we apply Algorithm 3 to find $\Sigma_o$.

Now we run Algorithm 3 to find the desired $\Sigma_o$. We write the steps below.
1. \( E' = E, \Sigma_o = \emptyset \);

2. Using Algorithm 1 to find \( \beta_1, \beta_2, \beta_3 \). Add them to \( \Sigma_o \);

3. \( P_o \) is already \( (M, \{\beta_3\}) \)-observer;

4. Add \( \{\alpha_3\} \) to \( \Sigma_o \) to make \( P_o \) \( (M, \{\beta_3\}) \)-lcc;

5. \( P_o \) is \( (M, \{\beta_3\}) \)-observer, lcc;

6. Construct \( M^{hi} = P_o(M), E^{hi'} = P_o(E') \);

7. Synthesize \( K^{hi'} = supC(M^{hi}, E^{hi'}, \Sigma_u \cap \Sigma_o) \);

8. The equation \( E^{hi'} = K^{hi'} \cap E^{hi'} \) holds true;

9. Return \( \{\alpha_3, \beta_1, \beta_2, \beta_3\} \).

The algorithm 2 runs in similar fashion to the steps above, but it adds events \( \{\alpha_1, \alpha_2\} \) to \( \Sigma_o \) at step 4, which don’t play a role in the control. Therefore partial lcc is a weaker condition than lcc.

The high-level plant and specification are

\[
M^{hi} : = P_o(M) \\
E^{hi} : = P_o(E)
\]
which are shown in 3.14.

\[ M^{hi} = L(G), \ E^{hi} = L_m(G) \]

The high-level controller is designed as

\[ K^{hi} = supC(M^{hi}, E^{hi}, \Sigma_u \cap \Sigma_o) \]

which is shown in Fig. 3.15.

\[ G(K^{hi}) \]

The monolithic controller \( K \) is designed as

\[ K = supC(M, \ E, \Sigma_u) \]

which is shown in Fig. 3.16.

Therefore the high-level controller in vertical control is designed without building \( K \).
The local projections for decentralized architecture devised by [29] are shown below:

\[ \Sigma'_{1,o} = \Sigma_1, \quad Q'_1 : \Sigma_1 \to \Sigma'_{1,o} \]
\[ \Sigma'_{2,o} = \Sigma_2, \quad Q'_2 : \Sigma_2 \to \Sigma'_{2,o} \]

This means that the decentralized approach by [29] reduces to the monolithic control in this example, while the following definitions achieve the objectives of decentralized control.

\[ \Sigma_{1,o} = \Sigma_1 \cap \Sigma_o, \quad Q_1 : \Sigma_1 \to \Sigma_{1,o} \]
\[ \Sigma_{2,o} = \Sigma_2 \cap \Sigma_o, \quad Q_2 : \Sigma_2 \to \Sigma_{2,o} \]

The local projections are more economical than the ones devised in [29]. The reason is because the conditions of Theorem 5 are less restrictive than the conditions of [29]. Therefore the conditions of Theorem 7 can help to devise economical decentralized supervisory control.

Finally we review the example of the Transfer-Line system, and apply Theorem 6 to design the controller without building \( K \). The system is described in Section 3.1, but we put the critical items below for the reader’s convenience.
There are three machines $G_1, G_2, G_3$ and two buffers $B_1, B_2$ in the Transfer-Line system. Their generators are shown in Fig. 3.18.

According to Section 2.3, we first prepare the local plants and local specifications.

\[
M_1 := \text{L}(G_1) \parallel \text{L}(G_2) \parallel \text{L}(G_3)
\]
\[
E_1 := \text{L}_m(G_1) \parallel \text{L}_m(G_2) \parallel \text{L}_m(G_3) \parallel \text{L}_m(B_1)
\]
\[
M_2 := \text{L}(G_2) \parallel \text{L}(G_3)
\]
\[
E_2 := \text{L}_m(G_2) \parallel \text{L}_m(G_3) \parallel \text{L}_m(B_2)
\]

The local controllers are

\[
K_1 := \text{supC}(M_1, E_1, \Sigma_{1,u})
\]
\[
K_2 := \text{supC}(M_2, E_2, \Sigma_{2,u})
\]

The DES are shown in Fig. 3.19.
The global plant and specification are

\[ M := \overline{K_1} \parallel \overline{K_2} \]
\[ E := K_1 \parallel K_2 \]

The DES are shown in Fig. 3.20

Now we apply Theorem 6 here. Namely we have to find \( \Sigma_o \) which satisfies the following conditions:

1. \( E \) is \((M, P_o)\)-observer;
2. $P_o$ is $M$-observer;

3. $P_o$ is $M$-lcc;

Algorithm 2 will find $\Sigma_o = \{1, 6\}$ in the first step, which satisfies Problem 3.2. The high-level plant and specification are then generated as:

$$
M^{hi} = P_o(M)
$$
$$
E^{hi} = P_o(E)
$$

which are shown in Fig. 3.21

![Figure 3.21: $M^{hi} = L(G), E^{hi} = L_m(G)$](image1)

The high-level controller is designed as

$$
K^{hi} = \sup C(M^{hi}, E^{hi}, \Sigma_o \cap \Sigma_u)
$$

which is shown in Fig. 3.22

![Figure 3.22: $G(K^{hi})$](image2)

Therefore the high-level controller is built without building $K$.

### 3.3 Chapter Summary and Conclusions

The controller abstraction and hierarchical supervisory control challenges are unified into Problem (2.1) in Chapter 2. If the monolithic controller
$K$ is known, then the high-level controller $K^{hi}$ is built based on $K$. If the monolithic controller is unknown, then the high-level controller is built based on the high-level plant and specification, which are the image of plant and specification under $P_o$. In both scenarios, the task is to find the minimal observable events $\Sigma_o$ for $P_o$ for maximally permissive and nonblocking control.

In this chapter, we provide a complete solution when the monolithic controller $K$ is known. Theorem 1 provides the necessary and sufficient conditions for $\Sigma_o$ to solve Problem (2.1). Furthermore, a practical procedure is proposed to find a solution for Problem (2.1) instead of enumerating all subsets of $\Sigma$.

When the monolithic controller is unknown, sufficient conditions for a feasible solution to Problem (2.1) are proposed in Theorem 6. The conditions are further weakened in Theorem 7 by employing partial observer/lcc. Practical algorithms are proposed to find $\Sigma_o$ satisfying those theorems.
Chapter 4

Application

In this chapter, we apply the results of Chapter 3 to hierarchical supervisory control of DES. Specifically we generalize the results of Section 3.1 to hierarchical controllers. Recall that Theorem 1 of Section 3.1 provides a way to design an abstraction for the monolithic controller; here we generalize it to hierarchical controllers.

In Section 4.1 we generalize Theorem 1 to multiple high-level controllers. An example of a large-scale hierarchical supervisory control application will be introduced in Section 4.2 to show the effectiveness of our method.

4.1 Abstractions of Hierarchical Controllers

Hierarchical supervisory control has two benefits over monolithic supervisory control. In hierarchical architecture, multiple small high-level controllers are employed in supervising the system rather than one single monolithic controller; thus it is much easier to understand each high-level controller than the large monolithic counterpart. The second advantage is
that it can be computationally economical to build hierarchical controllers. In this chapter we focus on the transparency of hierarchical controllers.

An illustration is shown in Figure 4.1. There are five local controllers $K_1, \ldots, K_5$ which are designed based on $(M_1, E_1), \ldots, (M_5, E_5)$. Three hierarchical controllers $K_{sub_1}, K_{sub_2}, K_{sub_3}$ are designed afterwards with the hierarchical control method of [29]. Those three controllers working with local controllers can achieve (1) Maximally Permissive, (2) Nonblocking control. However, the generators of $K_{sub_1}, K_{sub_2}, K_{sub_3}$ may be too complicated for understanding. Therefore it is desired to have smaller abstractions of the hierarchical controllers, which achieve maximally permissive and nonblocking control but are simpler for understanding.

The problem setting is given here. Let $(M_1, E_1), \ldots, (M_n, E_n)$ be the local systems, such that $M_i \subseteq \Sigma_i^*$ is closed and $E_i \subseteq M_i$, $\Sigma = \cup_i \Sigma_i$ is the global event set. Let $\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,u}$, where $\Sigma_{i,u}$ is the local uncontrollable event set and $\Sigma_{i,c}$ is the controllable event set, $\Sigma_u = \cup_{i=1}^n \Sigma_{i,u}$ is the global uncontrollable events and $\Sigma_c = \cup_{i=1}^n \Sigma_{i,c}$ is the global controllable events. The monolithic controller is $K := supC(M, E, \Sigma_u)$. Suppose
$M_{\text{sub}_i}, i = 1, \ldots, m$ are the high-level plants. Furthermore, multiple controllers $K_{\text{sub}_i} \subseteq \Sigma^*_{\text{sub}_i}, \ldots, K_{\text{sub}_m} \subseteq \Sigma^*_{\text{sub}_m}$, where $\Sigma_{\text{sub}_i} \subseteq \Sigma, \forall i = 1, \ldots, m$, are designed by hierarchical approaches so that they achieve (1) Maximally Permissive, (2) Nonblocking control. The problem is to find projections $Q_{\text{sub}_i} : \Sigma^*_{\text{sub}_i} \rightarrow \Sigma^*_{\text{sub}_i,o}$ that preserve (1) Maximally Permissive (2) Nonblocking property. The problem is formalized as:

$$\text{minimize } | \bigcup_{i=1}^n \Sigma_{\text{sub}_i,o} | \quad \text{subject to }$$

$$\| \bigcup_{j=1}^m Q_{\text{sub}_j}(K_{\text{sub}_j}) \| M = K \quad \text{(Maximally Permissive)}$$

$$\| \bigcup_{j=1}^m Q_{\text{sub}_j}(\overline{K_{\text{sub}_j}}) \| M = \overline{K} \quad \text{(Nonblocking)}$$

Problem (4.1) seeks to minimize the observable events of high-level controllers under constraints that (1) Maximally Permissive, (2) Nonblocking control are achieved. Abstraction $K_{\text{sub}_i}^{ab} := Q_{\text{sub}_i}(K_{\text{sub}_i})$ preserves the critical information of $K_{\text{sub}_i}$, which may be simpler than $K_{\text{sub}_i}$. Therefore the abstractions may help to improve the transparency of the hierarchical controllers.

The next theorem provides a necessary and sufficient condition for $\Sigma_{\text{sub}_i,o}, i = 1, \ldots, m$, to satisfy the constraints.

**Theorem 9.** For $i = 1, \ldots, m$, $\Sigma_{\text{sub}_i,o}$ satisfies the constraints of Problem 4.1 if for all $i = 1, \ldots, m$

$$1. K_{\text{sub}_i} \text{ is } (M_{\text{sub}_i}, Q_{\text{sub}_i})\text{-normal};$$
$$2. \overline{K_{\text{sub}_i}} \text{ is } (M_{\text{sub}_i}, Q_{\text{sub}_i})\text{-normal}.$$
Therefore Problem (4.1) can be transformed to the following one:

\[
\text{minimize } \sum_{i=1}^{m} |\Sigma_{sub,o}| \tag{4.2}
\]

subject to \((\forall j = 1, \ldots, m)\)

\[K_{sub_j} \text{ is } (M_{sub_i}, Q_{sub_i})\text{-normal}\]

\[\overline{K_{sub_j}} \text{ is } (M_{sub_i}, Q_{sub_i})\text{-normal}\]

\(\Sigma_{sub,o}\) can be individually calculated by Algorithm 1, since the constraints don’t couple with each other. In the next section we illustrate our abstractions of hierarchical controllers, and show that it significantly improves the transparency of the controllers.

### 4.2 A Triple Loop Example

In this example, we apply our previous results to a large system, of which the final monolithic controller’s size is over \(10^5\). The triple loop system is shown in Fig. 4.2, where there are 7 machines \(G_1, \ldots, G_7\) and 6 buffers \(B_1, \ldots, B_6\). The numbers represent events; the odd events are controllable while the even ones are uncontrollable.

We could interpret the loop as a checking procedure, that is if a product is not confirmed to have been correctly processed, then it has to be sent back to previous processing steps. In this regard, the capacities of buffers \(B_i, i = 1, \ldots, 6\) provide constraints on how many products could be in process simultaneously.
Figure 4.2: Triple Loop System

Take a look at the smallest internal loop, which consists of $G_3, B_3, G_4, B_4, G_5$. Suppose that the slots of $B_3, B_4$ are all occupied. If event 7 occurs, then event 8 may occur since it is uncontrollable, which will violate the constraint of $B_4$. If event 9 occurs, then event 16 may occur, which will violate the constraint of $B_3$. Event 5 should also be disabled, as it may lead to an occurrence of event 6, which violates the constraint of $B_3$. Therefore we find that although it is feasible to utilize all the capacities of $B_3, B_4$, it may lead to a deadlock where the processing will halt. Therefore it is necessary to design controllers for the system to avoid deadlock problems.

The generator models of the machines $G_1, \ldots, G_7$ are given in figure 4.3:

The generator models of buffers are shown in figure 4.4. The capacities of $B_1, \ldots, B_6$ are 4, 3, 2, 2, 3, 4 respectively.

There are 6 local systems, of which the local plant $M_i$, and local specification $E_i$ are modeled as
\[ M_1 = L(G_1) \parallel G_2 \parallel G_7, \quad E_1 = L_m(G_1) \parallel L_m(G_2) \parallel L_m(G_7) \parallel L_m(B_1) \]
\[ M_2 = L(G_2) \parallel G_3 \parallel G_6, \quad E_2 = L_m(G_2) \parallel L_m(G_3) \parallel L_m(G_6) \parallel L_m(B_2) \]
\[ M_3 = L(G_3) \parallel G_4 \parallel G_5, \quad E_3 = L_m(G_3) \parallel L_m(G_4) \parallel L_m(G_5) \parallel L_m(B_3) \]
\[ M_4 = L(G_4) \parallel L_m(G_5), \quad E_4 = L_m(G_4) \parallel L_m(G_5) \parallel L_m(B_4) \]
\[ M_5 = L(G_5) \parallel L_m(G_6), \quad E_5 = L_m(G_5) \parallel L_m(G_6) \parallel L_m(B_5) \]
\[ M_6 = L(G_6) \parallel G_7, \quad E_6 = L_m(G_6) \parallel L_m(G_7) \parallel L_m(B_6) \]
The event sets are

\[ \Sigma_1 = \{1, 2, 3, 4, 13, 14, 20\} \]
\[ \Sigma_2 = \{3, 4, 5, 6, 11, 12, 18\} \]
\[ \Sigma_3 = \{5, 6, 7, 8, 9, 10, 16\} \]
\[ \Sigma_4 = \{7, 8, 9, 10, 16\} \]
\[ \Sigma_5 = \{9, 10, 11, 12, 16, 18\} \]
\[ \Sigma_6 = \{11, 12, 13, 14, 18, 20\} \]
\[ \Sigma_{1,u} = \{2, 4, 14, 20\} \]
\[ \Sigma_{2,u} = \{4, 6, 12, 18\} \]
\[ \Sigma_{3,u} = \{6, 8, 10, 16\} \]
\[ \Sigma_{4,u} = \{8, 10, 16\} \]
\[ \Sigma_{5,u} = \{10, 12, 16, 18\} \]
\[ \Sigma_{6,u} = \{12, 14, 20\} \]

The local controllers are designed as

\[ K_1 = supC(M_1, E_1, \Sigma_{1,u}) \quad (32, 98) \]
\[ K_2 = supC(M_2, E_2, \Sigma_{2,u}) \quad (24, 70) \]
\[ K_3 = supC(M_3, E_3, \Sigma_{3,u}) \quad (16, 42) \]
\[ K_4 = supC(M_4, E_4, \Sigma_{4,u}) \quad (10, 21) \]
\[ K_5 = supC(M_5, E_5, \Sigma_{5,u}) \quad (14, 37) \]
\[ K_6 = supC(M_6, E_6, \Sigma_{6,u}) \quad (18, 49) \]

The bracket shows the size of the generator, where the first number is the number of states of the generator, and the second one is the number of transitions.
The monolithic controller is calculated as

\[
M = \|_{i=1}^{6} K_i \\
E = \|_{i=1}^{6} K_i \\
K = supC(M, E, \Sigma_u) \tag{100416, 587337}
\]

Thus \(K\) has 100416 states, and 587337 transitions.

Now we design the hierarchical controllers. The hierarchical architecture is shown in Fig. 4.5.

![Three-level Hierarchical Architecture](image)

Figure 4.5: Three-Level Hierarchical Architecture

The hierarchy consists of three levels, each level includes one loop of
the subsystems. $K_3, K_4$ forms the first loop, thus they form the first level. $K_2, K_5, K_{sub1}$ forms the second loop, thus we group them in the second level. Finally the $K_1, K_6, K_{sub2}$ forms a loop, that we group them in the third level. We use the method of [29] to compute the hierarchical controller $K_{sub1}, K_{sub2}, K_{sub3}$.

Now we give the corresponding observable events preserved for each decentralized control.

$K_3, K_4$ are grouped in level 1, and the decentralized controller $K_{sub1}$ is designed as:

$$\Sigma_{sub1} = \{5, 6, 7, 8, 9, 10, 16\}$$

$$\Sigma_{sub1,u} = \Sigma_{3,u} \cap \Sigma_{4,u} \cap \Sigma_{sub1}$$

$$Q_3 : \Sigma_1^* \rightarrow (\Sigma_3 \cap \Sigma_{sub1})^*$$

$$Q_4 : \Sigma_2^* \rightarrow (\Sigma_4 \cap \Sigma_{sub1})^*$$

$$K_{hi3} = Q_3(K_3)$$

$$K_{hi4} = Q_4(K_4)$$

$$M_{sub1} = \overline{K_{hi3}^*} \parallel K_{hi4}^*$$

$$E_{sub1} = K_{hi3}^* \parallel K_{hi4}^*$$

$$K_{sub1} = supC(M_{sub1}, E_{sub1}, \Sigma_{sub1,u}) \quad (32, 73)$$

Thus $K_{sub1}$ has 32 states, and 73 transitions.

The we group $K_2, K_{sub1}, K_5$ to be the second level, and then design the
decentralized controller $K_{sub_2}$.

$$\Sigma_{sub_2} = \{3, 4, 5, 6, 9, 10, 11, 12, 16, 18\}$$

$$\Sigma_{sub_2,u} = \Sigma_{3,u} \cap \Sigma_{4,u} \cap \Sigma_{sub_1,u} \cap \Sigma_{sub_2}$$

$$Q_2 : \Sigma_2^* \to (\Sigma_2 \cap \Sigma_{sub_2})^*$$

$$Q_5 : \Sigma_5^* \to (\Sigma_5 \cap \Sigma_{sub_2})^*$$

$$Q_{sub_1} : \Sigma_{sub_1}^* \to (\Sigma_{sub_1} \cap \Sigma_{sub_2})^*$$

$$K_{hi}^2 = Q_2(K_2)$$

$$K_{hi}^5 = Q_5(K_5)$$

$$K_{hi}^{\text{sub}_1} = Q_{sub_1}(K_{sub_1})$$

$$M_{sub_2} = K_{hi}^2 \parallel \overline{K_{hi}^5} \parallel \overline{K_{hi}^{\text{sub}_1}}$$

$$E_{sub_2} = K_{hi}^3 \parallel \overline{K_{hi}^4} \parallel \overline{K_{hi}^{\text{sub}_1}}$$

$$K_{sub_2} = \text{sup}_{C}(M_{sub_2}, E_{sub_2}, \Sigma_{sub_2,u}) \quad (508, 1895)$$

Thus $K_{sub_2}$ has 508 states, and 1895 transitions.

Finally we group $K_1, K_6, K_{sub_2}$ together, and design the decentralized controller $K_{sub_3}$.

$$\Sigma_{sub_3} = \{1, 2, 3, 4, 11, 12, 13, 14, 18, 20\}$$

$$\Sigma_{sub_3,u} = \Sigma_{1,u} \cap \Sigma_{6,u} \cap \Sigma_{sub_2,u} \cap \Sigma_{sub_3}$$

$$Q_1 : \Sigma_1^* \to (\Sigma_1 \cap \Sigma_{sub_3})^*$$

$$Q_6 : \Sigma_6^* \to (\Sigma_6 \cap \Sigma_{sub_3})^*$$

$$Q_{sub_2} : \Sigma_{sub_2}^* \to (\Sigma_{sub_2} \cap \Sigma_{sub_3})^*$$

$$K_{hi}^1 = Q_1(K_2)$$

$$K_{hi}^6 = Q_6(K_5)$$

$$K_{hi}^{\text{sub}_2} = Q_{sub_2}(K_{sub_2})$$

$$M_{sub_3} = \overline{K_{hi}^1} \parallel \overline{K_{hi}^6} \parallel \overline{K_{hi}^{\text{sub}_2}}$$

$$E_{sub_3} = K_{hi}^1 \parallel \overline{K_{hi}^6} \parallel \overline{K_{hi}^{\text{sub}_2}}$$

$$K_{sub_3} = \text{sup}_{C}(M_{sub_3}, E_{sub_3}, \Sigma_{sub_3,u}) \quad (2312, 9615)$$
Thus $K_{sub_3}$ has 2312 states, and 9615 transitions.

With the hierarchical controllers $K_{sub_1}, K_{sub_2}, K_{sub_3}$ at hand, we need to find the abstractions. We can apply Algorithm 1 to find $\Sigma_{sub_1,o}, \Sigma_{sub_2,o}, \Sigma_{sub_3,o}$. Indeed, the events are found in the initialization steps of Algorithm 1, thus no iterations of adding events are needed.

For $K_{sub_1}$, events $\{5\}$ point from a marked state to an unmarked state, events $\{10\}$ point from an unmarked state to a marked state, thus they are added to $\Sigma_o$. Furthermore $\{5, 10\}$ satisfies the property that $K_{sub_1}$ is $(M_{sub_1}, P_o)$-normal, while no single event can satisfy it, hence it is the minimal solution to Problem (3.1). Similarly we find $\{3, 12\}$ as the observable events for $K_{sub_2}$, and $\{1, 14\}$ as the observable events for $K_{sub_3}$ for maximally permissive and nonblocking control. The results are written below.

\[
\begin{align*}
\Sigma_{sub_1,o} &= \{5, 10\} \\
\Sigma_{sub_2,o} &= \{3, 12\} \\
\Sigma_{sub_3,o} &= \{1, 14\} \\
Q_{sub_1} &= \Sigma_{sub_1} \rightarrow \Sigma_{sub_1,o} \\
Q_{sub_2} &= \Sigma_{sub_2} \rightarrow \Sigma_{sub_2,o} \\
Q_{sub_3} &= \Sigma_{sub_3} \rightarrow \Sigma_{sub_3,o} \\
K_{ab_{sub_1}} &= Q_{sub_1}(K_{sub_1}) \quad (4, 6) \\
K_{ab_{sub_2}} &= Q_{sub_2}(K_{sub_2}) \quad (9, 16) \\
K_{ab_{sub_3}} &= Q_{sub_3}(K_{sub_3}) \quad (16, 30)
\end{align*}
\]

We display all the results in the next table below.
### Table 4.1: Three Types of Controllers

<table>
<thead>
<tr>
<th></th>
<th>Monolithic</th>
<th>Decentralized</th>
<th>Our Abstractions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstractions</td>
<td>(100416,587337)</td>
<td>(508,1895)</td>
<td>(9,16)</td>
</tr>
<tr>
<td>Abstractions</td>
<td>(32,73)</td>
<td>(4,6)</td>
<td>(16,30)</td>
</tr>
</tbody>
</table>

Since we can compute the monolithic controller in this example, thus we can verify that the hierarchical controllers or their abstractions achieve the maximally permissive and nonblocking control. The result confirms our theoretical results of Section 4.1.

Our abstractions of the high-level controllers are shown in Fig. 4.6.

![Abstractions of Hierarchical Controllers](image)

Figure 4.6: Abstractions of Hierarchical Controllers

We can see that the abstractions of hierarchical controllers resemble the buffers $B_i, i = 1, \ldots, 6$. It has been discussed in [42] that the controller/abstraction of a buffer system behaves similar to a buffer. Further-
more, a coordinator for the Transfer-Line system is designed in [42] by considering the capacities of the buffers. However, that approach is highly nontrivial as it requires a deep understanding of the structure, while our abstractions are designed automatically without human intervention.

Finally we compare our result to two other approaches towards simpler abstraction of controller. Localization aims to decompose the controller to each local plant, thus a high-level controller can be distributed to its local machines $G_i$. SupReduction aims to find a simplified controller by reducing the redundancy of generators of $K_{sub_1}, K_{sub_2}, K_{sub_3}$. All computations are carried out in TCT. We show the results below.

<table>
<thead>
<tr>
<th></th>
<th>Decentralized</th>
<th>Ours</th>
<th>Localization</th>
<th>SupReduce</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_{sub_1}$</td>
<td>(32,73)</td>
<td>(4,6)</td>
<td>(7,23)(14,60)(8,30)</td>
<td>(18,67)</td>
</tr>
<tr>
<td>$K_{sub_2}$</td>
<td>(508,1895)</td>
<td>(9,16)</td>
<td>(36,251)(26,126) /</td>
<td>(254,1607)</td>
</tr>
<tr>
<td>$K_{sub_3}$</td>
<td>(2312,9615)</td>
<td>(16,30)</td>
<td>/</td>
<td>(1170,8326)</td>
</tr>
</tbody>
</table>

Table 4.2: Comparison of Abstraction Reduction Methods

Localization will decompose the hierarchical controller into several parts, which are related to how many machines the high-level plant consists of. For example, the first level plant consists of three machines $G_3, G_4, G_5$, thus $K_{sub_1}$ is decomposed to three local controllers. However, the procedure is time consuming in that it doesn’t generate all local controllers for the $K_{sub_2}$ after 5 hours. Furthermore it doesn’t decompose $K_{sub_3}$ successfully after 5 hours. Finally, the results of SupReduce of the hierarchical controllers are much larger than our hierarchical abstractions of controllers $K_{sub_1}^{ab}, K_{sub_2}^{ab}, K_{sub_3}^{ab}$. 
4.3 Chapter Summary and Conclusions

In this Chapter, we generalize the results of abstraction of monolithic controller to hierarchical controllers. In words, given hierarchical controllers achieving maximally permissive and nonblocking control, we provide necessary and sufficient conditions for the abstractions to maintain the two objectives, while discarding all unnecessary events for simpler expression. The algorithm devised in Section 3.1.2 can be applied to hierarchical controller abstraction.

In the example, the state size of the monolithic controller is over $10^6$, which is infeasible for understanding. Although it is computationally economical to devise multiple hierarchical controllers instead of a monolithic one, the state size of the hierarchical controllers is still too complicated for comprehension. Besides our abstraction technique, two other standard methods as SupReduction and Localization are applied to the hierarchical controllers for smaller abstractions. Their abstractions are either time consuming or not sufficiently small for understanding, while the states of our abstractions can be significantly smaller. Particularly we can reduce the states of one hierarchical controller from 2312 to 16, which makes the controller easy to understand, whereas the other two approaches can not reduce the controllers to a comprehensible size.
Chapter 5

Conclusions and Future Work

Ramadge-Wonham supervisory control [26] suffers from exponential complexity of the computation, while hierarchical supervisory control well structures the problem by dividing the system into small groups of subsystems effectively. Each group of subsystems can be supervised by an individual controller designed by a vertical approach, or decentralized approach. Nevertheless, two problems of hierarchical control are still open. The first challenge is to find simpler abstractions of the hierarchical controllers for easier understanding. The second is to design the hierarchical controllers to achieve maximally permissive and nonblocking control.

In this thesis, the two problems are unified into one optimization problem, which seeks the minimal observable event set for the high-level controllers to achieve maximally permissive and nonblocking control. An optimal solution to controller abstraction is proposed. A suboptimal solution is found for vertical controller design, which helps to analyze hierarchical supervisory control.
5.1 Main Contributions

1. **Incorporating controller abstraction and hierarchical supervisory control into one problem.** Controller abstraction and hierarchical supervisory control problems have been researched for a long time, while the approaches are devised individually for each problem. Although some previous work discussed the relationship between the two problems, no research has formally unified them together. In this thesis, the two challenges are formally unified into one optimization problem, where the objectives are to find the minimal observable events for maximally permissive and nonblocking control. The difference between the two problems is that when the monolithic controller $K$ is known, then the high-level controller $K^{hi}$ (abstraction) is built based on $K$, while if $K$ is not known, then $K^{hi}$ is built based on the high-level plant and specification which are the images of plant and specification under projection $P_o$. Therefore the solution to the first scenario corresponds to controller abstraction, and the solution to the second scenario corresponds to hierarchical supervisory control.

2. **An optimal solution to controller abstraction.** Various approaches towards constructing simpler controllers for understanding have been proposed, but they mainly focus on minimising the redundancies of the generators of controller. In contrast, we focus on the redundancy of the controller’s language rather than its generator. We solve the problem completely by providing necessary and sufficient conditions for the abstractions to achieve maximally permissive and nonblocking control.

3. **Weaker conditions for vertical supervisory control.** Hierarchical supervisory control is composed of multiple levels of vertical supervisory control, where at each level a controller is designed to supervise the grouped subsystems. We provide sufficient conditions for maximally permissive and nonblocking vertical supervisory control. Furthermore our conditions are weaker than those of most recent work [12, 29].
Chapter 5. Conclusions and Future Work

5.2 Future Work

1. **Efficient Algorithm to find $\Sigma_o$ for $K$, such that $K$ is $(M, P_o)$-normal.** In Section 3.1 and Section 4.1 we see that the critical part of building abstractions of controllers is to find the minimal observable events that $K$ is normal w.r.t. $(M, P_o)$. Although we provide an empirical algorithm to find a reasonable $\Sigma_o$, it is not guaranteed to find the minimal one. Therefore it is interesting to analyze the complexity of the problem of finding a minimal observable events $\Sigma_o$ for $K$ to be $(M, P_o)$-normal. Furthermore, it is important to build a more efficient algorithm than the proposed one.

2. **Implement partial natural observer and lcc.** Conventional natural observer and lcc are sufficient to achieve maximally permissive and nonblocking control, but their definitions require the projections to hold for all observable events or uncontrollable observable events, which is overkill. Partial versions of natural observer and lcc can potentially be more economical than conventional observer, lcc. Therefore it is interesting to implement them in TCT in the future.

3. **Extending results to decentralized supervisory control.** In Section 3.2 we have proposed sufficient conditions for maximally permissive and nonblocking control in vertical supervisory control. Since decentralized supervisory control can be more economical than vertical supervisory control, thus it is interesting to extend those conditions to decentralized architecture. Namely we hope to have conditions on the projections for each subsystem to achieve maximally permissive and nonblocking control.
5.3 Final Remark

Two open problems in the development of DES control theory are critically important, namely understanding the controller and designing the controller. In this thesis we provide a full solution to the projection-based controller abstraction, while our results towards the second problem are far from complete. Since one major obstacle for real applications of DES control theory is the high complexity of computation, thus further investigation for efficient hierarchical supervisory control will surely be fruitful for the development of DES control theory.
Bibliography


Appendices
Appendix A

Proofs

For the reader’s convenience, we put all the proofs of our results into this Appendix. Section A.1 consists of proofs of the results of Section 3.1. Section A.2 consists of proofs of the results of Section 3.2. Section A.3 consists of proofs of the results of Section 4.1.

A.1 Proofs for Section 3.1

A.1.1 Proposition 2

Proof. (Proposition 2)
We first prove \( \Rightarrow \) direction. Since \( s \in M, P_o(s) \in P_o(N) \), and \( N \) is \( (M, P_o) \)-normal, thus \( s \in M \parallel P_o(N) = N \), hence \( s \in N \).

Now we prove \( \Leftarrow \) direction. Since \( N \subseteq M \), and \( N \subseteq P_o^{-1}(P_o(N)) \), thus \( M \parallel P_o(N) \subseteq N \). On the other hand, if \( s \in M \parallel P_o(N) \), then \( s \in M \) and \( P_o(s) \in P_o(N) \), thus \( s \in N \) according to the condition, hence \( M \parallel P_o(N) \subseteq N \). Therefore \( M \parallel P_o(N) = N \), which means \( N \) is \( (M, P_o) \)-
A.1.2 Lemma 1

Proof. (Lemma 1)
We have to show that $M \parallel K^{hi} = K$, where $K^{hi} := P_o(K)$. Thus the statement is $M \parallel P_o(K) = K$ iff $K$ is $(M, P_o)$-normal.

The proof is obvious since

$$M \parallel P_o(K) = K \text{ iff } K \text{ is } (M, P_o)\text{-normal.}$$

\[\square\]

A.1.3 Lemma 2

Proof. (Lemma 2)
The condition is that $M \parallel K^{hi} = K$, where $K^{hi} := P_o(K)$. Thus the statement is $M \parallel P_o(K) = K$ iff $K$ is $(M, P_o)$-normal. The proof is obvious since

$$M \parallel P_o(K) = K \text{ iff } K \text{ is } (M, P_o)\text{-normal.}$$

Finally we have that $P_o(K) = P_o(K)$ to finish the proof.  \[\square\]

A.1.4 Theorem 1

Proof. (Theorem 1)
With lemma 1, 2, we have that if $K, \overline{K}$ are $(M, P_o)$-normal, then the
Appendix A. Proofs

Constraints of Problem (2.1) are satisfied. On the other hand, if the constraints of Problem (2.1) are satisfied, then Lemma 1 means that $K$ is $(M, P_o)$-normal, and Lemma 2 means that $\overline{K}$ is $(M, P_o)$-normal.

\[ \square \]

A.2 Proofs for Section 3.2

A.2.1 Lemma 3

Proof. (Lemma 3)
We first prove $\Rightarrow$ direction, namely assuming that $L$ is $(M, \Sigma_u)$ controllable, to prove that $P_o(\overline{L})$ is $(P_o(M), \Sigma_u \cap \Sigma_o)$ controllable. Given a string $t \in P_o(L)$, and an uncontrollable event $\sigma_u \in \Sigma_u \cap \Sigma_o$, $t\sigma_u \in P_o(M)$, then $\exists s \in M$ such that $s\sigma_u \in M$ and $P_o(s) = t$. Since $\overline{L}$ is $(M, P_o)$-normal, with Proposition 2, we have that $s \in \overline{L}$. Since $L$ is $(M, \Sigma_u)$ controllable, thus $s\sigma_u \in \overline{L}$, hence $t\sigma_u = P_o(s)\sigma_u \in P_o(\overline{L})$. Therefore $P_o(L)$ is $(P_o(M), \Sigma_u \cap \Sigma_o)$ controllable.

Given $P_o(L)$ is $(M, \Sigma_u \cap \Sigma_o)$ controllable, we have to show that $L$ is $(M, \Sigma_u)$ controllable. Let $s \in \overline{L}$, let $\sigma_u \in \Sigma_u$ such that $s\sigma_u \in M$. Since $P_o(L)$ is $(P_o(M), \Sigma_u \cap \Sigma_o)$ controllable, thus $P_o(s\sigma_u) \in P_o(\overline{L})$. Furthermore, since $\overline{L}$ is $(M, P_o)$-normal, thus $s\sigma_u \in \overline{L}$ due to Proposition 2. Therefore $L$ is $(M, \Sigma_u)$ controllable.

Putting the above two parts together, we have proved the lemma. \[ \square \]
Appendix A. Proofs

A.2.2 Theorem 2

Proof. (Theorem 2)

Recall that the high-level controller is defined as $K^{hi} := \sup C(M^{hi}, E^{hi}, \Sigma_u \cap \Sigma_o)$, where $M^{hi} := P_o(M), E^{hi} := P_o(E)$. We prove $\Rightarrow$ direction. Assuming that $E$ is $(M, P_o)$-normal, and $\Sigma_o$ satisfies the constraints of Problem (2.1), we have to show that $K, \overline{K}$ are $(M, P_o)$-normal. First we prove that $K^{hi} = P_o(K)$. Since $K \subseteq E$, thus $P_o(K) \subseteq P_o(E)$. Furthermore, Lemma 3 implies that $P_o(K)$ is controllable w.r.t. $(P_o(M), \Sigma_u \cap \Sigma_o)$, hence $P_o(K) \subseteq K^{hi}$. Next we prove the other direction. Since $E$ is $(M, P_o)$-normal, thus $K^{hi} \subseteq P_o(E)$ implies that $M \parallel K^{hi} \subseteq E$ due to Proposition 2. Furthermore Lemma 3 says that $M \parallel K^{hi}$ is controllable w.r.t. $(M, \Sigma_i)$, thus $M \parallel K^{hi} \subseteq K$, hence $P_o(K) \subseteq P_o(E)$, thus for any string $t \in K^{hi}$, there exists $s \in E$ such that $P_o(s) = t$. Furthermore since $E$ is $(M, P_o)$-normal, thus $K^{hi} \subseteq P_o(E)$ implies that $M \parallel K^{hi} \subseteq E$ due to Proposition 2. Additionally, Lemma 3 says that $M \parallel K^{hi}$ is controllable w.r.t. $(M, \Sigma_u)$, thus $M \parallel K^{hi} \subseteq K$, hence $s \in K$. Therefore $K^{hi} \subseteq P_o(K)$. Now we have $P_o(K) = K^{hi}$. The two constraints in Problem (2.1) thus imply that $K, \overline{K}$ are $(M, P_o)$-normal.

We prove the other direction. Suppose $E$ is $(M, P_o)$-normal. Given that $K, \overline{K}$ are $(M, P_o)$-normal, we have to show that $\Sigma_o$ satisfies the constraints of Problem (2.1). If we can show that $P_o(K) = K^{hi}$, then the proof is done. Lemma 3 says that $P_o(K)$ is controllable w.r.t. $(M, \Sigma_u)$, thus $P_o(K) \subseteq K^{hi}$. On the other hand, if $t \in K^{hi}$, then $\exists s \in E$ such that $P_o(s) = t$. Furthermore, Proposition 2 implies that $M \parallel K^{hi} \subseteq E$; Lemma 3 implies that $M \parallel K^{hi}$ is controllable w.r.t. $(M, \Sigma_u)$. Thus $s \in K$, hence $t \in P_o(K)$. Therefore we have proved that $P_o(K) = K^{hi}$. Finally $K, \overline{K}$ being $(M, P_o)$-normal imply that $\Sigma_o$ satisfies the two constraints of Problem (2.1).
A.2.3 Proposition 3

Proof. (Proposition 3)
Since $E$ is closed, thus $\overline{\text{supC}(M, E, \Sigma_u)} \subseteq E$. Furthermore, $\overline{\text{supC}(M, E, \Sigma_u)}$ is controllable w.r.t. $(M, \Sigma_u)$. Therefore $\text{supC}(M, E, \Sigma_u) = \overline{\text{supC}(M, E, \Sigma_u)}$, which implies that $\text{supC}(M, E, \Sigma_u)$ is closed. □

A.2.4 Proposition 4

Proof. (Proposition 4)
The proof can be easily derived from Proposition 2.

To prove $\Rightarrow$ direction, Proposition 2 implies that $s \in L \iff P_o(s) \in P_o(L)$ and $s\sigma \in M - L \iff P_o(s\sigma) \in P_o(M) - P_o(L)$.

To prove $\Leftarrow$ direction, from conditions $s \in L \land s\sigma \in M - L \iff P_o(s) \in P_o(L) \land P_o(s\sigma) \in P_o(M) - P_o(L)$, we have that $s \in L \iff P_o(s) \in L$.
Therefore Proposition 2 implies that $L$ is $(M, P_o)$-normal. □

A.2.5 Proposition 5

Proof. (Proposition 5)
Recall that $\sigma$ is an event of the boundary of the language $\overline{L}$ iff $\exists s \in L$ such that $s\sigma \in M - \overline{L}$.

If $L$ is controllable w.r.t. $(M, \Sigma_u)$, then $\forall s \in L$, $\forall \sigma \in \Sigma$ such that $s\sigma \in M$, then $s\sigma \in M - \overline{L}$ means that $\sigma \in \Sigma_c$.

On the other hand, if $(\forall s \in \overline{L})(\forall \sigma \in \Sigma)s\sigma \in M - \overline{L} \Rightarrow \sigma \in \Sigma_c$, then
\[ \Sigma_u \cap M \subseteq \bar{L}, \text{ which means that } L \text{ is controllable w.r.t. } (M, \Sigma_u). \]

### A.2.6 Theorem 3

**Lemma 8.** Let the definitions be the same as in Theorem 3. If \( \Sigma_{b,K} \subseteq \Sigma_o \), then

\[ M \parallel K^{hi} \subseteq K \]

**Proof.** Since \( E \) is a closed language, thus \( P_o(E) \) is closed, hence \( K^{hi} \) is closed. Therefore \[ \bar{P_o^{-1}(K^{hi}) \cap M} = \bar{P_o^{-1}(K^{hi}) \cap M}. \]

With Proposition 2, we have that \( M \parallel K^{hi} \subseteq M \parallel P_o(E) = E \). Now we prove that \( P_o^{-1}(K^{hi}) \cap L \) is controllable w.r.t. \( (M, \Sigma_u) \). Given \( s \in \bar{P_o^{-1}(K^{hi}) \cap L} \), and an event \( \sigma_u \in \Sigma_u \) such that \( s\sigma_u \in M \). Since \( \exists w \) such that \( sw \in P_o^{-1}(K^{hi}) \cap L \), thus \( P_o(sw) \in K^{hi} \), hence \( P_o(s) \in K^{hi} \). If \( \sigma_u \in (\Sigma - \Sigma_o) \), then \( P_o(s\sigma_u) = P_o(s) \in \bar{K^{hi}} \), else if \( \sigma_u \in \Sigma_o \), then with the fact that \( P_o(s)\sigma_u \in P_o(M) \) we have \( P_o(s)\sigma_u \in \bar{K^{hi}} \). Therefore we always have \( P_o(s\sigma_u) \in \bar{K^{hi}} = K^{hi} \), and finally we have \( s\sigma_u \in P_o^{-1}(K^{hi}) \cap M = \bar{P_o^{-1}(K^{hi}) \cap M}. \) Therefore \( P_o^{-1}(K^{hi}) \cap L \) is controllable w.r.t. \( (M, \Sigma_u) \).

The two results imply that \( P_o^{-1}(K^{hi}) \cap M \subseteq K \).

**Lemma 9.** Let the definitions be the same as in Theorem 3. We have that

\[ (\forall s \in \Sigma^+)(s \in K) \iff (\forall t \leq s, \forall u \in \Sigma_u^*, (tu \in M \Rightarrow tu \in E)) \]

**Proof.** We prove the \( \Rightarrow \) direction. With the definition of controllable language, we have that \( s\Sigma_u \cap \bar{L} \subseteq \text{supC}(M, E, \Sigma_u) \subseteq E \). Inductively, \( \forall u \in \Sigma_u^*, \)


we have that \( su \in E \).

Now we prove the \( \Leftarrow \) direction. Suppose that there exists \( s \) such that 
\[
(\forall t \leq s)(\forall u \in \Sigma_u)tu \in M \Rightarrow tu \in E,
\]
then we construct a new language 
\[
E' := \{ s' | \exists t \leq s, \exists u \in \Sigma_u, tu \in L \},
\]
thus \( E' \subseteq E \) and \( E' \Sigma_u \cap L \subseteq E' \), hence 
\( E' \subseteq K \). Therefore \( s \in K \).

**Lemma 10.** Let the definitions be the same as in Theorem 3. If \( \Sigma_{b,K} \subseteq \Sigma_o \), then

\[
P_o^{-1}(K^{hi}) \cap L = K
\]

**Proof.** With lemma 8, we have that \( M \parallel K^{hi} \subseteq K \).

Now we prove the other direction. Since \( P_o(K) \subseteq P_o(E) \), thus if we can prove that \( P_o(K) \) is controllable w.r.t. \( (M, \Sigma_u) \) then we have proved that 
\( P_o(K) \subseteq K^{hi} \), and hence that 
\( P_o^{-1}(K^{hi}) \cap M \supseteq K \). Given a string \( s \in K \), and an uncontrollable event \( \sigma_u \in (\Sigma_u \cap \Sigma_o) \) such that 
\( P_o(s)\sigma_u \in P_o(M) \). Suppose \( P_o(s)\sigma_u \in P_o(L) - K^{hi} \), with lemma 9 we have that \( \exists u \in (\Sigma_u \cap \Sigma_o)^* \) such that 
\( P_o(s)\sigma_u u \in P_o(s)\sigma_u u \in P_o(M) - P_o(E) \). Therefore we can rephrase it as 
\( u = u_1\sigma'_u u_2 \) such that 
\( P_o(s)\sigma_u u_1 \in P_o(E) \land P_o(s)\sigma_u u_1\sigma'_u \in P_o(L) - P_o(E) \), thus \( \sigma'_u \in \Sigma_o \) due to Proposition 2. Since \( P_o \) is an \( M \)-observer, thus \( \exists w \in \Sigma^* \) such that 
\( sw\sigma'_u \in L \land P_o(w) = \sigma_u u_1 \). Since \( s \in K \), thus we can rephrase \( w \) as 
\( w = w_1\sigma_c w_2 \) such that 
\( sw_1 \in K \land sw_1\sigma_c \in M - K \).
We have that \( \sigma_c \in \Sigma_c \), else \( sw_1 \notin K \), thus \( \sigma_c \in \Sigma_{b,K} \), hence \( \sigma_c \in \Sigma_o \), which contradicts that \( P_o(w) \in (\Sigma_u \cap \Sigma_o)^* \). Therefore it’s always true that 
\( P_o(s)\sigma_u \in K^{hi} \). Lemma 8 implies \( \exists w \in E \) such that 
\( w\sigma_u \in K \land P_o(w) = P_o(s) \), hence 
\( P_o(s)\sigma_u = P_o(w\sigma_u) \in P_o(K) \). Therefore \( P_o(K) \) is controllable w.r.t. \( (P_o(M), \Sigma_u \cap \Sigma_o) \).

Combining the two parts above, the proof is done.

**Lemma 11.** Let the definitions be the same as in Theorem 3. If \( \Sigma_{b,K} \not\subseteq \Sigma_o \),
then

\[ P_o^{-1}(K^{hi}) \cap L \subseteq K \]

**Proof.** Lemma 8 says that \( M \parallel K^{hi} \subseteq K \).

Since \( \Sigma_{b,K} \not\subseteq \Sigma_o \), thus there exists a string \( s \), and an event \( \sigma_c \) such that \( s \in K \land s\sigma_c \in (L - K) \), hence \( P_o(s) = P_o(s\sigma_c) \in P_o(L) - K^{hi} \) as \( P_o^{hi}(K^{hi}) \cap L \subseteq K \). Therefore \( s \not\in M \parallel K^{hi} \), which completes the proof.

\[ \square \]

**Proof.** (Theorem 3)
Lemma 10 and 11 complete the proof.

\[ \square \]

### A.2.7 Proposition 6

**Proof.** (Proposition 6)
Lemma 9 proves this proposition.

\[ \square \]

### A.2.8 Lemma 4

**Proof.** (Lemma 4)
Suppose the statement is not true, then \( (\exists s \in K)(\exists \sigma_c \in \Sigma_c)s \in K \land s\sigma_c \in M - K \land \sigma_c \not\in \Sigma_o \). By lemma 9, we have that \( (\exists u \in \Sigma_u^*)s\sigma_c u \in M - E \), thus we can write \( u \) as \( u = u_1\sigma_u u_2 \) such that \( s\sigma_c u_1 \in E \land s\sigma_c u_1\sigma_u \in M - E \).

We have that \( P_o(s)P_o(\sigma_c u_1)\sigma_u = P_o(s)\sigma_u \in P_o(M) \), since \( s \in K \), thus \( \not\exists v \in \Sigma_u^* \) such that \( P_o(v) = P_o(u_1) \land s\sigma_u \sigma_u \in M - E \). Consider the second
part of the statement of lcc, since the image of \( \sigma_c u_1 \) under \( P_o \) extends \( P_o(s) \) to \( P_o(s) \sigma_u \), thus it violates the definition of lcc. Therefore \( \Sigma_{b,K} \subseteq \Sigma_o \). \( \square \)

### A.2.9 Theorem 4

*Proof. (Theorem 4)*

Theorem 3 and Lemma 4 complete the proofs. \( \square \)

### A.2.10 Theorem 5

*Proof. (Theorem 5)*

In the proofs of Theorem 3 and Lemma 4, we only use the properties of natural observer, and lcc to \( \Sigma_{b,E} \), which is the event set of the boundary of \( E \). Since partial natural observer and partial lcc preserve this property, thus the conclusions of Theorem 3 and Lemma 4 can be derived from the conditions of Theorem 5. \( \square \)

### A.2.11 Proposition 7

*Proof. (Proposition 7)*

With Proposition 2, if we can show that \( (\forall s \in M) P_o(s) \in P_o(L) \Rightarrow s \in L \), then the proof is done. Since \( P_o(s) \in L \), thus \( (\exists t \in \Sigma_o^*) P_o(s) t \in P_o(L) \). Furthermore, since \( P_o \) is \( M \)-observer, thus \( (\exists w \in \Sigma^*) sw \in M \land P_o(w) = t \). Proposition 2 implies that \( sw \in L \), thus \( s \in L \). \( \square \)
A.2.12 Proposition 8

Proof. (Proposition 8)
Since $\Sigma_o$ satisfies the maximally permissive control constraint of Problem (3.2), thus $K$ is $(M, P_o)$-normal. Since $P_o$ is $M$-observer, thus Proposition 7 implies that $K$ is $(M, P_o)$-normal, hence the proof is done.

A.2.13 Lemma 5

Proof. (Lemma 5)
We have to show that $\text{supC}(M, K' \cap E, \Sigma_u) = K$. Obviously $\text{supC}(M, K' \cap E, \Sigma_u) \subseteq E$ and is controllable w.r.t. $(M, \Sigma_u)$, thus $\text{supC}(M, K' \cap E, \Sigma_u) \subseteq K$.

Now we deal with the other direction. Obviously $K \subseteq \text{supC}(M, E, \Sigma_u) = K'$, thus $K \subseteq (K' \cap E)$. Furthermore, $K$ is controllable w.r.t. $(M, \Sigma_u)$. Therefore we have that $\text{supC}(M, E \cap K', \Sigma_u) \supseteq N_E$.

Combining the two parts above we have that $\text{supC}(L, K' \cap E, \Sigma_u) = K$.

A.2.14 Lemma 6

Proof. (Lemma 6)
Theorem 6 says that $K'$ is $(M, P_o)$-normal. Furthermore, $E$ is $(M, P_o)$-normal. Let $s \in M, P_o(s) \in P_o(K' \cap E)$. Proposition 2 says that $s \in K' \land s \in E$, thus $s \in K' \cap E$, hence $K' \cap E$ is $(M, P_o)$-normal. Since $P_o$ is an $M$-observer, thus Proposition 7 implies that $E \cap K'$ is also $(M, P_o)$-normal.
A.2.15 Theorem 6

Proof. (Theorem 6)
We use the induction to prove that $E^i, \overline{E^i}, i \in \mathcal{N}$ are $(M, P_o)$-normal. Theorem 6 proves $i = 1$ case. Suppose it’s true for $i = m$ that $E^m, \overline{E^m}$ are $(M, P_o)$-normal. Lemma 6 implies that $E^{m+1}, \overline{E^{m+1}}$ are $(M, P_o)$-normal.

Finally Lemma 7 says that the series converge to $K$ in finite steps, thus the proof is done.

A.2.16 Theorem 7

Proof. (Theorem 7)
In the proofs of Theorem 6, we only use the properties of natural observer, and lcc to $\Sigma_b, E$, the events of the boundary of $E$. Since partial natural observer and partial lcc preserve this property, thus the conclusions of Theorem 6 can be derived from the conditions of Theorem 5.

A.2.17 Proposition 9

Proof. (Proposition 9)
Recall that $E = \big||_{i=1}^n K_i, M = \big||_{i=1}^n \overline{K_i}$. Assuming that $\Sigma$ are all controllable, then $E$ is controllable w.r.t. $(M)$ in this scenario. Theorem 8 says that $E, \overline{E}$ are $(M, P_o)$-normal. Therefore the proof is done.
A.2.18 Proposition 10

**Proof.** (Proposition 10)
Let $Z_o : \Sigma_o^* \rightarrow \Sigma_o^*$. Since $(\forall s \in M)(P'_o(s) \in P'_o(E)) \Rightarrow Z_o \circ P'_o(s) \in Z_o \circ P'_o(E) = P_o(E)$, thus $s \in E$ due to Proposition 2. Therefore Proposition 2 implies that $E$ is $(M, P'_o)$-normal.

A.3 Proofs for Section 4.1

A.3.1 Theorem 9

**Proof.** (Theorem 9)

Denote the alphabets of $M_{sub_i}$ as $\Sigma_{sub_i} \subseteq \Sigma$. Let $P_{sub_i} : \Sigma^* \rightarrow \Sigma_{sub_i}^*$. If we can show that $M \parallel K_{sub_i}$ is $(M, Q_{sub_i} \circ P_{sub_i})$-normal, then we have that

\[
M \parallel (\|_{i=1}^n Q_i(K_{sub_i})) = M \parallel (\|_{i=1}^n K_{sub_i}) \\
M \parallel (\|_{i=1}^n \overline{Q_i(K_{sub_i})}) = M \parallel (\|_{i=1}^n \overline{K_{sub_i}})
\]

Therefore $Q_{sub_i}(K_{sub_i})$ can recover $K_{sub_i}$ given $M$.

Now we prove that $K_{sub_i}$ is $(M, Q_{sub_i} \circ P_{sub_i})$-normal. Proposition 9 implies that $M \parallel K_{sub_i}, M \parallel M_{sub_i}$ are $(M, P_{sub_i})$-normal. Let $s \in M$. If $Q_{sub_i}(P_{sub_i}(s)) \in Q_{sub_i}(K_{sub_i})$, then $P_{sub_i}(s) \in K_{sub_i}$ due to Proposition 2. Similarly $s \in M \parallel K_{sub_i}$, due to Proposition 2. Therefore $K_{sub_i}$ is $(M, Q_{sub_i} \circ P_{sub_i})$-normal.