Supervisor Localization: A Top-Down Approach to Distributed Control of Discrete-Event Systems

by

Cai, Kai

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science and Engineering
Graduate Department of Electrical & Computer Engineering
University of Toronto

2008.06.25

Abstract

Supervisor Localization: A Top-Down Approach to Distributed Control of Discrete-Event Systems

Cai, Kai

Master of Applied Science and Engineering

Graduate Department of Electrical & Computer Engineering

University of Toronto

2008

In this thesis we study the design of distributed control for discrete-event systems (DES) in the framework of supervisory control theory. Our DES comprise a group of agents, acting independently except for specifications on 'global' (group) behavior. The central problem investigated is how to synthesize 'local' controllers for individual agents such that the resultant controlled behavior is identical with that achieved by global supervision.

The investigation is carried out with both language- and state-based models. In the language-based setting, a supervisor localization algorithm is developed that solves the problem in a top-down fashion: first, compute a global supervisor, then decompose it into local controllers while preserving the global controlled behavior. In the case of large-scale DES where a global supervisor might not be feasibly computable owing to state explosion, a decomposition-aggregation solution procedure is developed. In the state-based setting, specifically that of 'state tree structures' (STS), a counterpart supervisor localization algorithm is obtained having potential to exploit the known efficiency of STS for large-DES control design.

Contents

List of Tables

List of Figures

List of Symbols

$\bar{L}$ (prefix) closure of language L, page 13

$\mathcal{B}(\mathbf{S}T)$ Family of all basic state trees of state tree $\mathbf{S}T$, page 98

Δ global transition function of STS, page 100

Γ backward global transition function of STS, page 100

$\hat{f}_\sigma$ extended control function for event σ (or extended local decision maker), page 111

$\mathcal{C}^k$ control cover with respect to agent $\mathbf{LOC}^k$, page 18

$\mathcal{R}^k$ control consistency binary relation with respect to agent $\mathbf{LOC}^k$, page 17

$\mathcal{ST}(\mathbf{S}T)$ Family of all sub-state-trees of state tree $\mathbf{S}T$, page 97

$\mathbf{J}^k$ induced generator with respect to agent $\mathbf{LOC}^k$, page 18

$\mathbf{LOC}^k$ local controller of agent $\mathbf{G}^k$, page 14

SUP monolithic optimal nonblocking supervisor, page 14

$\mathbf{S}T$ state tree, page 96

$\mathbf{S}T^y$ child state tree rooted at y, page 96

Θ encoding function for sub-state-trees, page 101

B_C^k local state tracker, page 112

f_σ control function for event σ, page 104

H holon, page 98

$L(\mathbf{G})$ closed behavior of generator $\mathbf{G}$, page 13

$L_m(\mathbf{G})$ marked behavior of generator $\mathbf{G}$, page 13

PD_σ preliminary disablement predicate for event σ, page 107

$Pwr(\Sigma)$ Power set of set Σ, page 16

$sup\,\mathcal{C}(F)$ supremal controllable sublanguage of language F, page 14

$sup\,\mathcal{C}^2\mathcal{P}(Q)$ supremal weakly controllable and coreachable subpredicate of predicate Q, page 105

$sup\,\mathcal{QC}(Y)$ supremal quasi-congruence of set Y, page 46

BDD binary decision diagram, page 94

DASLP decomposition-aggregation supervisor localization procedure, page 59

DES discrete-event systems, page 3

LA localization algorithm, page 23

OCC output control consistency, page 51

SCT supervisory control theory, page 4

SFBC state feedback control, page 104

SLA symbolic localization algorithm, page 121

STS state tree structure, page 93

Chapter 1

Introduction

1.1 Motivation and Background

Rapid advances in communication networks and embedded computing technologies have made *distributed systems* pervasive in engineering practice. By these are meant systems that consist of multiple interconnected agents locally interacting in pursuit of a global goal. Instances of distributed systems abound: multi-robot search teams explore terrain, ocean, and even space; wireless sensor networks support military reconnaissance and surveillance; and automated guided vehicles transport material in a manufacturing workcell or serve a loading dock. The evolving relevance of distributed systems has attracted researchers from diverse scientific disciplines, leading them to propose underlying mechanisms effective in governing this type of system. Particular attention has been focused on designing individual 'built-in' strategies (as opposed to devising external supervisors), subject to interagent coupling constraints, to arrive at prescribed collective behavior. This is usually referred to as *distributed control*. It has been significantly motivated by sociobiological studies on aggregate actions of animal groups.

> Flocks [of birds] and related synchronized group behaviors such as schools of fish or herds of land animals are both beautiful to watch and intriguing

> to contemplate. A flock [of birds] exhibits many contrasts. It is made up of discrete birds yet overall motion seems fluid; it is simple in concept yet is so visually complex; it seems randomly arrayed and yet is magnificently synchronized. Perhaps most puzzling is the strong impression of intentional, centralized control. Yet all evidence indicates that flock motion must be merely the aggregate result of the actions of individual animals, each acting solely on the basis of its own local perception of the world [45].

Stimulated by the above biological observation, Reynolds [45] wrote *boids*, a celebrated program that successfully simulates a flock of birds in flight, each navigating according to a handful of local rules. Following Reynolds numerous computer animations have been created, reproducing various natural grouping phenomena. In one remarkable example [19], McCool reports a simulation of a barnyard where 16,000 virtual chickens move towards a rooster; the collective locomotion again emerges from the individual maneuvers of these faux fowl.

A considerable body of biologically inspired research has also been reported in the artificial intelligence and robotics communities, with the hope of endowing robotic systems with desirable behavioral traits such as those observed in biological systems. Two seminal studies – Brooks' subsumption architecture [8] and Arkin's motor schema [3] – introduced the prominent behavior-based paradigm, which in turn initiated a large number of investigations in *intelligent agents* [1, 41] and *cooperative robotics* [2, 10]. The behavior-based approach seems so promising that multi-robot teams have been increasingly applied to a variety of engineering domains. However, the corresponding research results are typically presented via dogmatic description and illustrated with heuristic simulation and experiment; namely the approach "is thin when it comes to providing controllers with guarantees; and engineers and theorists want guarantees [4]". This gap has generated interest in the rigorous mathematical treatment of distributed control design, so that the derived results can be formally justified for validity and completeness.

Systems control theorists have investigated distributed systems with continuous- and discrete-time dynamics (i.e., with system states evolving as a function of time). A collection of component agents is considered as the plant to be controlled, and the desired global behavior as the specification; the control objective is to synthesize local strategies for individual agents so as to enforce the specification. A crucial feature of this distributed control design is the 'disturbance' due to possibly dynamic interagent coupling, an issue which gives rise to an extensive exploration and application of mathematical graph theory [33]. So far, rigorous results have been established for collective behaviors as diverse as rendezvous (all agents congregate at the same location) [25, 34], coverage (all agents spread out to cover a space uniformly) [13], and formation (agents cooperatively maneuver to form a geometric shape such as a line, a circle, and even various polygons) [35, 40].

A rigorous approach is taken as well by researchers in distributed computing. Here the distributed system of concern is typically a collection of interleaving sequential programs, an instance of *discrete-event systems* (DES) [1]. These programs are regarded as executing in parallel on multiple processors, while now and then communicating with one another by passing messages. Desired properties of synchronized global behavior usually include safety (e.g., mutual exclusion of a non-sharable resource), liveness (or absence of starvation), and deadlock-freeness [6]. An approach commonly taken is first to design, with respect to specified global properties, communication protocols for individual programs, and then to employ mathematical tools – notably I/O automata [36] and temporal logics [38] – to prove formally whether or not the specifications are satisfied; if not, the two steps above are repeated. In that sense, this approach is, however, a trial-and-error process that could involve considerable heuristic effort in design, as contrasted with systematic

[1] A DES is a dynamic system that is equipped with a discrete state space and an event-driven transition structure (i.e., state transitions are triggered by occurrences of events other than, or in addition to, the tick of a clock) [63]. DES can be thought of as an abstraction of continuous- or discrete-time systems, an abstraction that allows reasoning about a system's logical behavior without reference to time-driven quantitative detail.

synthesis from a systems control perspective.

This thesis studies distributed control design for general DES (in contrast to the specific instance studied in distributed computing) in the framework of *supervisory control theory* (SCT) [2]. SCT models a DES as the generator of a formal language, and "supports the formulation of various control problems of standard types, like the synthesis of controlled dynamic invariants by state feedback, and the resolution of such problems in terms of naturally definable control-theoretic concepts and properties, like reachability, controllability and observability" [63]. Having adopted SCT as the framework of choice, the thesis undertakes a control-theoretic approach, the principal objective being local strategy synthesis. Namely, given a set of coupled DES as the plant and some desired global property as the specification, one aims to design an algorithm that automatically generates local controllers for each of the plant components that participate in the specification.

1.2 System Architectures in SCT

A system architecture is a mode of organization of collective actions of both plant components and their controllers. In this section a survey is given of system architectures that have been studied in SCT, with the aim of placing the author's work on distributed control in perspective in the SCT literature. While by no means exhaustive, our survey does cite key references in each of the categories of architecture identified.

1.2.1 Monolithic Architecture

SCT was initiated by Ramadge and Wonham [43, 64], with cornerstone results of the field established for a monolithic architecture, an organization where all plant components are controlled by a single centralized supervisor. With the supervisor, the controlled behavior

[2] See [11, 63] for textbook treatment, and [44, 54] for surveys.

can be made 'optimal' (i.e., minimally restrictive) with respect to imposed specifications, typically safety and nonblockingness [3]. This architecture is extended by constraining the scope of the supervisor's observation; partial event observation [31] and partial state observation [29] are investigated separately.

Although monolithic architecture plays a fundamental conceptual role, the complex transition structure of the centralized supervisor typically renders it prohibitively difficult for a human designer to grasp the overall control logic. In other words, the synthesized control typically lacks transparency. The situation is made worse by the fact, proved by Gohari and Wonham [18], that global supervisor synthesis is NP-hard, inasmuch as the state space size grows exponentially in the number of individual plant components and specifications.

1.2.2 Modular Architecture

Stimulated by the twin goals of improving understandability of control logic and reducing computational effort, subsequent literature has witnessed the emergence of a variety of alternative modular system architectures – decentralized architecture, hierarchical architecture, and heterarchical architecture.

1. Decentralization can be viewed as horizontal modularity, wherein the overall synthesis task is decomposed into several smaller-scale subtasks. In a decentralized architecture, specialized individual supervisors, each synthesized to perform a particular subtask, operate in parallel, each observing and controlling only the relevant subset of plant components. Early work in this vein includes [30, 32, 65], where it was assumed that the global specification is decomposable (as a 'shuffle' product) into independent local subspecifications. This assumption was later dropped in

[3] Safety and nonblockingness in the SCT context have a broader meaning than their counterparts in distributed computing: safety refers to the avoidance of prohibited regions of state space; while nonblockingness means that distinguished target states always remain reachable [63].

[12, 48, 58, 66], where a decentralized solution was sought to a possibly indecomposable specification. More recently, this architecture was extended by permitting communications among decentralized supervisors, which thus may cooperatively resolve ambiguity due to 'myopic' local observation [5, 46, 47, 55, 59].

2. Hierarchy, by contrast, can be viewed as vertical modularity that constructs a high-level aggregate model of the underlying system dynamics, separating control synthesis into 'low-level' and 'high-level' design procedures. Here high-level design is based on an aggregate model of the underlying system dynamics, constructed by abstracting out information irrelevant to high-level control objectives. Thus in a purely hierarchical architecture, all plant components are placed under the control of a single hierarchical supervisor that runs at a higher level of temporal and/or logical abstraction. This architecture is studied in [7, 67], and a general hierarchical control theory emerges in [60].

3. Both horizontal decomposition and vertical aggregation can be effective approaches to handling complexity. Combining them gives rise to a *heterarchical* architecture, wherein all plant components are controlled by a hierarchy of decentralized supervisors. Research on this architecture is currently very active; notable results are reported in [16, 23, 28, 49, 61].

1.2.3 Purely Distributed Architecture

The defining characteristic of the preceding architectures is a 'supervisor-subordinate' paradigm: a monolithic supervisor, or an organization of modular supervisors, monitors the behavior of subordinate agents and makes all decisions on their behalf, while the controlled agents themselves act 'blindly' based on the commands they receive. Intuitively one could think of these supervisors as 'external to', rather than 'built into', the subordinate agents. From this perspective, monolithic and modular architectures are not, in our

view, properly considered to be purely distributed. We adopt the view that distributed architecture is a 'flat' system organization where global functions emerge through the collective actions of individual agents and are not, at least directly, guided by higher-level, external supervisors [22].

What are the possible advantages of a purely distributed organization over a supervisor-subordinate one? One important motivator is the goal of higher system reliability (or greater robustness). When control functions are distributed over many agents, an individual agent failure, while it may admittedly degrade performance, may be much less likely to bring other agents down; on the other hand, a supervisor malfunction is quite likely to incapacitate all its associated agents, possibly wreaking major havoc on system performance. Another potential benefit of distributed architecture is easier maintainability (or in a narrower sense, scalability). Many systems have to be modified in order to cope with changes in the dynamic environment in which they function, and/or changes in the tasks they undertake. For example, new functions have to be added to a bank's computer system if and when it offers a new line of business. In a supervisor-subordinate architecture, such changes are likely to entail major redesign, whereas with a distributed organization, system modifications could more likely be confined to the agents directly affected, leaving the rest intact.

In the present state of knowledge, these benefits are mainly envisaged intuitively. Their rigorous validation would require a quantitative analysis of the tradeoffs involved in each particular case. While such analysis falls outside the scope of this thesis, its potential interest does provide underlying motivation for our research on the structural issues for DES explored in the thesis. In any case, it will turn out that these latter issues have (in the author's view) ample intrinsic interest in themselves.

In the SCT literature, the term "distributed architecture" along with "distributed control" and "agent" have been used by other authors. Lafortune argues [27] that "...one limitation of decentralized architecture is the fact that they do not allow real-time com-

munication among supervisors. Allowing real-time communication could greatly enhance the classes of specifications that can be achieved under control...such *distributed architectures* could be costly in terms of communication required." He further formulates the distributed control problem: "Consider a distributed networked DES modeled by automaton G. There are n agents observing the behavior of G using their own sets of sensors. The agents may be supervisors or diagnosers. The agents are able to communicate among each other...". A similar argument and formulation are also found in [20, 24, 26], and some related results are implemented using distributed extended finite state machines [39]. In these authors' usage, "distributed architecture" actually refers to decentralized architecture with communicating modular supervisors (or diagnosers) as mentioned in Section 1.2.2, while the local strategy design for individual subordinate agents is not an explicit objective.

To fill this gap, the thesis initiates the study of distributed control design for DES, thereby adding a new 'dimension' – the purely distributed architecture – to the 'space' of system architectures in SCT.

1.3 Outline of Thesis and Contribution

This thesis investigates distributed control design for DES in the SCT framework. The objective is local controller synthesis. Concretely, given a DES plant comprised of interconnected components, and control specifications constraining collective behavior, the thesis proposes a systematic design procedure for individual local controllers that as a whole satisfy the imposed specifications. The rest of the thesis with its primary contributions is in outline as follows.

Chapter 2 begins by assuming that the optimal nonblocking monolithic supervisor for a given control problem exists and can be computed. Given this assumption, we formulate the distributed control problem as that of decomposing the monolithic supervisor into

local controllers for individual plant components, while preserving the optimality and nonblocking properties of the monolithic solution. We solve this problem by developing a *supervisor localization algorithm*, which is then implemented by a computer program and validated by examples familiar in the literature. Further, two boundary cases of the localization algorithm are identified which indicate, as a property of the localization problem itself, an extreme degree of 'easiness' or 'hardness', respectively.

Contribution of Chapter 2: To our knowledge, the formulated distributed control problem is original in the SCT literature, and the proposed supervisor localization algorithm is the first solution to this problem.

Chapter 2 leaves open the question of how to solve the formulated distributed control problem for large-scale DES. In that case it is often not feasible to compute the monolithic supervisor owing to state space explosion. In Chapter 3, we address this question by proposing a *decomposition-aggregation procedure* that in essence combines supervisor localization with modular supervisory control theory in the following manner: first, design an organization of modular supervisors that achieves optimal nonblocking control, then decompose each of these modular supervisors into local controllers for the relevant components. This procedure is then applied to solve two medium-sized but nontrivial industrial examples.

Contribution of Chapter 3: The established decomposition-aggregation procedure is the first, and usually efficient, solution to the distributed control problem for large-scale DES.

The modelling setup of the above two chapters is language-based. By contrast, Chapter 4 turns to a state-based setting, specifically the state tree structure, and studies distributed control design therein. The counterpart distributed control problem is formulated, and the counterpart supervisor localization algorithm developed. The algorithm is implemented by a computer program and validated by a familiar example.

Contribution of Chapter 4: The formulation of the distributed control problem in

the state tree structure is original, and the state-based supervisor localization algorithm is the first solution to this problem.

Finally, in Chapter 5 we conclude the thesis and propose some future research topics.

Chapter 2

On Supervisor Localization

2.1 Introduction

This chapter takes the first and fundamental step towards distributed control design for DES. Special attention is given to the class of DES consisting of independent components whose coupling is due solely to specifications via shared events. With the goal of local strategy synthesis in mind, we address the following question: *given a control problem, what should each individual do (in terms of sensing and decision making) so as to satisfy the control objective, and realize performance identical to that achieved by monolithic or modular control?*

Consider an idealized scenario of motorists at an intersection. Monolithic supervisory control could be the case where the intersection is controlled by traffic lights which, we assume, are strictly respected by every motorist. It is the lights that 'command' motorists whether to stop or to move. On the other hand, distributed control is needed if and when the lights happen to fail. Since no motorist is a supervisor of others, each has to be alert in order to survive. The above generic question for this specific problem is: how should individual motorists behave to cross the intersection safely, while maintaining traffic flow just as well as when the lights are operating? Common sense suggests that each motorist

must keep an eye on his immediate neighbors and respond accordingly. We shall pose this kind of problem as one of "mutual exclusion", or generally "resource allocation", and present a formal solution.

We address this issue in the standard framework of SCT. To design local strategies for each component agent, we propose a top-down approach: first build an external (monolithic or modular) optimal nonblocking supervisor through synthesizing the supremal controllable sublanguage of the given specification language; then develop a localization procedure which systematically decomposes the external supervisor to local controllers for individual agents. We call this procedure *supervisor localization*, as displayed in Fig.2.1.

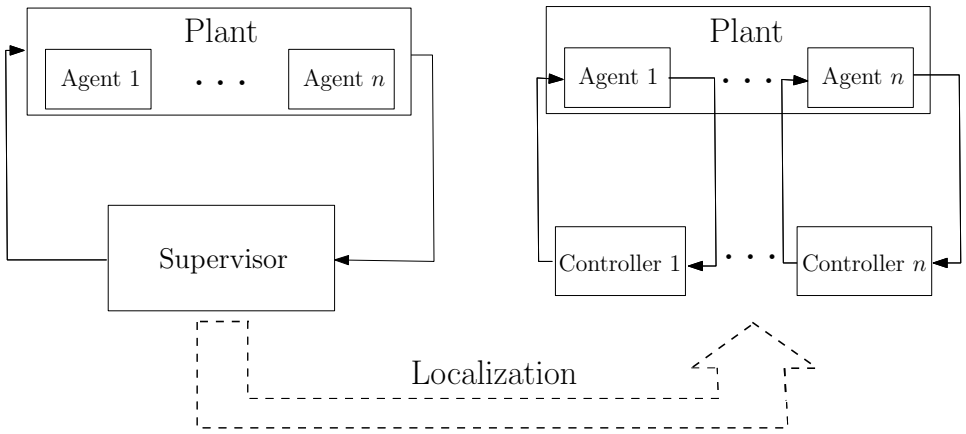


Figure 2.1: Supervisor localization

The goal of supervisor localization is first of all to preserve the optimality and nonblocking property of the external supervisor, namely realizing performance identical to that achieved by monolithic or modular control.

It is also desired that each localized controller be as 'simple' as possible, so that individual strategies are more readily comprehensible. Among diverse criteria of 'simplicity', we focus on the state size. This goal is achieved by a straightforward extension of *supervisor reduction* [56, 53], of which the essence is to 'project' the plant model out of the supervisor model while preserving the controlled behavior. To localize an external

supervisor to a local controller for an individual agent, we carry the reduction idea one step further: in addition to projecting the plant model out of the supervisor, we also project out the transition constraints enforced by other agents. Namely, the localization procedure is conducted based solely on control information directly relevant to the target agent; we proceed this way for each agent in the plant, taken individually. The result is that each agent acquires its own local controller, as displayed in Fig.2.1. Intuitively one could think of these controllers as 'built in', rather than 'external to', the corresponding agents. It will be shown that the local controllers, when running concurrently, achieve the same control behavior as that achieved by the external supervisor.

The rest of this chapter is organized as follows. Section 2.2 formulates the distributed control problem; Section 2.3 presents the development and main results of supervisor localization; Section 2.4 proposes an efficient algorithm for computation; Section 2.5 illustrates supervisor localization with two familiar examples; and Section 2.6 discusses boundary cases of localizability.

2.2 Problem Formulation

The plant to be controlled is modeled, as usual, by a generator

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

where Y is the nonempty state set; $y_0 \in Y$ is the initial state; $Y_m \subseteq Y$ is the set of marker states; Σ is the finite event set, partitioned into the controllable event set Σ_c and the uncontrollable event set Σ_u; $\eta : Y \times \Sigma \to Y$ is the (partial) state transition function. In the usual way, η is extended to $\eta : Y \times \Sigma^* \to Y$ (pfn), and we write $\eta(y, s)!$ to mean that $\eta(y, s)$ is defined. The *closed behavior* of $\mathbf{G}$ is the language

$$L(\mathbf{G}) := \{s \in \Sigma^* | \eta(y_0, s)!\}$$

and the *marked behavior* of $\mathbf{G}$ is thesublanguage

$$L_m(\mathbf{G}) := \{s \in L(\mathbf{G}) | \eta(y_0, s) \in Y_m\} \subseteq L(\mathbf{G})$$

For a language $L \subseteq \Sigma^*$, the (prefix) closure of L is the language $\bar{L}$ consisting of all prefixes of strings in L:

$$\bar{L} = \{t \in \Sigma^* | t \leq s \text{ for some } s \in L\}$$

We say that $\mathbf{G}$ is *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$.

Consider the case where $\mathbf{G}$ consists of component agents $\mathbf{G}^k$ defined over pairwise disjoint alphabets Σ^k ($k \in K, K$ an index set):

$$\Sigma = \dot{\bigcup} \{\Sigma^k | k \in K\}$$

With $\Sigma = \Sigma_c \,\dot{\cup}\, \Sigma_u$ we assign control structure to each agent:

$$\Sigma_c^k = \Sigma^k \cap \Sigma_c \ , \ \Sigma_u^k = \Sigma^k \cap \Sigma_u$$

Let $k \in K$. We say $\mathbf{LOC}^k$ (over Σ) is a *local controller* for agent $\mathbf{G}^k$ if $\mathbf{LOC}^k$ can disable only events in Σ_c^k. Precisely, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$, there holds

$$s\sigma \in L(\mathbf{G}) \ \& \ s \in L(\mathbf{LOC}^k) \ \& \ s\sigma \notin L(\mathbf{LOC}^k) \Rightarrow \sigma \in \Sigma_c^k$$

The observation scope of $\mathbf{LOC}^k$ is, however, neither confined within Σ^k nor fixed beforehand. Indeed, it will be systematically determined to guarantee the correct local control. Thus, while a local controller's control authority is strictly local, its observation scope need not, and generally will not, be.

Let $F \subseteq \Sigma^*$, and recall that F is *controllable* (with respect to $\mathbf{G}$) if

$$\overline{F}\Sigma_u \cap L(\mathbf{G}) \subseteq \overline{F}$$

If F is not controllable, we denote by $\mathcal{C}(F)$ the set of all controllable sublanguages of F. $\mathcal{C}(F)$ is nonempty because $\emptyset$, the empty language, is trivially controllable and hence always belongs to $\mathcal{C}(F)$. Further, $\mathcal{C}(F)$ contains a (unique) supremal element, denoted $sup\mathcal{C}(F)$ [64].

The independent components of the plant are implicitly coupled through an imposed specification language $E \subseteq \Sigma^*$ that (as usual) imposes a behavioral constraint on $\mathbf{G}$. Let $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$ be a generator that represents the language $sup\mathcal{C}(E \cap L_m(\mathbf{G}))$. $\mathbf{SUP}$ is the monolithic optimal nonblocking supervisor for $\mathbf{G}$ (with respect to E).

Now we formulate the *Distributed Optimal Nonblocking Control Problem* ($\divideontimes$) :

> Given $\mathbf{G}$ and $\mathbf{SUP}$ described above, construct a set of local controllers $\mathbf{LOC} = \{\mathbf{LOC}^k | k \in K\}$, one for each agent, with $L(\mathbf{LOC}) = \bigcap\{L(\mathbf{LOC}^k) | k \in K\}$ and $L_m(\mathbf{LOC}) = \bigcap\{L_m(\mathbf{LOC}^k) | k \in K\}$, such that the following two properties hold:
>
> $$L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP}) \qquad (1a)$$
>
> $$L_m(\mathbf{G}) \cap L_m(\mathbf{LOC}) = L_m(\mathbf{SUP}) \qquad (1b)$$
>
> We say that $\mathbf{LOC}$, satisfying $(1a)$ and $(1b)$, is *control equivalent* to $\mathbf{SUP}$ with respect to $\mathbf{G}$.

For the sake of easy implementation and transparent comprehensibility, it would be desired in practice that the state sizes of local supervisors be very much less than that

of their 'parent' monolithic supervisor:

$$(\forall k \in K)\ |\mathbf{LOC}^k| \ll |\mathbf{SUP}|$$

where $|\cdot|$ denotes the state size of the argument. Inasmuch as this property is neither precise nor always achievable, it must needs be omitted from the formal problem statement; in applications, nevertheless, it should be kept in mind.

2.3 Supervisor Localization

We solve (⋇) by developing a supervisor localization procedure, essentially a straightforward extension of supervisor reduction [56, 53].

It follows from $\Sigma = \dot{\bigcup}\{\Sigma^k | k \in K\}$ that the set $\{\Sigma_c^k \subseteq \Sigma_c | k \in K\}$ forms a partition on Σ_c. Fix an element $k \in K$. Following [53], we first establish a *control cover* on X, the state space of **SUP**, based only on control information pertaining to Σ_c^k, as captured by the following four functions. First define $E : X \to Pwr(\Sigma)$ according to

$$E(x) = \{\sigma \in \Sigma | \xi(x, \sigma)!\}$$

Thus $E(x)$ denotes the set of events that are enabled at x. Next define $D^k : X \to Pwr(\Sigma_c^k)$ according to

$$D^k(x) = \{\sigma \in \Sigma_c^k | \neg \xi(x, \sigma)!\ \&\ (\exists s \in \Sigma^*)[\xi(x_0, s) = x\ \&\ \eta(y_0, s\sigma)!]\}$$

So $D^k(x)$ is the set of controllable events in Σ_c^k that must be disabled at x. Notice that if $\sigma \in \Sigma_c^k$ is not in $D^k(x)$, then either $\sigma \in E(x)$ or σ was not defined at any state in Y that corresponds to x. Define $M : X \to \{1, 0\}$ according to

$$M(x) = 1 \text{ iff } x \in X_m$$

Thus M is a predicate on X that determines if a state is marked in **SUP**. Finally define $T : X \to \{1, 0\}$ according to

$$T(x) = 1 \text{ iff } (\exists s \in \Sigma^*)\xi(x_0, s) = x \;\&\; \eta(y_0, s) \in Y_m$$

So T is a predicate on X that determines if some corresponding state is marked in **G**. Note that for each $x \in X$, we have by $(1b)$ that $T(x) = 1 \Rightarrow M(x) = 1$ and $M(x) = 0 \Rightarrow T(x) = 0$.

Definition 2.1.

We define a binary relation $\mathcal{R}^k$ on X as follows. Let $x, x' \in X$. We say x and x' are *control consistent* (with respect to Σ_c^k) (cf [53]), and write $(x, x') \in \mathcal{R}^k$, if

(i) $E(x) \cap D^k(x') = \emptyset = E(x') \cap D^k(x)$

(ii) $T(x) = T(x') \Rightarrow M(x) = M(x')$ ◊

Informally, a pair of states (x, x') is in $\mathcal{R}^k$ if (i) there is no event in Σ_c^k that is enabled at x but is disabled at x', or vice versa (consistent disablement information); and (ii) x and x' are both marked or unmarked in **SUP** provided that they were both marked or unmarked in **G** (consistent marking information). In addition, it should be noted that $\mathcal{R}^k$ is a tolerance relation on X, namely it is reflexive and symmetric but in general need not be transitive.

Example 2.1.

As shown in Fig.2.2, $(x_0, x_1) \in R^k$, for $E(x_0) \cap D^k(x_1) = \emptyset = E(x_1) \cap D^k(x_0)$ and $T(x_0) \neq T(x_1)$. Also, $(x_1, x_2) \in R^k$, for $E(x_1) \cap D^k(x_2) = \emptyset = E(x_2) \cap D^k(x_1)$ and $T(x_0) \neq T(x_1)$. But $(x_0, x_2) \notin R^k$, for $E(x_2) \cap D^k(x_0) \neq \emptyset$ and $T(x_0) = T(x_2)$ & $M(x_0) \neq M(x_1)$. ♦

Thus, in general $\mathcal{R}^k$ need not be an equivalence relation. This fact leads to the following definition of control cover (with respect to Σ_c^k.) First recall that a cover on a

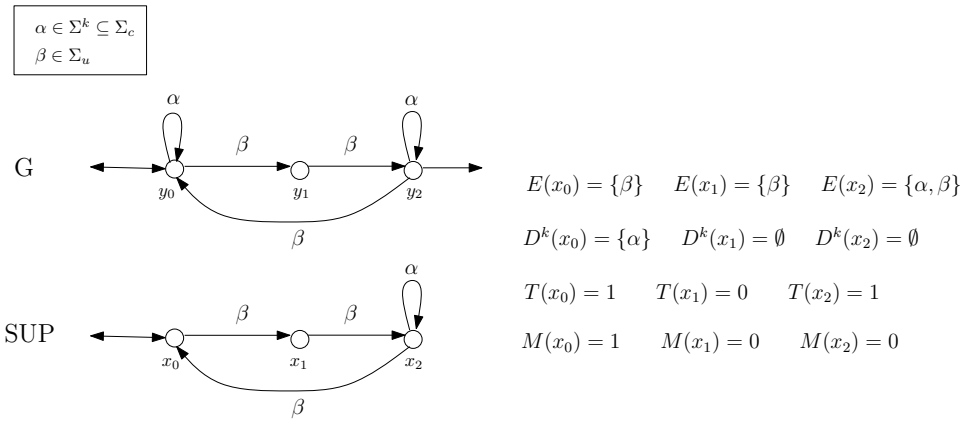


Figure 2.2: Example: $\mathcal{R}^k$ is not transitive

set X is a collection of nonempty subsets of X whose union is X. Precisely, a collection $\{X_i | i \in I\}$ is a *cover* on X if

(i) $(\forall i \in I)\ X_i \neq \emptyset$

(ii) $(\forall i \in I)\ X_i \subseteq X$

(iii) $\bigcup\{X_i | i \in I\} = X$

Definition 2.2.

Let I^k be some index set, and $\mathcal{C}^k = \{X^k_{i^k} \subseteq X | i^k \in I^k\}$ be a cover on X. $\mathcal{C}^k$ is a *control cover* (cf [53]) on X (with respect to Σ^k_c) if

(i) $(\forall i^k \in I^k)(\forall x, x' \in X^k_{i^k})\ (x, x') \in \mathcal{R}^k$

(ii) $(\forall i^k \in I^k, \forall \sigma \in \Sigma)[(\exists j^k \in I^k)(\forall x \in X^k_{i^k})\xi(x, \sigma)! \Rightarrow \xi(x, \sigma) \in X^k_{j^k}]$ $\Diamond$

Informally, a control cover $\mathcal{C}^k$ groups states of **SUP** into (possibly overlapping) cells $X^k_{i^k}, i^k \in I^k$. According to (i) all states that reside in a cell $X^k_{i^k}$ have to be pairwise control consistent; and (ii) for each event $\sigma \in \Sigma$, all states that can be reached from any state in $X^k_{i^k}$ by a one-step transition σ have to be covered by a certain cell $X^k_{j^k}$. Recursively, two states x, x' belong to a common cell in $\mathcal{C}^k$ if and only if (1) x and x' are control consistent; and (2) two future states that can be reached from x and x', respectively, by

the same string are again control consistent. In addition we say that a control cover $\mathcal{C}^k$ is a *control congruence* if $\mathcal{C}^k$ happens to be a partition on X.

Having established a control cover $\mathcal{C}^k$ on X based only on the control information of Σ_c^k, we can then always obtain an induced generator $\mathbf{J}^k = (I^k, \Sigma, \kappa^k, i_0^k, I_m^k)$ by the following construction (cf [53]):

(i) $i_0^k \in I^k$ such that $x_0 \in X_{i_0^k}^k$

(ii) $I_m^k = \{i^k \in I^k | X_{i^k}^k \cap X_m \neq \emptyset\}$

(iii) $\kappa^k : I^k \times \Sigma \to I^k$ (pfn) with $\kappa^k(i^k, \sigma) = j^k$ if
$(\exists x \in X_{i^k}^k)\xi(x, \sigma) \in X_{j^k}^k$ & $(\forall x' \in X_{i^k}^k)[\xi(x', \sigma)! \Rightarrow \xi(x', \sigma) \in X_{j^k}^k]$

Note that, owing to overlapping, the choices of i_0^k and κ^k may not be unique, and consequently $\mathbf{J}^k$ may not be unique. In that case we pick an arbitrary instance of $\mathbf{J}^k$. Clearly if $\mathcal{C}^k$ happens to be a control congruence, then $\mathbf{J}^k$ is unique.

Let $\mathbf{J} := \{\mathbf{J}^k | k \in K\}$ be a set of induced generators for the partition $\{\Sigma_c^k \subseteq \Sigma_c | k \in K\}$, and define $L(\mathbf{J}) := \bigcap\{L(\mathbf{J}^k) | k \in K\}$ and $L_m(\mathbf{J}) := \bigcap\{L_m(\mathbf{J}^k) | k \in K\}$. Our first result shows that $\mathbf{J}$ is a solution to (※).

Proposition 2.1.

$\mathbf{J}$ is control equivalent to **SUP** with respect to $\mathbf{G}$, i.e.,

$$L(\mathbf{G}) \cap L(\mathbf{J}) = L(\mathbf{SUP})$$

$$L_m(\mathbf{G}) \cap L_m(\mathbf{J}) = L_m(\mathbf{SUP})$$

Proof.

See Appendix. □

Next we investigate if the converse is true: that is, can a set of generators that is control equivalent to **SUP** always be induced from a set of suitable control covers on X?

In response, we bring in the following two definitions.

Definition 2.3.

A generator $\mathbf{LOC} = (Z, \Sigma, \zeta, z_0, Z_m)$ is *normal* (with respect to **SUP**) [53] if

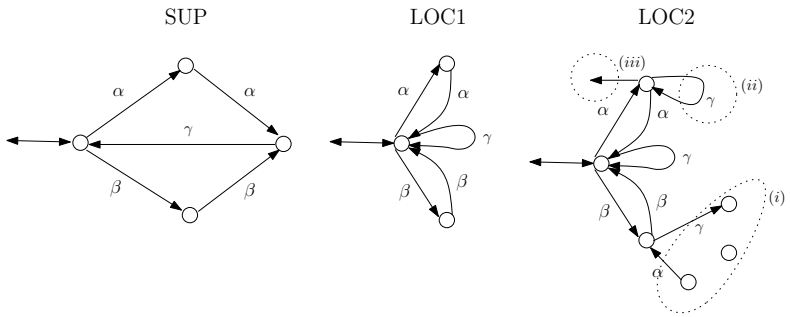
(i) $(\forall z \in Z)(\exists s \in L(\mathbf{SUP}))\ \zeta(z_0, s) = z$

(ii) $(\forall z \in Z, \forall \sigma \in \Sigma)\zeta(z, \sigma)! \Rightarrow (\exists s \in L(\mathbf{SUP}))[\zeta(z_0, s) = z\ \&\ s\sigma \in L(\mathbf{SUP})]$

(iii) $(\forall z \in Z_m)(\exists s \in L_m(\mathbf{SUP}))\ \zeta(z_0, s) = z$ $\Diamond$

Informally, a generator is normal with respect to **SUP** if (1) each of its states is reachable by at least one string in $L(\mathbf{SUP})$; and (2) each of its one-step transitions, say σ, defined at a state that is reached by a string s in $L(\mathbf{SUP})$, preserves membership of $s\sigma$ in $L(\mathbf{SUP})$; and (3) each of its marked states is reachable by at least one string in $L_m(\mathbf{SUP})$.

Example 2.2.



One can verify by inspection that **LOC1** is normal with respect to **SUP**, but **LOC2** is not. Indeed for **LOC2**, (i), (ii), and (iii) respectively display the violation of the three conditions in the definition of normality. ⧫

If a generator $\mathbf{LOC} = (Z, \Sigma, \zeta, z_0, Z_m)$ is not normal (with respect to **SUP**), the following three *normalization* operations will replace it by one that is.

(N1) Delete $z \in Z$, if $(\neg\exists s \in L(\mathbf{SUP}))\ \zeta(z_0, s) = z$.

(N2) Delete $\sigma \in \Sigma$ at $z \in Z$, if $(\forall s \in L(\mathbf{SUP}))\ \zeta(z_0, s) = z \Rightarrow s\sigma \notin L(\mathbf{SUP})$.

(N3) Unmark $z \in Z_m$, if $(\neg\exists s \in L_m(\mathbf{SUP}))\ \zeta(z_0, s) = z$.

It is straightforward to verify that (N1)–(N3) will convert **LOC** into a normal generator. Also note that (N1)–(N3) will not increase the state size of **LOC**. More importantly, the following proposition asserts that (N1)–(N3) preserve control equivalence. From now on we can safely consider localization only for normal generators.

Proposition 2.2.

Let $\mathbf{LOC} := \{\mathbf{LOC}^k = (Z^k, \Sigma, \zeta^k, z_0^k, Z_m^k) | k \in K\}$ be a set of generators that are not normal (with respect to **SUP**). Assume that **LOC** is control equivalent to **SUP** (with respect to **G**). Convert each $\mathbf{LOC}^k$ $(k \in K)$ into a normal generator $\mathbf{NLOC}^k$ by (N1)–(N3). Then $\mathbf{NLOC} := \{\mathbf{NLOC}^k | k \in K\}$ is control equivalent to **SUP** (with respect to **G**).

Proof.

See Appendix. □

Definition 2.4.

Given generators $\mathbf{LOC} = (Z, \Sigma, \zeta, z_0, Z_m)$ and $\mathbf{J} = (I, \Sigma, \kappa, i_o, I_m)$. **LOC** and **J** are *DES-isomorphic* [53] if there exists a *DES-isomorphism* $\theta : Z \to I$ such that

(i) $\theta : Z \to I$ is a bijection

(ii) $\theta(z_0) = i_0$ & $\theta(Z_m) = I_m$

(iii) $(\forall z \in Z, \sigma \in \Sigma)\zeta(z, \sigma)! \Rightarrow [\kappa(\theta(z), \sigma)!$ & $\kappa(\theta(z), \sigma) = \theta(\zeta(z, \sigma))]$

(iv) $(\forall i \in I, \sigma \in \Sigma)\kappa(i, \sigma)! \Rightarrow [(\exists z \in Z)\zeta(z, \sigma)!$ & $\theta(z) = i]$ ◊

Under normality and DES-isomorphism, we have the following result in response to the converse question posed above.

Theorem 2.1.

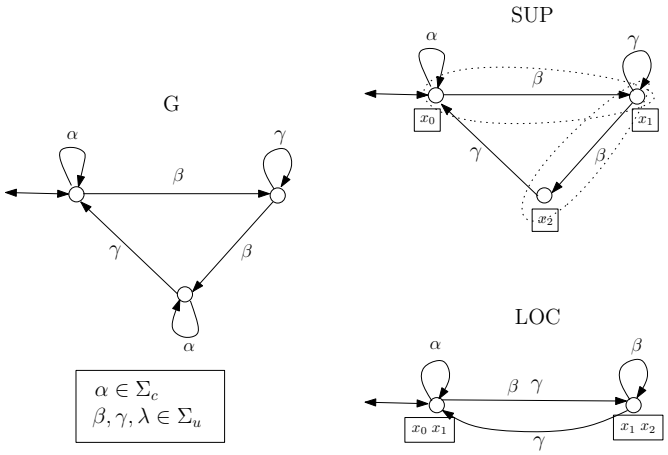
Let $\mathbf{LOC} := \{\mathbf{LOC}^k = (Z^k, \Sigma, \zeta^k, z_0^k, Z_m^k) | k \in K\}$ be a set of normal generators that is control equivalent to **SUP** with respect to **G**. Then there exists a set of control covers $\mathcal{C} := \{\mathcal{C}^k | k \in K\}$ on X with a corresponding set of induced generators $\mathbf{J} := \{\mathbf{J}^k | k \in K\}$ such that $(\forall k \in K)$ $\mathbf{J}^k$ and $\mathbf{LOC}^k$ are DES-isomorphic.

Proof.

See Appendix. □

It is important to notice that Theorem 2.1 is not valid if "control cover" is replaced by "control congruence".

Example 2.3.



The supervisor's control action is to disable event α at state x_2. It is straightforward by inspection that x_0, x_2 are not control consistent, while x_0, x_1 and x_1, x_2 are. But the partition $\{\{x_0, x_1\}, \{x_2\}\}$ is not a control congruence, for $\xi(x_0, \beta) = x_1$ and $\xi(x_1, \beta) = x_2$; neither is the partition $\{\{x_1, x_2\}, \{x_0\}\}$, because $\xi(x_1, \gamma) = x_1$ and $\xi(x_2, \gamma) = x_0$. Therefore, there does not exist a control congruence that can reduce the state size of **SUP**.

On the other hand, the generator **LOC** with two states displayed above can be verified to be control equivalent to **SUP** with respect to **G**. It is induced from the control cover

$\{\{x_0, x_1\}, \{x_1, x_2\}\}$. ♦

To summarize, every set of control covers generates a solution to (⋇) (Proposition 2.1); and every solution to (⋇) can be induced from some set of control covers (Theorem 2.1). In particular, a set of state-minimal normal generators can be induced from a set of suitable control covers. However, such a set is in general not unique, even up to DES-isomorphism. This conclusion accords with that for a state-minimal supervisor in supervisor reduction [53].

2.4 Localization Algorithm

It would be desirable to have an efficient algorithm that always computes a set of state-minimal normal generators, despite its non-uniqueness. Unfortunately, this minimal state problem is NP-hard [53], and consequently we cannot expect a polynomial-time algorithm that can compute a control cover which yields a state-minimal generator.

Nevertheless, a polynomial-time algorithm for supervisor reduction is proposed in [53]. The algorithm generates a control congruence, rather than a control cover, and empirical evidence is given showing that significant state size reduction can often be achieved. Therefore we employ this algorithm, suitably modified to work for supervisor localization, and call the altered version a *localization algorithm* (LA).

We sketch the idea of LA as follows. Given $\mathbf{SUP} = (X, \Sigma_c \dot{\cup} \Sigma_u, \_, \_, \_)$ and $\Sigma_c^k \subseteq \Sigma_c$, LA generates a control congruence $\mathcal{C}^k$ on X with respect to Σ_c^k. LA initializes $\mathcal{C}^k$ to be the singleton partition on X, i.e.,

$$\mathcal{C}^k_{init} = \{[x] \subseteq X | [x] = \{x\}\}$$

where $[x]$ denotes a cell in $\mathcal{C}^k$ to which x belongs. Then LA merges $[x]$ and $[x']$ into one cell if x and x', as well as all their corresponding future states reachable by identical

strings, are control consistent. This *mergibility* condition is checked by lines 14 and 19 in the pseudocode displayed below: line 14 checks control consistency for the current state pair (x, x') and line 19 recursively checks consistency for all their related future states. Throughout, in order to generate a control congruence, it is crucial to prevent states from being shared by more than one cell. This is achieved by inserting in LA three 'filters' – at lines 3, 5, and 18 – to eliminate redundant mergibility tests as well as element overlapping in $\mathcal{C}^k$. LA loops until all of the states are checked.

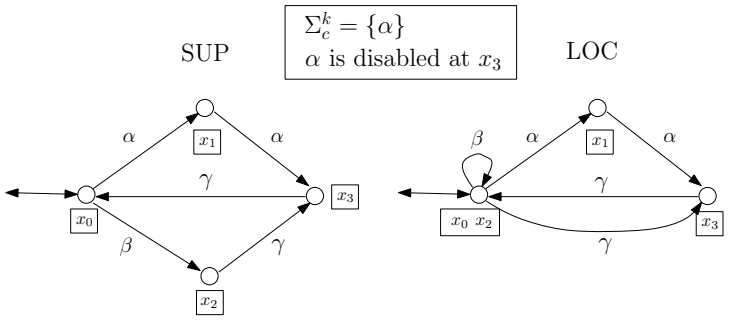
Notation: $X = \{x_0, \ldots, x_{n-1}\}$ is an ordering of states; $wl \subseteq X \times X$ is a list of state pairs whose mergibility is pending. T, F denote *true*, *false* respectively.

```
1  int main()
2  for i : 0 to n − 2 do
3      if i > min{k|x_k ∈ [x_i]} then continue;
4      for j : i + 1 to n − 1 do
5          if j > min{k|x_k ∈ [x_j]} then continue;
6          wl = ∅;
7          if Check_Mergebility(x_i, x_j, wl, x_i) = T then
           C^k = {[x] ∪ ⋃_{x':{(x,x'),(x',x)}∩wl≠∅} [x'] | [x], [x'] ∈ C^k}
8      end
9  end

10 bool Check_Mergibility(x_i, x_j, wl, cnode)
11 for each x_p ∈ [x_i] ∪ ⋃_{x:{(x,x_i),(x_i,x)}∩wl≠∅} [x] do
12     for each x_q ∈ [x_j] ∪ ⋃_{x:{(x,x_j),(x_j,x)}∩wl≠∅} [x] do
13         if {(x_p, x_q), (x_q, x_p)} ∩ wl ≠ ∅ then continue;
14         if (x_p, x_q) ∉ R^k then return F;
15         wl = wl ∪ {(x_p, x_q)};
16         for each σ ∈ Σ with ξ(x_p, σ)!, ξ(x_q, σ)! do
17             if [ξ(x_p, σ)] = [ξ(x_q, σ)] ∨
               {(ξ(x_p, σ), ξ(x_q, σ)), (ξ(x_p, σ), ξ(x_q, σ))} ∩ wl ≠ ∅ then continue;
18             if min{k|x_k ∈ [ξ(x_p, σ)]} < cnode ∨ min{k|x_k ∈ [ξ(x_q, σ)]} < cnode
               then return F;
19             if Check_Mergebility(ξ(x_p, σ), ξ(x_q, σ),
               wl, cnode) = F then return F;
20         end
21     end
22 end
23 return T;
```

Remark 2.1. LA preserves all computational properties of the reduction algorithm in [53]. Namely, LA terminates, generates a control congruence, and has time complexity $O(n^4)$, where n is the state size of **SUP**.

Example 2.4.



This example is a simple illustration of LA.

(0) Initially, $\mathcal{C}_{init}^k = \{[x_0], [x_1], [x_2], [x_3]\}$.

(1) (x_0, x_1) cannot be merged: they pass line 14 because $(x_0, x_1) \in \mathcal{R}^k$, but they fail at line 19 for $(\xi(x_0, \alpha), \xi(x_1, \alpha)) \notin \mathcal{R}^k$; (x_0, x_2) can be merged: they pass line 14 because $(x_0, x_2) \in \mathcal{R}^k$, and they trivially pass line 16 since there is no common event defined on them, so that no further control consistency needs to be verified; (x_0, x_3) cannot be merged: they fail at line 14, for $(x_0, x_3) \notin \mathcal{R}^k$.
So, $\mathcal{C}_1^k = \{[x_0, x_2], [x_1], [x_3]\}$.

(2) (x_1, x_2) cannot be merged: they cannot pass line 5, because x_2 and x_0 are now in the same cell and $2 > 0$; (x_1, x_3) cannot be merged: they failed at line 14, since $(x_1, x_3) \notin \mathcal{R}^k$.
Thus, $\mathcal{C}_2^k = \{[x_0, x_2], [x_1], [x_3]\}$.

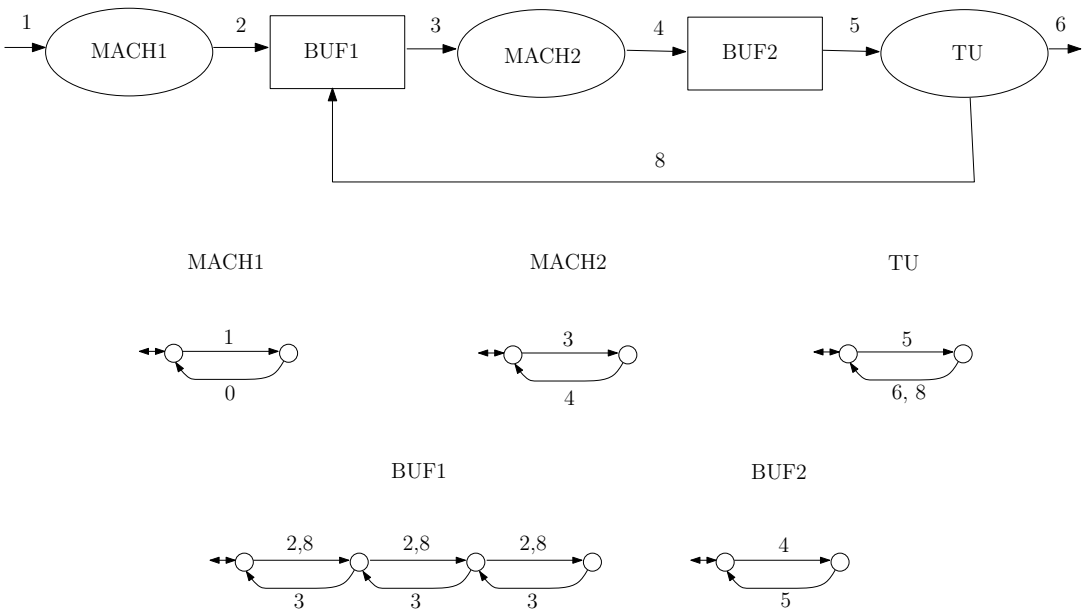
(3) (x_2, x_3) cannot be merged: they failed at line 3 for, again, x_2 and x_0 are now in the same cell and $2 > 0$.
Finally, $\mathcal{C}_3^k = \{[x_0, x_2], [x_1], [x_3]\}$, and the induced generator **LOC** (unique in this case) is displayed above. ♦

2.5 Examples

In this section, we apply the supervisor localization procedure to solve the distributed control problem for two familiar examples. The results are computed by the presented localization algorithm (implemented in a C++ program); the desired control equivalence between the set of local controllers and the optimal nonblocking supervisor is verified in TCT [62], by confirming

$$\mathbf{isomorph}(\mathbf{meet}(\{\mathbf{LOC}^k | k \in K\}, \mathbf{G}), \mathbf{SUP}) = \text{TRUE}$$

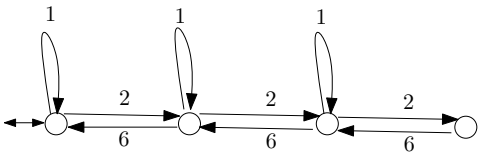
2.5.1 Distributed Control: Transfer Line



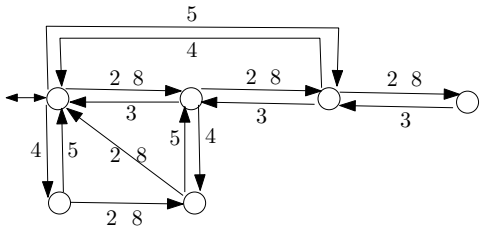
The transfer line system [63, Section 4.6] consists of two machines **MACH1, MACH2** followed by a test unit **TU**; these components are linked by two buffers with capacities of three slots and one slot, respectively. We model **MACH1, MACH2**, and **TU** as the plant to be controlled, and the specification is to protect the two buffers against overflow and underflow. The distributed control objective is to design for each component a local controller – but with no external supervisor.

By using the standard method of SCT, we first build the monolithic optimal non-blocking supervisor, which has 28 states. Then we employ the localization algorithm to compute for each component a local controller from the global supervisor. The resultant controllers are displayed in Fig. 2.3, having 4, 6, and 2 states, respectively.

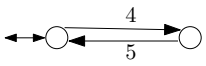
With these individual controllers, we can account for the local strategies of each component. **MACH1**, controlling event 1, ensures that no more than three workpieces can be processed simultaneously in the system, i.e., prevents 'choking' in the 'material feedback' loop; **MACH2**, controlling event 3, guarantees the safety of both buffers in an interleaving manner; and **TU**, controlling event 5, is only responsible for the safety of **BUF2**.



Local controller for MACH 1



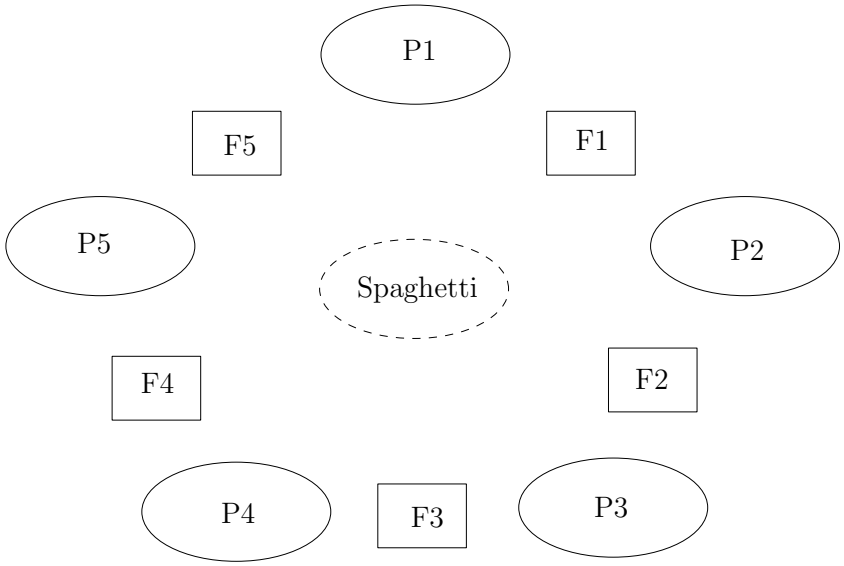
Local controller for MACH 2



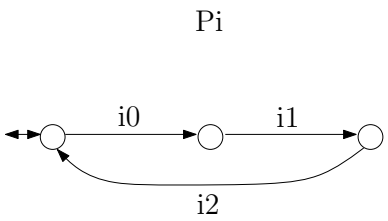
Local controller for TU

Figure 2.3: Distributed control of transfer line

2.5.2 Distributed Control: Dining Philosophers

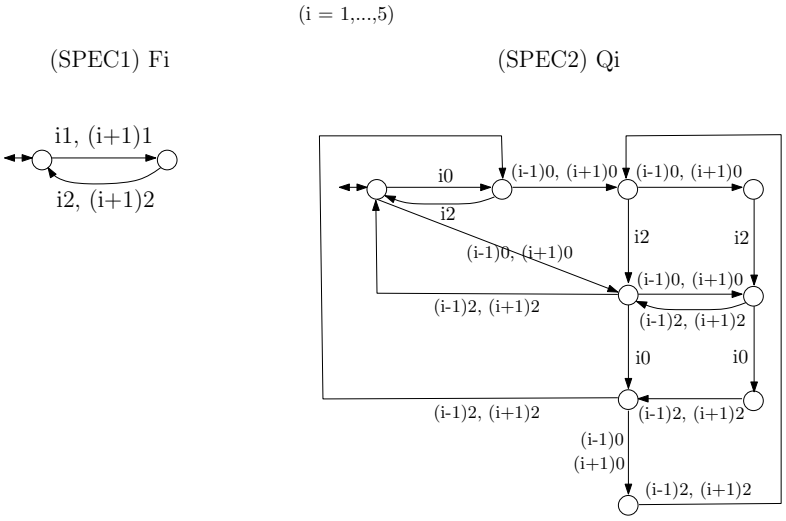


Consider the celebrated dining philosopher problem (due to E.W. Dijkstra), adapted from [63, Exercise 4.9.5]. Five philosophers (**P1**,...,**P5**) are seated at a round table, at the center of which is placed a bowl of spaghetti. There are five forks (**F1,**. . .,**F5**) on the table, one between each pair of adjacent philosophers. Taking (**P1,**. . .,**P5**) to be the plant, we model them symmetrically as follows:



(i = 1,...,5)
i0: request for forks
i1: obtain forks and start eating
i2: finish eating and return forks

There are two control specifications that restrict the philosophers' behavior: (1) a fork is used by at most one philosopher at a time; and (2) no philosopher may commence eating if either of his two neighbors has been ready longer.



The distributed control objective is to design for each philosopher a local controller – but with no external supervisor.

By using the standard method of SCT, we first build the monolithic optimal nonblocking supervisor, which has 341 states. Then we employ the localization algorithm to compute for each philosopher a local controller from the global supervisor. The resultant controllers each have 6 states and are symmetric in terms of transition structure, as displayed in Fig.2.4.

According to these symmetric local strategies, every philosopher lines up with both of his immediate neighbors and eats the spaghetti in the order that he requests the forks. It is also worth noting that each philosopher only needs the information from his immediate neighbors in order to make a correct local decision.

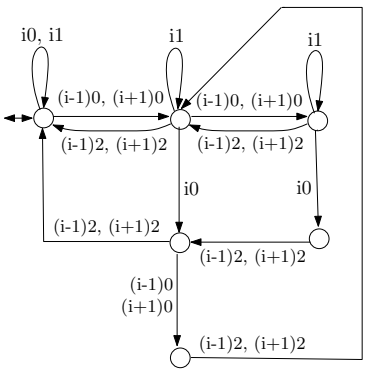


Figure 2.4: Local controller for Pi (i=1,...,5)

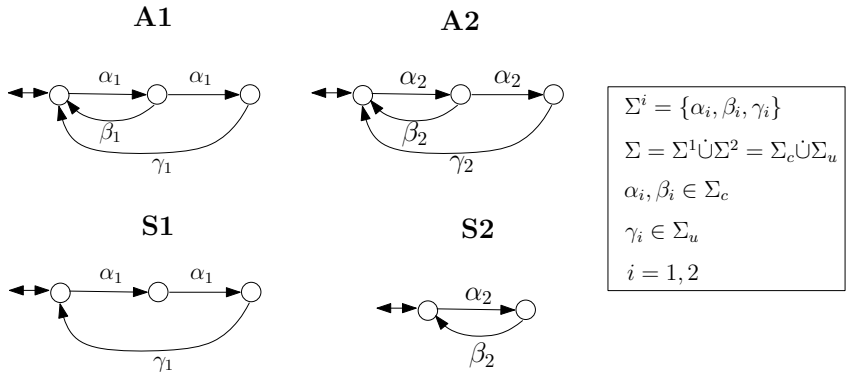
2.6 Boundary Cases

In this section we identify two boundary cases of supervisor localization which indicate, as a property of the localization problem itself, an extreme degree of 'easiness' or 'hardness', respectively.

2.6.1 Fully-localizable

This case is the 'easy' situation where component agents are completely decoupled: each agent works independently without interaction through shared events.

Example 2.5.



The plant consists of two independent agents **A1** and **A2**, while two specifications **S1** and **S2** are imposed over the disjoint alphabets corresponding to the two agents respectively. The centralized approach generates the monolithic supervisor **SUP** through synthesizing the language

$$sup\mathcal{C}([L_m(\mathbf{S1})||L_m(\mathbf{S2})] \;||\; [L_m(\mathbf{A1})||L_m(\mathbf{A2})])$$

At the local level, we can obtain $\mathbf{SUP}^i$ $(i = 1, 2)$ by synthesizing the language

$$sup\mathcal{C}(L_m(\mathbf{Si})||L_m(\mathbf{Ai}))$$

Let $P_i : \Sigma^* \to (\Sigma^i)^*$. The global extension of $\mathbf{SUP}^i$, denoted $\overline{\mathbf{SUP}^i}$, recognizes the language

$$P_i^{-1}(sup\mathcal{C}(L_m(\mathbf{Si})||L_m(\mathbf{Ai})))$$

One can easily verify that the set $\{\overline{\mathbf{SUP}^1}, \overline{\mathbf{SUP}^2}\}$ is control equivalent to $\mathbf{SUP}$, and therefore $\overline{\mathbf{SUP}^i}$ is a valid local controller for $\mathbf{Ai}$; here it can be obtained locally without going through the top-down localization procedure.

♦

In general given a plant $\mathbf{G}$ (over Σ) composed of independent agents over disjoint alphabets Σ^k $(k \in K)$, let $P_k : \Sigma^* \to (\Sigma^k)^*$ and $\mathbf{SUP}$ be a supervisor with respect to some specification.

Definition 2.5.

$\mathbf{SUP}$ is *fully-localizable* if there exists a set of local controllers $\mathbf{LOC} = \{\mathbf{LOC}^k | k \in K\}$ such that $(\forall k \in K)\ L(\mathbf{LOC}^k) = P_k^{-1}(L^k)$, for some $L^k \subseteq (\Sigma^k)^*$; and $\mathbf{LOC}$ is control equivalent to $\mathbf{SUP}$. ◊

A sufficient condition that ensures full-localizability is immediate.

Proposition 2.3.

Assume the overall specification is $\mathbf{S} = \{\mathbf{S}^m | m \in M\}$, where M is an index set. If $(\forall m \in M)(\exists k \in K)\ L(\mathbf{S}^m) \subseteq (\Sigma^k)^*$, then $\mathbf{SUP}$ (with respect to $\mathbf{S}$ and $\mathbf{G}$) is fully-localizable.

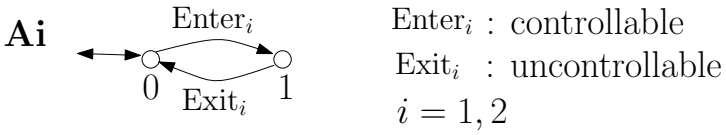
Proof.

See Appendix. □

2.6.2 Non-localizable

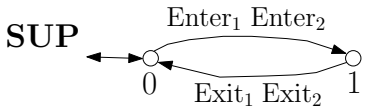
The other extreme of the localization problem is the 'hard' case where component agents are coupled so tightly that each one has to be 'globally aware'.

Example 2.6 (Mutual Exclusion)**.**

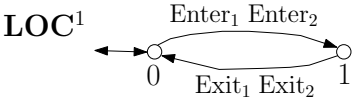


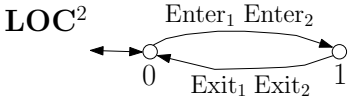
Agents **Ai** $(i = 1, 2)$ share a common resource. The specification is mutual exclusion (i.e., two agents must not simultaneously occupy the resource). In terms of state, the combination $(1, 1)$ is not allowed. The scenario of motorists can perhaps be viewed as an instance of this model.

It is easy to see that the following is a valid supervisor **SUP** that satisfies the mutual exclusion requirement.



Now we localize **SUP** by our algorithm, which results in two local controllers $\mathbf{LOC}^i$ for agents **Ai** $(i = 1, 2)$, respectively.





These are nothing but the same as **SUP**. Namely, our supervisor localization accomplished nothing useful. ♦

In the above example, the localization procedure fails to achieve a 'truly local' result. In general, we aim to find conditions that can identify this situation before we perform localization, for in that case we need only make copies of **SUP** for the relevant agents.

Definition 2.6.

Let $\mathbf{MLOC}^k$ be a state-minimal local controller (w.r.t. $\Sigma^k \subseteq \Sigma$ and **SUP**). **SUP** is *non-localizable* with respect to $\Sigma^k \subseteq \Sigma$ if $|\mathbf{SUP}| = |\mathbf{MLOC}^k|$.

$\Diamond$

First note that $|\mathbf{SUP}| = |\mathbf{MLOC}^k|$ implies $\mathbf{SUP} = \mathbf{MLOC}^k$. This is because if **SUP** is already state-minimal, then no more pairs of states in **SUP** can be merged, which in turn implies that the transition structure will remain the same.

We proceed to determine the number of cells of the control cover $\mathcal{C}^k$, corresponding to $\mathbf{MLOC}^k$, on X (the state set of **SUP**). By the definition of control cover, two states $x, x' \in X$ that belong to an identical cell must satisfy both conditions

(1) $(x, x') \in \mathcal{R}^k$

(2) $(\forall s \in \Sigma^*)\ \xi(x,s)!\ \&\ \xi(x',s)! \Rightarrow (\xi(x,s), \xi(x',s)) \in \mathcal{R}^k$

Negating (1) and (2), we get

(3) $(x, x') \notin \mathcal{R}^k$

(4) $(\exists s \in \Sigma^*)\ \xi(x,s)!\ \&\ \xi(x',s)!\ \&\ (\xi(x,s), \xi(x',s)) \notin \mathcal{R}^k$

Hence, two states x, x' belong to different cells of $\mathcal{C}^k$ if and only if either (3) or (4) holds. Let

$$\Omega_{\mathcal{C}^k} := max\,\{n | (\exists X' \subseteq X)\,|X'| = n\ \&\ (\forall x, x' \in X')\,x \neq x' \Rightarrow (3) \text{ or } (4)\}$$

The above discussion has proved the following fact.

Proposition 2.4.

$|\mathbf{MLOC}^k| = \Omega_{\mathcal{C}^k}$ $\square$

Now a necessary and sufficient condition for non-localizability is immediate.

Proposition 2.5.

SUP is non-localizable with respect to $\Sigma^k \subseteq \Sigma$ if and only if $|\mathbf{SUP}| = \Omega_{\mathcal{C}^k}$

Proof.

Follows directly from Definition 2.6 and Proposition 2.4. □

Admittedly the above condition is hardly more than a restatement of the definition of non-localizability. We still know nothing about how to check whether or not the condition holds. Nevertheless, a slight modification of $\Omega_{\mathcal{C}^k}$ will lead to a computationally verifiable sufficient condition for non-localizability.

Consider

$$\Omega_k := max\,\{n|(\exists X' \subseteq X)\,|X'| = n\ \&\ (\forall x, x' \in X')\,x \neq x' \Rightarrow (x, x') \notin \mathcal{R}^k\}$$

That is, we disregard those cases where control inconsistency is caused by related future states. It should be obvious that $\Omega_k \leq \Omega_{\mathcal{C}^k}$. More importantly, if we construct an undirected graph $G = (V, E)$ with $V = X$ and $E = \{(x, x')|(x, x') \notin \mathcal{R}^k\}$, then calculating Ω_k amounts to finding the maximum clique in G. Although the maximum clique problem is a well-known NP-complete problem, there exist efficient algorithms that compute suboptimal solutions [42]. In particular, the implemented polynomial-time algorithm that computes *lbe* (lower bound estimate) in [53] can be directly employed for our purpose. Let us denote by lbe_k the outcome of the suboptimal algorithm w.r.t. $\mathcal{R}^k$. Thus we have $lbe_k \leq \Omega_k \leq \Omega_{\mathcal{C}^k} \leq |\mathbf{SUP}|$, which gives rise to the following result.

Proposition 2.6.

If $|\mathbf{SUP}| = lbe_k$, then **SUP** is non-localizable with respect to $\Sigma^k \subseteq \Sigma$.

Proof.

$|\mathbf{SUP}| = lbe_k$ implies that $|\mathbf{SUP}| = \Omega_{\mathcal{C}^k}$, and consequently $|\mathbf{SUP}| = |\mathbf{MLOC}^k|$ by Proposition 2.4. □

This condition is not necessary for non-localizability. If we obtain $|\mathbf{SUP}| > lbe_k$, lbe_k tells us little about localizability and can only serve as a conservative lower bound

estimate indicating how much localization might (conceivably) be achieved. However, if $|\mathbf{SUP}| = lbe_k$ does hold, then the problem instance admits no useful solution, and we can avoid wasting further computational effort.

Continuing Example 2.6, and applying the adopted algorithm from [53], we obtain $lbe_{\{Enter_1\}} = lbe_{\{Enter_2\}} = 2 = |\mathbf{SUP}|$. Hence, **SUP** is non-localizable for either of the two agents, and we then simply assign the agents with the copies of **SUP** as their local controllers.

2.7 Appendix of Proofs

Proof of Proposition 2.1:

First we show $L_m(\mathbf{SUP}) \subseteq L_m(\mathbf{G}) \cap L_m(J)$. It suffices to show $(\forall k \in K)\ L_m(\mathbf{SUP}) \subseteq L_m(J^k)$. Let $k \in K$, and let $s = \sigma_0 \cdots \sigma_n \in L_m(\mathbf{SUP})$. By the definition of $\mathcal{C}^k$ and κ^k, there exist $x_0, \ldots, x_n \in X$ such that

$$\xi(x_j, \sigma_j)! \ \&\ \xi(x_j, \sigma_j) = x_{j+1}, \quad j = 0, \ldots, n-1$$

and

$$(\exists i_j^k, i_{j+1}^k \in I^k)\ x_j \in X_{i_j^k}^k \ \&\ x_{j+1} \in X_{i_{j+1}^k}^k \ \&\ \kappa^k(i_j^k, \sigma_j) = i_{j+1}^k, \quad j = 0, \ldots, n-1$$

So $\kappa^k(i_0^k, \sigma_0 \cdots \sigma_n)!$, i.e., $\kappa^k(i_0^k, s)!$. Let $i_n^k = \kappa^k(i_0^k, s)$. Then $\xi(x_0, s) \in X_{i_n^k}^k \cap X_m$, which implies $i_n^k \in I_m^k$, i.e., $s \in L_m(J^k)$.

Now that we have shown $L_m(\mathbf{SUP}) \subseteq L_m(\mathbf{G}) \cap L_m(J)$, it follows that

$$\begin{aligned} \overline{L_m(\mathbf{SUP})} &\subseteq \overline{L_m(\mathbf{G}) \cap L_m(J)} \\ &\subseteq \overline{L_m(\mathbf{G})} \cap \overline{L_m(J)} \\ &\subseteq L(\mathbf{G}) \cap L(J) \end{aligned}$$

So $L(\mathbf{SUP}) \subseteq L(\mathbf{G}) \cap L(J)$.

Next we show the converse: $L(\mathbf{G}) \cap L(J) \subseteq L(\mathbf{SUP})$. Let $s \in L(\mathbf{G}) \cap L(J)$ we proceed by induction on the length of s.

(Base case) Since none of $L(\mathbf{G}), L(J)$, and $L(\mathbf{SUP})$ is empty, ε belongs to all of them.

(Induction step) Suppose $s \in L(\mathbf{G}) \cap L(J) \Rightarrow s \in L(\mathbf{SUP})$. Let $\sigma \in \Sigma$, and assume $s\sigma \in L(\mathbf{G}) \cap L(J)$. We must show that $s\sigma \in L(\mathbf{SUP})$. By hypothesis we have $s \in L(\mathbf{SUP})$. If $\sigma \in \Sigma_u$, then $s\sigma \in L(\mathbf{SUP})$, because $L(\mathbf{SUP})$ is controllable. If $\sigma \in \Sigma_c$, then there must exist $k \in K$ such that $\sigma \in \Sigma_c^k$. Since $s\sigma \in L(J)$, $s\sigma \in L(J^k)$ and thus $s \in L(J^k)$. Hence $\kappa^k(i_0^k, s\sigma)!$ and $\kappa^k(i_0^k, s)!$. Let $i_n^k = \kappa^k(i_0^k, s)$. Then $(\exists x \in X_{i_n^k}^k, \exists x' \in X)\ \xi(x, \sigma) = x'$, which implies $\sigma \in E(x)$. It has been shown that $\xi(x_0, s) \in X_{i_n^k}^k$, so x and $\xi(x_0, s)$ belong to a common cell, i.e., $(x, \xi(x_0, s)) \in R^k$. Therefore $\sigma \notin D^k(\xi(x_0, s))$, which implies either $\xi(\xi(x_0, s), \sigma)!$ or $(\forall t \in \Sigma^*)\xi(x_0, t) = \xi(x_0, s) \Rightarrow \neg\eta(y_0, t\sigma)!$ But if $t = s$, since we have $s\sigma \in L(\mathbf{G})$, the latter case is invalid. Therefore we conclude that $\xi(\xi(x_0, s), \sigma)!$, i.e., $s\sigma \in L(\mathbf{SUP})$. This accomplishes the induction, and consequently $L(\mathbf{G}) \cap L(J) \subseteq L(\mathbf{SUP})$.

It is left to show $L_m(\mathbf{G}) \cap L_m(J) \subseteq L_m(\mathbf{SUP})$. Let $s \in L_m(\mathbf{G}) \cap L_m(J)$. Since $L_m(\mathbf{G}) \cap L_m(J) \subseteq L(\mathbf{G}) \cap L(J) \subseteq L(\mathbf{SUP})$, we have $s \in L(\mathbf{SUP})$, i.e., $\xi(x_0, s)!$, which in turn gives $(\forall k \in K)\ i_n^k = \kappa^k(i_0^k, s)!\ \&\ \xi(x_0, s) \in X_{i_n^k}^k$. In addition, $s \in L_m(J)$ implies $(\forall k \in K)\ i_n^k \in I_m^k$, and therefore $(\forall k \in K)\ X_{i_n^k}^k \cap X_m \neq \emptyset$. Let $k \in K$, and let $x' \in X_{i_n^k}^k \cap X_m$. Then $M(x') = 1$, and thus $T(x') = 1$. By $s \in L_m(\mathbf{G})$, we have $\eta(y_0, s) \in Y_m$, i.e., $T(\xi(x_0, s)) = 1$. Hence both x' and $\xi(x_0, s)$ are in $X_{i_n^k}^k$, i.e., $(x', \xi(x_0, s)) \in R^k$. Consequently $M(\xi(x_0, s)) = M(x') = 1$, i.e., $s \in L_m(\mathbf{SUP})$. □

Proof of Proposition 2.2:

It is assumed that

$$L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP}) \text{ and } L_m(\mathbf{G}) \cap L_m(\mathbf{LOC}) = L_m(\mathbf{SUP})$$

We will show that

$$L(\mathbf{G}) \cap L(\mathbf{NLOC}) = L(\mathbf{SUP}) \text{ and } L_m(\mathbf{G}) \cap L_m(\mathbf{NLOC}) = L_m(\mathbf{SUP})$$

First we show $L(\mathbf{G}) \cap L(\mathbf{NLOC}) = L(\mathbf{SUP})$.

($\subseteq$) By (N1) and (N2), $L(\mathbf{NLOC}^k) \subseteq L(\mathbf{LOC}^k)$ for all $k \in K$. So $\bigcap\{L(\mathbf{NLOC}^k)|k \in K\} \subseteq \bigcap\{L(\mathbf{LOC}^k)|k \in K\}$, i.e., $L(\mathbf{NLOC}) \subseteq L(\mathbf{LOC})$. It follows that $L(\mathbf{G}) \cap L(\mathbf{NLOC}) \subseteq L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP})$.

($\supseteq$) Letting $s \in L(\mathbf{SUP})$, we proceed by induction on the length of s.

(Base case) The empty string $\varepsilon \in L(\mathbf{SUP})$ because $L(\mathbf{SUP}) \neq \emptyset$. So $\varepsilon \in L(\mathbf{G}) \cap \bigcap\{L(\mathbf{LOC}^k)|k \in K\}$. Suppose $\varepsilon \notin L(\mathbf{NLOC})$. Then there exists $k \in K$ such that $\varepsilon \notin L(\mathbf{NLOC}^k)$, i.e., z_0^k was deleted by (N1). Hence $(\neg\exists t \in L(\mathbf{SUP}))\ \zeta^k(z_0^k, t) = z_0^k$. But this is a contradiction, for $\varepsilon \in L(\mathbf{SUP})$ and $\varepsilon \in L(\mathbf{LOC}^k)$. Therefore, $\varepsilon \in L(\mathbf{NLOC})$, and thus $\varepsilon \in L(\mathbf{G}) \cap L(\mathbf{NLOC})$.

(Induction step) Suppose that $s \in L(\mathbf{SUP}) \Rightarrow s \in L(\mathbf{G}) \cap L(\mathbf{NLOC})$. Let $\sigma \in \Sigma$, and assume that $s\sigma \in L(\mathbf{SUP})$. We must show $s\sigma \in L(\mathbf{G}) \cap L(\mathbf{NLOC})$. Since $s\sigma \in L(\mathbf{SUP}) = L(\mathbf{G}) \cap \bigcap\{L(\mathbf{LOC}^k)|k \in K\}$, we have $s \in L(\mathbf{SUP}) = L(\mathbf{G}) \cap \bigcap\{L(\mathbf{LOC}^k)|k \in K\}$. Let $k \in K$ and $z^k = \zeta^k(z_0^k, s)$, and suppose that $s\sigma \notin L(\mathbf{NLOC}^k)$. It follows from $s\sigma \in L(\mathbf{LOC}^k)$ that this one-step transition σ at z^k was deleted by (N2), which implies that $(\forall t \in L(\mathbf{SUP}))\ \zeta^k(z_0^k, t) = z^k\ \&\ t\sigma \notin L(\mathbf{SUP})$. But this is a contradiction, for s, $s\sigma \in L(\mathbf{SUP})$ and $\zeta^k(z_0^k, s) = z^k$. So $s\sigma \in L(\mathbf{NLOC}^k)$ for any $k \in K$, i.e., $s\sigma \in \bigcap\{L(\mathbf{NLOC}^k)|k \in K\}$. Therefore $s\sigma \in L(\mathbf{G}) \cap L(\mathbf{NLOC})$. This accomplishes the induction, and establishes $L(\mathbf{G}) \cap L(\mathbf{NLOC}) \supseteq L(\mathbf{SUP})$.

Now we show $L_m(\mathbf{G}) \cap L_m(\mathbf{NLOC}) = L_m(\mathbf{SUP})$.

($\subseteq$) By (N3) we have $L_m(\mathbf{NLOC}^k) \subseteq L_m(\mathbf{LOC}^k)$ for all $k \in K$. So $\bigcap\{L_m(\mathbf{NLOC}^k)|k \in K\} \subseteq \bigcap\{L_m(\mathbf{LOC}^k)|k \in K\}$, i.e., $L_m(\mathbf{NLOC}) \subseteq L_m(\mathbf{LOC})$. It follows that $L_m(\mathbf{G}) \cap L_m(\mathbf{NLOC}) \subseteq L_m(\mathbf{G}) \cap L_m(\mathbf{LOC}) = L_m(\mathbf{SUP})$.

($\supseteq$) Let $s \in L_m(\mathbf{SUP})$. Then $s \in L_m(\mathbf{G}) \cap \bigcap\{L_m(\mathbf{LOC}^k)|k \in K\}$. Let $k \in K$ and $z^k = \zeta^k(z_0^k, s)$. Hence $z^k \in Z_m^k$. Suppose that $s \notin L_m(\mathbf{NLOC}^k)$, i.e., z^k was unmarked by (N3). It follows that $(\neg\exists t \in L_m(\mathbf{SUP}))\ \zeta^k(z_0^k, t) = z^k$. But this is a contradiction, for $s \in L_m(\mathbf{SUP})$ & $\zeta^k(z_0^k, s) = z^k$. So $s \in L_m(\mathbf{NLOC}^k)$ for any $k \in K$, i.e., $s \in \bigcap\{L_m(\mathbf{NLOC}^k)|k \in K\}$. Therefore $s \in L_m(\mathbf{G}) \cap L_m(\mathbf{NLOC})$. We conclude that $L_m(\mathbf{G}) \cap L_m(\mathbf{NLOC}) \supseteq L_m(\mathbf{SUP})$ □

Proof of Theorem 2.1

Let $k \in K$. We must show that there exists a control cover $\mathcal{C}^k$ such that the induced generator J^k is DES-isomorphic to $\mathbf{LOC}^k$. Let $z^k \in Z^k$, and define

$$X(z^k) := \{x \in X|(\exists s \in L(\mathbf{SUP}))\xi(x_0, s) = x\ \&\ \zeta^k(z_0^k, s) = z^k\}$$

Also define $\mathcal{C}^k = \{X(z^k)|z^k \in Z^k\}$. We claim that $\mathcal{C}^k$ is a control cover on X with respect to Σ_c^k.

First we show that $\mathcal{C}^k$ covers X. Let $x \in X$. Then $(\exists s \in L(\mathbf{SUP}))\xi(x_0, s) = x$. Since $\mathbf{LOC}$ is control equivalent to $\mathbf{SUP}$ w.r.t. $\mathbf{G}$, we have $s \in L(\mathbf{LOC}^k)$, i.e., $\zeta^k(z_0^k, s)!$. Let $z^k = \zeta^k(z_0^k, s)$. Then $x \in X(z^k)$, by the definition of $X(z^k)$.

Next we show $(\forall z^k \in Z^k)\ X(z^k) \neq \emptyset$. Let $z^k \in Z^k$. Since $\mathbf{LOC}^k$ is normal w.r.t. $\mathbf{SUP}$, then $(\exists s \in L(\mathbf{SUP}))\ \zeta^k(z_0^k, s) = z^k$. Hence $\xi(x_0, s)!$ & $\xi(x_0, s) \in X(z^k)$, i.e., $X(z^k) \neq \emptyset$.

Now we must show that two states are control consistent if they belong to the same cell, i.e., $(\forall z^k \in Z^k, \forall a, b \in X(z^k))\ (a, b) \in R^k$. Let $z^k \in Z^k$ and $a, b \in X(z^k))$. It is equivalent to show that

$$E(a) \cap D^k(b) = \emptyset = E(b) \cap D^k(a)$$

and

$$T(a) = T(b) \Rightarrow M(a) = M(b)$$

Let $\sigma \in \Sigma$, and assume $\sigma \in E(a)$. It will be shown that $\sigma \notin D^k(b)$. If $\sigma \in \Sigma - \Sigma_c^k$, then by the definition of D^k, $\sigma \notin D^k(b)$. Otherwise if $\sigma \in \Sigma_c^k$, we have $(\exists s \in L(\mathbf{SUP}))\ \xi(x_0, s) = a\ \&\ \xi(a, \sigma)!$ because $a \in X\ \&\ \sigma \in E(a)$. Hence $s\sigma \in L(\mathbf{SUP})$. Since **LOC** is control equivalent to **SUP**, we obtain $s\sigma \in L(\mathbf{LOC}^k)$, i.e., $\zeta^k(\zeta^k(z_0^k, s), \sigma)!$. It then follows from $a \in X(z^k)$ that $\zeta^k(z^k, \sigma)!$. Because $b \in X(z^k)$, by definition $(\exists t \in L(\mathbf{SUP}))\ \xi(x_0, t) = b\ \&\ \zeta^k(z_0^k, t) = z^k$, so that $t\sigma \in L(\mathbf{LOC}^k)$. If $t\sigma \notin L(\mathbf{G})$, then trivially $\sigma \notin D^k(b)$; for the case $t\sigma \in L(\mathbf{G})$, i.e., $\eta(y_0, t\sigma)!$, since only $\mathbf{LOC}^k$ has control authority on Σ_c^k:

$$(\forall k' \in K)\ k' \neq k \Rightarrow \zeta^{k'}(z_0^{k'}, t\sigma)!,\ \text{i.e., } t\sigma \in L(\mathbf{LOC}^{k'})$$

we have $t\sigma \in L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP})$. Hence $\xi(\xi(x_0, t), \sigma)!$, i.e., $\xi(b, \sigma)!$, which implies $\sigma \in E(b)\ \&\ \sigma \notin D^k(b)$. Therefore $E(a) \cap D^k(b) = \emptyset$, and by the same argument we can show $E(b) \cap D^k(a) = \emptyset$.

We proceed to show $T(a) = T(b) \Rightarrow M(a) = M(b)$ by contraposition. Suppose $M(a) \neq M(b)$, say $M(a) = 1\ \&\ M(b) = 0$. Then $T(a) = 1$ and $a \in X_m$. Since $a \in X(z^k)$ and $\mathbf{LOC}^k$ is normal, we have $(\exists s \in L_m(\mathbf{SUP}))\ \xi(x_0, s) = a\ \&\ \zeta^k(z_0^k, s) = z^k$. It follows from the control equivalence condition that $s \in L_m(\mathbf{G}) \cap L_m(\mathbf{LOC})$, which gives that $(\forall k' \in K)\ z^k \in Z_m^{k'}$ (we use k' to distinguish the index from the fixed k). Because $b \in X(z^k)$, by definition $(\exists t \in L(\mathbf{SUP}))\ \xi(x_0, t) = b\ \&\ \zeta^k(z_0^k, t) = z^k$, and because $M(b) = 0$, we have $t \notin L_m(\mathbf{SUP})$. Therefore $t \notin L_m(\mathbf{G})$ or $t \notin \bigcap\{L_m(\mathbf{LOC}^{k'}) | k' \in K\}$. But $t \in \bigcap\{L_m(\mathbf{LOC}^{k'}) | k' \in K\}$, for $(\forall k' \in K)\ z^k \in Z_m^{k'}\ \&\ \zeta^k(z_0^k, t) = z^k$. So $t \notin L_m(\mathbf{G})$, i.e., T(b)=0. Thus $T(a) \neq T(b)$. The same conclusion could be drawn if we started with $M(a) = 0\ \&\ M(b) = 1$.

Finally we show $(\forall z^k \in Z^k, \forall \sigma \in \Sigma)[(\forall x \in X(z^k))\xi(x, \sigma)!$

$$\Rightarrow (\exists \tilde{z^k} \in Z^k)\xi(x,\sigma) \in X(\tilde{z^k})]$$

Let $z^k \in Z^k$ & $x \in X(z^k)$ & $\sigma \in \Sigma$, and assume $\xi(x,\sigma)!$. So $(\exists s \in L(\mathbf{SUP}))\ \xi(x_0,s) = x$ & $\zeta^k(z_0^k,s) = z^k$, which implies $s\sigma \in L(\mathbf{SUP}) = L(\mathbf{G}) \cap \{L(\mathbf{LOC})\}$. Hence $s\sigma \in L(\mathbf{LOC}^k)$, i.e., $\zeta^k(z_0^k,s\sigma)!$ Let $\tilde{z^k} = \zeta^k(z_0^k,s\sigma)$. Then $\xi(x,\sigma) = \xi(x_0,s\sigma) \in X(\tilde{z^k})$.

This establishes the claim, and it is left to show that J^k is DES-isomorphic to $\mathbf{LOC}^k$. Suppose $I^k = Z^k$ & $i_0^k = z_0^k$. It will be shown that $I_m^k = Z_m^k$ & $\kappa^k = \zeta^k$. Therefore J^k is DES-isomorphic to $\mathbf{LOC}^k$, with the identity map the DES-isomorphism.

For $I_m^k = Z_m^k$: ($\supseteq$) Let $z^k \in Z_m^k$. Since $\mathbf{LOC}^k$ is normal, by definition $(\exists s \in L_m(\mathbf{SUP}))\ \zeta^k(z_0^k,s) = z^k$, which gives $\xi(x_0,s)!$ & $\xi(x_0,s) \in X_m$. It follows that $\xi(x_0,s) \in X(z^k)$, and $X(z^k) \cap X_m \neq \emptyset$. Hence $z^k \in I_m^k$. ($\subseteq$) Let $z^k \in I_m^k$. Then $X(z^k) \cap X_m \neq \emptyset$. Let $x \in X(z^k) \cap X_m$. By definition of $X(z^k)$ we have $(\exists s \in L_m(\mathbf{SUP}))\ \xi(x_0,s) = x$ & $\zeta^k(z_0^k,s) = z^k$. It then follows from the control equivalence condition that $s \in L_m(\mathbf{LOC}^k)$, and threfore $z^k \in Z_m^k$.

For $\kappa^k = \zeta^k$: Let $z^k \in Z^k$ and $\sigma \in \Sigma$. We must show that $\kappa^k(z^k,\sigma) = \zeta^k(z^k,\sigma)$. ($\Rightarrow$) Assume $\kappa^k(z^k,\sigma) = \tilde{z^k}$. So $(\exists x \in X(z^k))\ \xi(x,\sigma) \in X(\tilde{z^k})$, which implies $(\exists s \in L(\mathbf{SUP}))\ \xi(x_0,s) = x$ & $\zeta^k(z_0^k,s) = z^k$ & $\zeta^k(z_0^k,s\sigma) = \tilde{z^k}$. Thus $\zeta^k(\zeta^k(z_0^k,s),\sigma) = \tilde{z^k}$,i.e., $\zeta^k(z^k,\sigma) = \tilde{z^k}$. ($\Leftarrow$) Assume $\zeta^k(z^k,\sigma) = \tilde{z^k}$. Since $\mathbf{LOC}^k$ is normal, by definition $(\exists s \in L(\mathbf{SUP}))[\zeta^k(z_0^k,s) = z^k$ & $s\sigma \in L(\mathbf{SUP})]$. It follows that $\xi(x_0,s) \in X(z^k)$ & $\zeta^k(z_0^k,s\sigma) = \tilde{z^k}$. Consequently, $\xi(\xi(x_0,s),\sigma) \in X(\tilde{z^k})$, and thus $\kappa^k(z^k,\sigma) = \tilde{z^k}$. □

Proof of Proposition 2.3:

First note that, for $m \in M$, if there exists $k \in K$ such that $L(\mathbf{S}^m) \subseteq (\Sigma^k)^*$, then this k is unique. Now suppose that, for $k \in K$,

$$(\forall m \in M_k)\ L(\mathbf{S}^m) \subseteq (\Sigma^k)^*$$

where $M_k \subseteq M$. Thus the overall marked language of these specifications is

$$L_m(\mathbf{SPEC}^k) := ||\{L_m(S^m)|m \in M_k\}$$

Then we can obtain a local controller $\mathbf{LOC}^k$ by synthesizing the language

$$L_m(\mathbf{LOC}^k) := sup\mathcal{C}(L_m(\mathbf{SPEC}^k)||L_m(\mathbf{G}^k)) \subseteq (\Sigma^k)^*$$

Let $P_k : \Sigma^* \to (\Sigma^k)^*$. The global extension of $\mathbf{LOC}^k$, denoted $\overline{\mathbf{LOC}^k}$, recognizes the language $P_k^{-1}(L_m(\mathbf{LOC}^k))$. For any $k \in K$, owing to the assumption that Σ^k are pairwise disjoint, the $P_k^{-1}(L_m(\mathbf{LOC}^k))$ are necessarily pairwise nonconflicting. Hence, $\{\overline{\mathbf{LOC}^k}|k \in K\}$ is control equivalent to the monolithic $\mathbf{SUP}$, and

$$(\forall k \in K)\ L(\overline{\mathbf{LOC}^k}) = P_k^{-1}(L(\mathbf{LOC}^k))$$

where $L(\mathbf{LOC}^k) \subseteq (\Sigma^k)^*$. Therefore, $\mathbf{SUP}$ is fully-localizable by Definition 2.5. □

Chapter 3

Supervisor Localization of Large-Scale DES

3.1 Introduction

In Chapter 2 we developed a supervisor localization procedure that accomplishes distributed control design for DES, whenever the monolithic supervisor for a given control problem exists. In this chapter we move on to study the same distributed control problem for large-scale DES. "Large" is a subjective term, and so is "large-scale DES". We take the pragmatic view that a DES is large-scale whenever "it is made up of a large number of parts that interact in a nonsimple way" [51]. Largeness may well bring in formidable complexity, which can render the "one-shot" supervisor synthesis for DES uneconomical or even intractable. Indeed, Gohari and Wonham [18] have proved that, if either the plant or specification is described modularly, the monolithic supervisor design in the automaton-based framework is NP-hard. This fact makes it clear that the monolithic supervisor is in general not feasibly computable for large-scale DES, and hence the supervisor localization procedure established in the previous chapter cannot be applied directly.

A promising strategy to handle complexity may be abstracted from the following illuminating example provided by Simon and Ando [52]:

> Suppose that government planners are interested in the effects of a subsidy to a basic industry, say the steel industry, on the total effective demand in the economy. Strictly speaking, we must deal with individual producers and consumers, and trace through all interactions among the economic agents in the economy. This being an obviously impossible task, we would use such aggregated variables as the total output of the steel industry, aggregate consumption and aggregate investment. The reasoning behind such a procedure may be summarized as follows: (1) we can somehow classify all the variables in the economy into a small number of groups; (2) we can study the interactions within the groups as though the interaction among groups did not exist; (3) we can define indices representing groups and study the interaction among these indices without regard to the interactions within each group.

In this example, the one-shot approach with the welter of detail is considered impossible, and instead a "decomposition-aggregation" procedure is proposed to tackle the problem systematically. These two strategies, decomposition and aggregation, are also considered by Siljak [50] to be two fundamental and effective processes by which we can achieve both conceptual simplification in abstract analysis, and numerical feasibility in actual computations. For these reasons, we are motivated to combine supervisor localization with computationally efficient modular control theories: first design an organization of modular supervisors that achieves optimal nonblocking control; then decompose each of these modular supervisors into the local controllers of the relevant agents.

In Section 3.2, we begin with the basics of the modular supervisory control theory with which the supervisor localization will be combined. Then in Section 3.3 we formulate the distributed control problem for large-scale DES, and present the solution in terms of a systematic procedure. Finally, in Sections 3.4 and 3.5, we illustrate our solution

procedure by going through two large-scale examples in detail.

3.2 Preliminaries

3.2.1 Quasi-Congruences of Nondeterministic Generator

Let us begin with *congruences of a deterministic dynamic system* [63, Example 1.4.1].

Let Y be a set, and $\xi : Y \to Y$ be a map. A *deterministic dynamic system* is the pair (Y, ξ), with the interpretation that the elements $y \in Y$ are the system 'states', and ξ is the 'state transition function'. Denote by $\mathcal{E}(Y)$ the set of all equivalence relations on Y, and let $\pi \in \mathcal{E}(Y)$ with canonical projection $P_\pi : Y \to \bar{Y} := Y/\pi$. Then π is a *congruence* of (Y, ξ) if there exists a map $\bar{\xi} : \bar{Y} \to \bar{Y}$ such that

$$\bar{\xi} \circ P_\pi = P_\pi \circ \xi$$

Namely, the following diagram commutes

$$\begin{array}{ccc} Y & \xrightarrow{\xi} & Y \\ {\scriptstyle P_\pi}\downarrow & & \downarrow{\scriptstyle P_\pi} \\ \bar{Y} & \xrightarrow{\bar{\xi}} & \bar{Y} \end{array}$$

Thus the induced deterministic dynamic system $(\bar{Y}, \bar{\xi})$ can be viewed as a consistent aggregated model of (Y, ξ).

Next we review *quasi-congruences of a nondeterministic dynamic system* [63, Exercise 1.4.10]. Again let Y be a set of states, but now let the state transition function be $\delta : Y \to Pwr(Y)$, mapping states y into subsets of Y. A *nondeterministic dynamic system* is the pair (Y, δ). Let $\pi \in \mathcal{E}(Y)$ with canonical projection $P_\pi : Y \to \bar{Y} := Y/\pi$.

With P_π we associate the induced function $P_{\pi*} : Pwr(Y) \to Pwr(\bar{Y})$ according to

$$P_{\pi*}(S) := \{P_\pi(y) | y \in S\} \subseteq \bar{Y}$$

for $S \subseteq Y$. Then π is a *quasi-congruence* of (Y, δ) if there exists a map $\bar{\delta} : \bar{Y} \to Pwr(\bar{Y})$ such that

$$\bar{\delta} \circ P_\pi = P_{\pi*} \circ \delta$$

Namely, the following diagram commutes

$$\begin{array}{ccc} Y & \xrightarrow{\delta} & Pwr(Y) \\ {\scriptstyle P_\pi}\downarrow & & \downarrow{\scriptstyle P_{\pi*}} \\ \bar{Y} & \xrightarrow{\bar{\delta}} & Pwr(\bar{Y}) \end{array}$$

Thus the induced nondeterministic dynamic system $(\bar{Y}, \bar{\delta})$ is a consistent 'lumped' abstraction of (Y, δ).

We now discuss *quasi-congruences of a nondeterministic generator* [63, Section 6.7]. A nondeterministic generator extends a nondeterministic dynamic system, in the sense that state transitions are triggered by the occurrences of events. Formally a *nondeterministic generator* is a 5-tuple

$$\mathbf{T} = (Y, \Sigma, \tau, y_0, Y_m)$$

where the state transition function τ maps pairs (y, σ) into subsets of Y:

$$\tau : Y \times \Sigma \to Pwr(Y)$$

Let $\pi \in \mathcal{E}(Y)$ with canonical projection $P_\pi : Y \to \bar{Y} := Y/\pi$ and its associated induced function $P_{\pi*} : Pwr(Y) \to Pwr(\bar{Y})$. We say π is a *quasi-congruence* of $\mathbf{T}$ if there

exists a map $\bar{\tau} : \bar{Y} \times \Sigma \to Pwr(\bar{Y})$ such that

$$(\forall \sigma \in \Sigma)\ \bar{\tau}(\cdot, \sigma) \circ P_\pi = P_{\pi *} \circ \tau(\cdot, \sigma)$$

Namely, the following diagram commutes.

$$\begin{array}{ccc} Y \times \Sigma & \xrightarrow{\tau} & Pwr(Y) \\ {\scriptstyle P_\pi}\downarrow & \downarrow{\scriptstyle id} & \downarrow{\scriptstyle P_{\pi *}} \\ \bar{Y} \times \Sigma & \xrightarrow{\bar{\tau}} & Pwr(\bar{Y}) \end{array}$$

It follows directly from [63, Proposition 1.4.1] that π is a quasi-congruence of $\mathbf{T}$ if and only if

$$(\forall \sigma \in \Sigma)\ \ker P_\pi \leq \ker P_{\pi *} \circ \tau(\cdot, \sigma)$$

Namely,

$$(\forall y, y' \in Y, \forall \sigma \in \Sigma)\ (y, y') \in \pi \Rightarrow (\tau(y, \sigma), \tau(y', \sigma)) \in \pi$$

Note that $\perp \in \mathcal{E}(Y)$ is trivially a quasi-congruence of $\mathbf{T}$, but $\top \in \mathcal{E}(Y)$ generally need not be. Let $\mathcal{QC}(Y) \subseteq \mathcal{E}(Y)$ be the set of all quasi-congruences of $\mathbf{T}$; then it can be shown that $\mathcal{QC}(Y)$ is a complete upper semilattice of $\mathcal{E}(Y)$: $\mathcal{QC}(Y)$ is closed under the join operation, but need not be closed under the meet operation, of $\mathcal{E}(Y)$. In particular, $\mathcal{QC}(Y)$ contains a (unique) supremal element, denoted by $\rho := sup\, \mathcal{QC}(Y)$.

We now consider the computation of ρ. For $\sigma \in \Sigma$ we define an equivalence relation $\pi \circ \tau(\cdot, \sigma) \in \mathcal{E}(Y)$ according to

$$(y, y') \in \pi \circ \tau(\cdot, \sigma) \text{ iff } (\tau(y, \sigma), \tau(y', \sigma)) \in \pi$$

Also let $E_\sigma := \{y \in Y | \tau(y, \sigma) \neq \emptyset\}$ and $\pi_\sigma := \{E_\sigma, Y - E_\sigma\} \in \mathcal{E}(Y)$; then we define

$$\rho_0 := \bigwedge \{\pi_\sigma | \sigma \in \Sigma\} \in \mathcal{E}(Y)$$

Consider the sequence $\rho_n \in \mathcal{E}(X)$:

$$\rho_n := \rho_{n-1} \wedge \bigwedge\{\rho_{n-1} \circ \tau(\cdot, \sigma) | \sigma \in \Sigma\}, \quad n \geq 1$$

Let Y be finite. One can prove that

$$\rho = \lim_{n \to \infty} \rho_n$$

with the limit being achieved in finitely many steps.

Finally we turn to the *induced nondeterministic generator* $\bar{\mathbf{T}}$. Let π be a quasi-congruence of $\mathbf{T}$, $\bar{y}_0 = P_\pi(y_0)$, and $\bar{Y}_m = P_{\pi *}(Y_m)$. Then the induced nondeterministic generator, or the *reduction* of $\mathbf{T}$ (mod π), is

$$\bar{\mathbf{T}} = (\bar{Y}, \Sigma, \bar{\tau}, \bar{y}_0, \bar{Y}_m)$$

With $\rho = sup\, \mathcal{QC}(Y)$, the reduction of $\mathbf{T}$ (mod ρ) can be regarded as the canonical form of $\mathbf{T}$ with respect to quasi-congruence. One can also verify that $\perp \in \mathcal{E}(\bar{Y})$ is the only quasi-congruence of $\bar{\mathbf{T}}$.

3.2.2 $L_m(\mathbf{G})$-Observer

In this subsection, we introduce a key property of natural projections – $L_m(\mathbf{G})$-observer – and provide a computational test for this property [63, Section 6.7]. In developing the test, quasi-congruences of a nondeterministic generator play a central role.

Given a deterministic generator $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ as before, for simplicity we assume $\mathbf{G}$ is reachable and coreachable. Let $\Sigma_o \subseteq \Sigma$ be a subset of observable events and $P : \Sigma^* \to \Sigma_o^*$ be the corresponding natural projection.

Definition 3.1.

P is an $L_m(\boldsymbol{G})$-*observer* if

$$(\forall s \in L(\mathbf{G}), \forall t_o \in \Sigma_o^*)\ (Ps)t_o \in PL_m(\mathbf{G}) \Rightarrow (\exists t \in \Sigma^*)\ Pt = t_o\ \&\ st \in L_m(\mathbf{G})$$

$\Diamond$

Informally, whenever Ps can be extended to $PL_m(\mathbf{G})$ by an observable string t_o, the underlying string s can be extended to $L_m(\mathbf{G})$ by a string t with $Pt = t_o$. The $L_m(\mathbf{G})$-observer property is of importance because it has been proved to ensure nonblocking decentralized supervisory control [14, Section 4.1.2]. Thus an immediate question is how to verify this key property. In the following, we present a computational procedure to check whether or not a given natural projection P is an $L_m(\mathbf{G})$-observer.

First we define a nondeterministic generator by $\mathbf{G}$ and P:

$$\mathbf{H} = (Q, \Sigma_o, \eta, q_0, Q_m)$$

with $\eta : Q \times \Sigma_o \to Pwr(Q)$ given by

$$\eta(q, \sigma) = \{\delta(q, s) | s \in \Sigma^*, \delta(q, s)!, Ps = \sigma\}$$

Notice that η can be considered a total function because of the possible evaluation $\eta(q, \sigma) = \emptyset$; namely, whenever

$$(\forall s \in \Sigma^*)\ Ps = \sigma \Rightarrow \neg\delta(q, s)!$$

Also note the following fact about 'silent transitions':

$$\eta(q, \varepsilon) = \{\delta(q, s) | Ps = \varepsilon\} \supseteq \{q\}$$

Next we bring in a new event label μ ($\mu \notin \Sigma$) to 'signal' each $q \in Q_m$ through adding

selfloops (q, μ, q). Denote this new nondeterministic generator

$$\mathbf{T} = (Q, \Sigma_o', \tau, q_0, Q_m)$$

where $\Sigma_o' = \Sigma_o \cup \{\mu\}$, and τ is η extended to $Q \times \Sigma_o'$ as described above.

Following Subsection 3.2.1, we now compute the supremal quasi-congruence, denoted by ρ, of $\mathbf{T}$. For $\sigma \in \Sigma_o'$, let $E_\sigma := \{q \in Q | \tau(q, \sigma) \neq \emptyset\}$ and $\pi_\sigma := \{E_\sigma, Q - E_\sigma\} \in \mathcal{E}(Q)$; we define

$$\rho_0 := \bigwedge \{\pi_\sigma | \sigma \in \Sigma_o'\} \in \mathcal{E}(Q)$$

Consider the sequence $\rho_n \in \mathcal{E}(Q)$

$$\rho_n := \rho_{n-1} \wedge \bigwedge \{\rho_{n-1} \circ \tau(\cdot, \sigma) | \sigma \in \Sigma_o'\}, \quad n \geq 1$$

Assuming Q is finite (as usual), $\rho = \lim_{n \to \infty} \rho_n$ and this limit is achieved in finitely many steps.

Having computed ρ, we obtain the reduction of $\mathbf{T}$ (mod ρ):

$$\bar{\mathbf{T}} = (\bar{Q}, \Sigma_o', \bar{\tau}, \bar{q}_0, \bar{Q}_m)$$

$\bar{\mathbf{T}}$ is said to be *structurally deterministic* if

$$\bar{\tau}(\bar{q}, s_o) \neq \emptyset \Rightarrow |\bar{\tau}(\bar{q}, s_o)| = 1$$

for all $\bar{q} \in \bar{Q}$, and for $s_o = \varepsilon$ or $s_o = \sigma \in \Sigma_o'$. Otherwise $\bar{\mathbf{T}}$ is *structurally nondeterministic*. The conclusion [63, Theorem 6.7.1] is: P is an $L_m(\mathbf{G})$-observer if and only if $\bar{\mathbf{T}}$ is structurally deterministic.

To summarize, given a deterministic generator $\mathbf{G}$ over Σ and an observable event subset $\Sigma_o \subseteq \Sigma$, checking if the natural projection $P : \Sigma^* \to \Sigma_o^*$ is an $L_m(\mathbf{G})$-observer

amounts to checking whether the result $\bar{\mathbf{T}}$ of the above computational procedure is structurally deterministic. When P does not enjoy the observer property, we consider adding a minimal number of events to Σ_o so that the augmented observable event set does define an $L_m(\mathbf{G})$-observer. This is the *minimal extension problem* addressed in [14, Chapter 5]. There, it was proved that a unique extension through adding a minimal number of events generally does not exist for $L_m(\mathbf{G})$-observers, and even finding some minimal extension is in fact NP-hard. Nevertheless, a polynomial-time algorithm is presented which accomplishes a 'reasonable' extension that achieves the observer property; of course this extension need not always be minimal. Henceforth we refer to this algorithm as the minimal extension (MX) algorithm.

3.2.3 Computationally Efficient Modular Supervisory Control

We now present the modular control theory with which we will combine supervisor localization. This modular approach generates a heterarchical system architecture: plant components are controlled by a hierarchy of decentralized supervisors and coordinators. The theory first and foremost ensures that these modular supervisors, operating concurrently, achieve performance identical to that realized by the monolithic optimal nonblocking supervisor; furthermore the approach is (usually) computationally efficient, for the model abstraction technique is employed to hide inessential generator dynamics. A key role in this theory is played by the $L_m(\mathbf{G})$-observer property, which provides a model abstraction that guarantees nonblocking control. The material presented here is adapted from [17].

Consider a plant $\mathbf{G}$ consisting of component agents $\mathbf{G}_i$ defined over pairwise disjoint alphabets Σ_i ($i \in I$, I an index set.) Thus $\mathbf{G}$ is defined over the alphabet

$$\Sigma = \dot{\bigcup} \{\Sigma_i | i \in I\}$$

Let $L_i := L(\mathbf{G}_i)$ and $L_{m,i} := L_m(\mathbf{G}_i)$; then the closed and marked languages of $\mathbf{G}$ are

$$L := L(\mathbf{G}) = ||\{L_i | i \in I\} \quad \text{and} \quad L_m := L_m(\mathbf{G}) = ||\{L_{m,i} | i \in I\}$$

For simplicity we assume $\mathbf{G}_i$ is nonblocking (i.e. $\bar{L}_{m,i} = L_i$), for all $i \in I$. Then $\mathbf{G}$ is necessarily nonblocking (i.e. $\bar{L}_m = L$.)

First we consider the case of a single specification. Let $E \subseteq \Sigma_o^*$, where $\Sigma_o \subseteq \Sigma$, be a specification language, and $P : \Sigma^* \to \Sigma_o^*$ be the corresponding natural projection. By SCT, we obtain the monolithic supervisor by synthesizing the language

$$K := sup\mathcal{C}(P^{-1}E \cap L_m) \subseteq \Sigma^*$$

On the other hand, we can obtain a decentralized supervisor by synthesizing the language

$$K_d := sup\mathcal{C}(E \cap PL_m) \subseteq \Sigma_o^*$$

Thus the central question is: what condition(s) can ensure identical controlled behaviors of the monolithic and the decentralized supervisor, i.e.,

$$K = P^{-1}K_d \cap L_m$$

Definition 3.2.

Let $\Sigma_u \subseteq \Sigma$ be the uncontrollable event subset. The natural projection $P : \Sigma^* \to \Sigma_o^*$ is *output control consistent* (OCC) for L if for every string $s \in L$ of the form

$$s = s'\sigma_1 \cdots \sigma_k, \ k \geq 1$$

where s' is either the empty string or terminates with an event in Σ_o, the following holds

$$\big((\forall i \in [1, k-1])\, \sigma_i \in \Sigma \backslash \Sigma_o\big) \;\; \& \;\; \sigma_k \in \Sigma_o \cap \Sigma_u \Rightarrow \big((\forall j \in [1, k])\, \sigma_j \in \Sigma_u\big)$$

$\Diamond$

Informally, whenever σ_k is observable and uncontrollable, its immediately preceding unobservable events must all be uncontrollable. In other words, its nearest controllable event must be observable.

Having a natural projection with the observer and the OCC property, we provide the following answer (a sufficient condition) to the question above.

Proposition 3.1. ([17, Corollary 2])

For all $i \in I$, if $P|_{\Sigma_i^*}$ is an $L_{m,i}$-observer and OCC for L_i, then

$$K = P^{-1}K_d \cap L_m$$

$\square$

Remark 3.1. Suppose $\Sigma_o \subseteq \Sigma$ is the union of a subcollection of component alphabets, i.e.,

$$\Sigma_o = \dot{\bigcup} \{\Sigma_i | i \in I'\}$$

where $I' \subseteq I$. Then for all $i \in I'$, $P|_{\Sigma_i^*}$ is the identity map:

$$P|_{\Sigma_i^*} : \Sigma_i^* \to \Sigma_i^*$$

It follows that the conditions in Proposition 3.1 hold automatically. In that case, when synthesizing a decentralized supervisor $K_d = sup\mathcal{C}(E \cap PL_m)$, we need only compute the synchronous product of all $L_{m,i}$ $(i \in I')$ to obtain PL_m, rather than project the global model L_m.

Next we turn to the case of more than one specification, for instance two. Given two specification languages $E_j \subseteq \Sigma_{o,j}^*,\ \Sigma_{o,j} \subseteq \Sigma\ (j = 1, 2)$, let $P_j : \Sigma^* \to \Sigma_{o,j}^*$ be the corresponding natural projections. Again by SCT, we obtain the monolithic supervisor by synthesizing the language

$$K := sup\mathcal{C}(P_1^{-1}E_1 \cap P_2^{-1}E_2 \cap L_m) \subseteq \Sigma^*$$

On the other hand, we can obtain two decentralized supervisors by synthesizing the languages

$$K_j := sup\mathcal{C}(E_j \cap P_j L_m) \subseteq \Sigma_{o,j}^*, \quad j = 1, 2$$

Thus the central question is: what condition(s) can guarantee

$$K = P_1^{-1}K_1 \cap P_2^{-1}K_2 \cap L_m$$

Recall that two languages $F_j \subseteq \Sigma_j^*\ (j = 1, 2)$ are called *synchronously nonconflicting* [63] over $(\Sigma_1 \cup \Sigma_2)^*$ if

$$\overline{F_1 \mid\mid F_2} = \bar{F}_1 \mid\mid \bar{F}_2$$

It is known that if $P_j\ (j = 1, 2)$ satisfy the conditions in Proposition 3.1 and K_1 and K_2 are synchronously nonconflicting, then

$$K = P_1^{-1}K_1 \cap P_2^{-1}K_2 \cap L_m$$

However, the computation for checking this synchronous nonconflictingness is as expensive as that for synthesizing the monolithic supervisor. To gain computational efficiency, a promising approach is first to simplify K_1 and K_2 using model abstraction, and then perform the synchronously nonconflicting test on the abstracted level. It is the model abstraction based on natural projections with the observer property that ensures the

validity of this approach.

Let $\Sigma_o \supseteq \Sigma_{o,1} \cap \Sigma_{o,2}$ and $P : \Sigma^* \to \Sigma_o^*$.

Lemma 3.1. ([17, Theorem 1])

Assume $P|_{\Sigma_{o,j}^*}$ is a K_j-observer $(j = 1, 2)$. Then

$$\overline{K_1 \parallel K_2} = \bar{K}_1 \parallel \bar{K}_2$$

if and only if

$$\overline{P|_{\Sigma_{o,1}^*}(K_1) \parallel P|_{\Sigma_{o,2}^*}(K_2)} = \overline{P|_{\Sigma_{o,1}^*}(K_1)} \parallel \overline{P|_{\Sigma_{o,2}^*}(K_2)}$$

□

If K_1 and K_2 fail to be synchronously nonconflicting, a *coordinator* has to be designed to resolve the conflict. It is again the observer property that allows us to design the coordinator on the abstracted level, thus achieving computational efficiency.

Lemma 3.2. ([17, Proposition 7])

Assume $P|_{\Sigma_{o,j}^*}$ is a K_j-observer $(j = 1, 2)$. If there exists a language $L_o \subseteq \Sigma_o^*$ such that

$$\overline{P|_{\Sigma_{o,1}^*}(K_1) \parallel P|_{\Sigma_{o,2}^*}(K_2) \parallel L_o} = \overline{P|_{\Sigma_{o,1}^*}(K_1)} \parallel \overline{P|_{\Sigma_{o,2}^*}(K_2)} \parallel \bar{L}_o$$

then

$$\overline{K_1 \parallel K_2 \parallel L_o} = \bar{K}_1 \parallel \bar{K}_2 \parallel \bar{L}_o$$

□

Finally,

Theorem 3.1. ([14, Proposition 4.10])

If $P|_{\Sigma_{o,j}^*}$ is a K_j-observer $(j = 1, 2)$ and $P|_{\Sigma_i^*}$ is OCC for L_i $(\forall i \in I)$, then there exists

a coordinator language $C \subseteq \Sigma_o^*$ such that

$$\overline{K_1 \parallel K_2 \parallel C} = \bar{K}_1 \parallel \bar{K}_2 \parallel \bar{C}$$

and

$$K = P_1^{-1}K_1 \cap P_2^{-1}K_2 \cap P^{-1}C$$

□

Remark 3.2 (Coordinator Design)**.** Suppose K_1 and K_2 are synchronously conflicting. To design a coordinator to resolve the conflict, we present the following procedure in TCT syntax. Let

$$\begin{aligned} L_{m,abs} &= P|_{\Sigma_{o,1}^*}(K_1) \parallel P|_{\Sigma_{o,2}^*}(K_2) \\ &= P_1^{-1}|_{\Sigma_o^*}(P|_{\Sigma_{o,1}^*}K_1) \cap P_2^{-1}|_{\Sigma_o^*}(P|_{\Sigma_{o,2}^*}K_2) \ \subseteq \Sigma_o^* \end{aligned}$$

be the marked language of the abstract-level plant, and let $E_{abs} = \Sigma_o^*$ be the abstract-level specification language.

1. First remove all of the blocking states in the abstract-level plant generator:

$$K_{abs} = \ \textbf{supcon}\ (L_{m,abs}, E_{abs})$$

2. Next create the *control data file* showing the disablement information corresponding to the removal of the blocking states:

$$K_{abs}.\text{DAT} = \ \textbf{condat}\ (L_{m,abs}, K_{abs})$$

3. Lastly project the abstract-level plant model out of its nonblocking counterpart, based on the disablement information from the control data file as well as the

marking information:

$$C = \textbf{supreduce}\ (L_{m,abs}, K_{abs}, K_{abs}.\mathrm{DAT})$$

3.3 Problem Formulation and Solution

First we formulate the distributed control problem, denoted by (⋇).

Consider a plant $\mathbf{G}$ consisting of component agents $\mathbf{G}_i$ defined over pairwise disjoint alphabets Σ_i ($i \in I$, I an index set.) Thus $\mathbf{G}$ is defined over the alphabet

$$\Sigma = \dot{\bigcup}\{\Sigma_i | i \in I\}$$

Let $L_i := L(\mathbf{G}_i)$ and $L_{m,i} := L_m(\mathbf{G}_i)$; then the closed and marked languages of $\mathbf{G}$ are

$$L := L(\mathbf{G}) = ||\{L_i | i \in I\} \quad \text{and} \quad L_m := L_m(\mathbf{G}) = ||\{L_{m,i} | i \in I\}$$

For simplicity we assume $\mathbf{G}_i$ is nonblocking (i.e. $\bar{L}_{m,i} = L_i$), for all $i \in I$. Then $\mathbf{G}$ is necessarily nonblocking (i.e. $\bar{L}_m = L$).

The independent agents are implicitly coupled through an imposed specification language E that (as usual) imposes a behavioral constraint on $\mathbf{G}$. Assume E is decomposable into component specifications $E_j \subseteq \Sigma_{o,j}^*$ ($j \in J$, J an index set), where the $\Sigma_{o,j} \subseteq \Sigma$ need not be pairwise disjoint; namely,

$$E = ||\{E_j | j \in J\}$$

Thus E is defined over the alphabet

$$\Sigma_o := \bigcup\{\Sigma_{o,j} | j \in J\}$$

Let $P_o : \Sigma^* \to \Sigma_o^*$ be the corresponding natural projection.

Given a control problem with the plant and the specification as described above, by SCT the monolithic supervisor **SUP** can be analytically expressed as the language

$$K := L_m(\mathbf{SUP}) = sup\mathcal{C}(P_o^{-1}E \cap L_m) \subseteq \Sigma^*$$

We are interested in designing a set of local controllers

$$\mathbf{LOC} = \{\mathbf{LOC}_i \text{ over } \Sigma | i \in I\}$$

one for each agent $\mathbf{G}_i$, which realizes performance identical with that achieved by **SUP**. Formally let $C_i := L_m(\mathbf{LOC}_i)$. Then

$$C := L_m(\mathbf{LOC}) = \bigcap\{C_i | i \in I\} \subseteq \Sigma^*$$

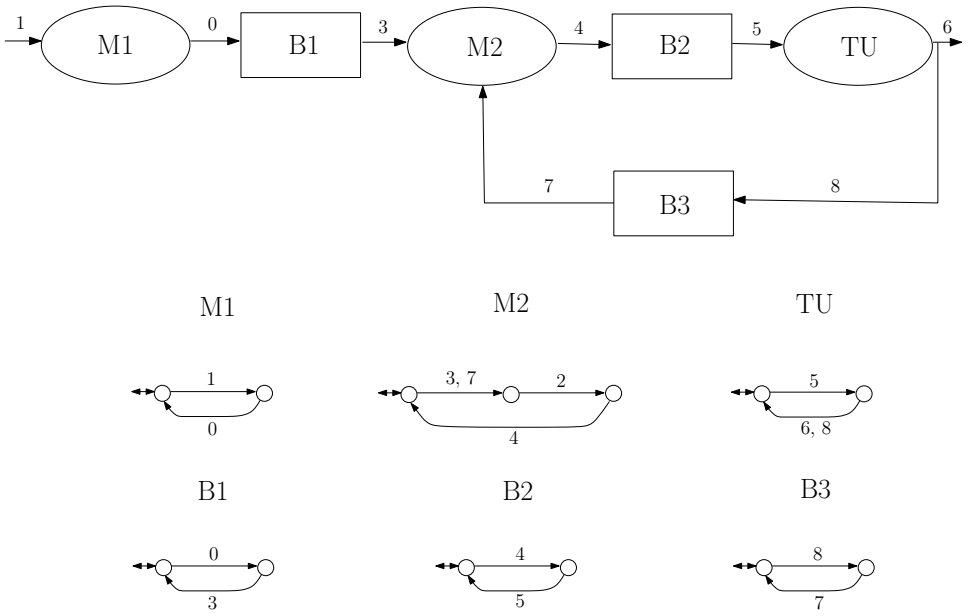
We require

$$K = C \cap L_m \quad \text{and} \quad \bar{K} = \bar{C} \cap L$$

This problem (⋇) has been solved in Chapter 2, for small-scale DES, by a direct localization on **SUP**. However, for large-scale DES, owing to the bottleneck of state explosion, it is computationally expensive even if still possible to synthesize **SUP** in the first place. Therefore, to tackle the distributed control problem of large-scale DES we employ a divide and conquer strategy, combining supervisor localization (Chapter 2) with modular control theory (Subsection 3.2.3): first design a hierarchy of decentralized supervisors and coordinators which realizes performance identical with that achieved by **SUP**; then localize each of these modular supervisors, generally of small state size, to local controllers for the relevant agents.

Example 3.1.

Consider a transfer line plant consisting of three independent agents: **M1**, **M2**, and



TU, defined over disjoint alphabets $\Sigma_1 = \{0, 1\}$, $\Sigma_2 = \{2, 3, 4, 7\}$, and $\Sigma_3 = \{5, 6, 8\}$, respectively. Thus the overall alphabet is

$$\Sigma = \Sigma_1 \dot{\cup} \Sigma_2 \dot{\cup} \Sigma_3 = \{0, 1, 2, 3, 4, 5, 7, 8\}$$

and the closed and marked languages of the overall plant are

$$L = L(\mathbf{M1})||L(\mathbf{M2})||L(\mathbf{TU}) \quad \text{and} \quad L_m = L_m(\mathbf{M1})||L_m(\mathbf{M2})||L_m(\mathbf{TU})$$

The specification imposed on this plant is that the three buffers – **B1**, **B2**, and **B3** – must be protected against underflow and overflow. The component specifications are expressed by the languages $L_m(\mathbf{B1})$, $L_m(\mathbf{B2})$, and $L_m(\mathbf{B3})$; they are defined over the alphabets $\Sigma_{o,1} = \{0, 3\}$, $\Sigma_{o,2} = \{4, 5\}$, and $\Sigma_{o,3} = \{7, 8\}$, respectively. So the overall specification language is

$$E = L_m(\mathbf{B1})||L_m(\mathbf{B2})||L_m(\mathbf{B3})$$

and is defined over the alphabet

$$\Sigma_o = \Sigma_{o,1} \dot{\cup} \Sigma_{o,2} \dot{\cup} \Sigma_{o,3} = \{0, 3, 4, 5, 7, 8\}$$

Let $P_o : \Sigma^* \to \Sigma_o^*$ be the corresponding natural projection.

According to SCT, the monolithic supervisor **SUP** for this control problem is the language

$$K := L_m(\mathbf{SUP}) = sup\mathcal{C}(P_o^{-1}E \cap L_m) \subseteq \Sigma^*$$

We are to design a set of local controllers

$$\mathbf{LOC} = \{\mathbf{LOC}_i \text{ over } \Sigma | i \in \{1, 2, 3\}\}$$

one for each agent, which realizes performance identical to that achieved by **SUP**. Let $C_i = L_m(\mathbf{LOC}_i)$, and thus

$$C := L_m(\mathbf{LOC}) = \bigcap\{C_i | i \in \{1, 2, 3\}\} \subseteq \Sigma^*$$

We require

$$K = C \cap L_m \quad \text{and} \quad \bar{K} = \bar{C} \cap L_m \qquad \blacklozenge$$

To solve the above distributed control problem (⋇), we now present a systematic procedure consisting of seven steps. We call it the *decomposition-aggregation supervisor localization procedure* (DASLP).

Step 1: Plant Model Abstraction

Part of the plant dynamics that is unrelated to the proposed specification may be concealed. By hiding irrelevant transitions, we can simplify the model of the plant components. The procedure for this step is the following.

1. Ensure the OCC property: for each $\sigma \in \Sigma_o \cap \Sigma_u$, add its nearest upstream control-

lable events to Σ_o. Call the augmented alphabet Σ'_o, and let $P'_o : \Sigma^* \to (\Sigma'_o)^*$.

2. Check if $P'_o|_{\Sigma_i^*}$ is an $L_{m,i}$-observer, for all $i \in I$. If yes, jump to (4).

3. Employ the MX algorithm to compute a reasonable extension of Σ'_o that does define an $L_{m,i}$-observer, for all $i \in I$ [1]. Denote the extended alphabet again by Σ'_o, and the corresponding natural projection again by P'_o.

4. Compute model abstractions for each component, denoted by $\mathbf{G}'_i$, with closed and marked languages

$$L'_i := P'_o|_{\Sigma_i^*}(L_i) \quad \text{and} \quad L'_{m,i} := P'_o|_{\Sigma_i^*}(L_{m,i})$$

Note that $\mathbf{G}'_i$ is defined over $\Sigma'_i := \Sigma_i \cap \Sigma'_o$.

Example 3.1 (Continued)**.**

We follow the procedure presented above.

1. Ensure the OCC property. For this problem, we have

$$\Sigma_o \cap \Sigma_u = \{0, 4, 8\}$$

For these three events, their nearest upstream controllable event sets are $\{1\}$, $\{3, 7\}$, and $\{5\}$, respectively. While events 3, 5 and 7 already belong to Σ_o, event 1 does not. Hence we add event 1 to Σ_o, thus obtaining

$$\Sigma'_o = \{0, 1, 3, 4, 5, 7, 8\}$$

[1] It is important to note that, for those $P'_o|_{\Sigma_i^*}$ $(i \in I)$ that are already $L_{m,i}$-observers, extending Σ'_o will not destroy their observer property. For example, assume that $P'_o|_{\Sigma_1^*} : \Sigma_1^* \to (\Sigma_1 \cap \Sigma'_o)^*$ is an $L_{m,1}$-observer and that $P'_o|_{\Sigma_2^*}$ is not an $L_{m,2}$-observer. It is by adding only certain events in Σ_2 that we extend Σ'_o in order to make $P'_o|_{\Sigma_2^*}$ an $L_{m,2}$-observer. It follows from the disjointness between Σ_1 and Σ_2 that the codomain of $P'_o|_{\Sigma_1^*}$, $(\Sigma_1 \cap \Sigma'_o)^*$, remains unchanged, and therefore the observer property of $P'_o|_{\Sigma_1^*}$ is not affected.

Let $P'_o : \Sigma^* \to (\Sigma'_o)^*$. Note that only events 2 and 6 are nulled by P'_o.

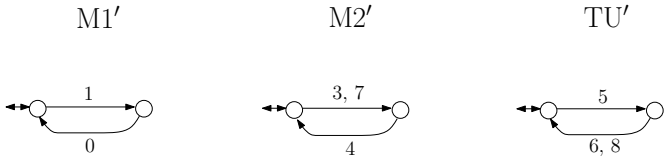
2. Check for the observer property. While $P'_o|_{\Sigma_1^*}$ and $P'_o|_{\Sigma_2^*}$ are an $L_m(\mathbf{M1})$-observer and $L_m(\mathbf{M2})$-observer, respectively, $P'_o|_{\Sigma_3^*}$ is *not* an $L_m(\mathbf{TU})$-observer.

3. Employ the MX algorithm to compute a reasonable extension of Σ'_o; the algorithm terminates with adding event 6 to Σ'_o. By inspecting the model of **TU**, event 6 is a transition from a marked state to an unmarked state; projecting it out will cause structural nondeterminism of the canonical reduction. Denote the extended alphabet again by

$$\Sigma'_o = \{0, 1, 3, 4, 5, 6, 7, 8\}$$

and the corresponding natural projection again by P'_o. Notice that only event 2 is nulled by P'_o.

4. Compute model abstractions for each agent by using $P'_o|_{\Sigma_i^*}$ ($i \in \{1, 2, 3\}$); the abstracted models are displayed below. Notice that only the model of **M2** is simplified, by projecting out event 2.



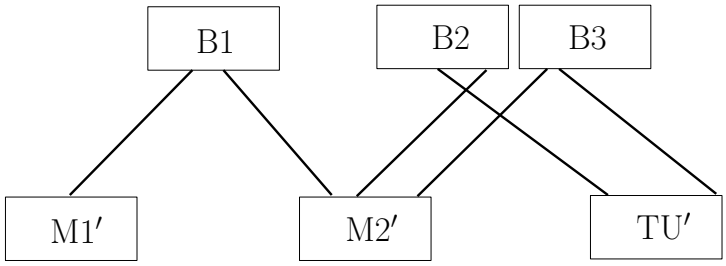
♦

Step 2: Decentralized Supervisor Synthesis

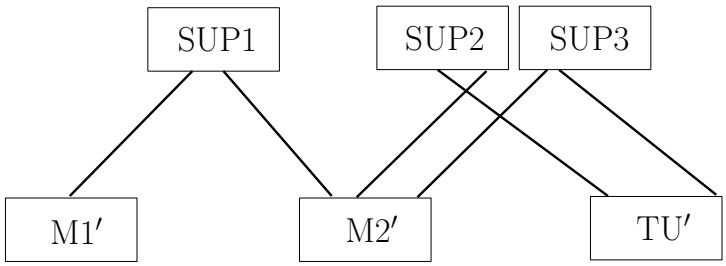
After step one, the system consists of component abstractions and specifications. We group for each specification $E_j \subseteq \Sigma_{o,j}^*$ its event-coupled component abstractions: those sharing events with E_j (i.e. $\Sigma'_i \cap \Sigma_{o,j} \neq \emptyset$.) Then for each group, we synthesize a decentralized supervisor.

Example 3.1 (Continued)**.**

We first group for each buffer specification its event-coupled component abstractions. The grouping is displayed as follows, with solid lines denoting event-coupling.



Then we compute a decentralized supervisor for each group, and in the figure above replace the specifications with the supervisors.



	State #	Reduced State #
SUP1	9	2
SUP2	8	2
SUP3	9	2

♦

Step 3: Subsystem Decomposition and Coordination

After step two, the system has several modules, each of which consists of a decentralized supervisor with associated component abstractions. We decompose the overall system into small-scale subsystems, through grouping these modules based on their interconnection dependencies (e.g., event-coupling). If these modules admit certain special structures, an effective approach for decomposition is *control-flow nets* [14, Chapter 2].

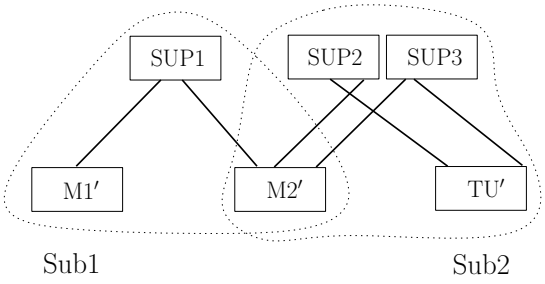
Having obtained a group of simple subsystems, we perform a nonconflicting check to verify the nonblocking property for each subsystem. If a subsystem fails to be nonblocking, we design a coordinator by using the method presented in Remark 3.2.

Example 3.1 (Continued)**.**

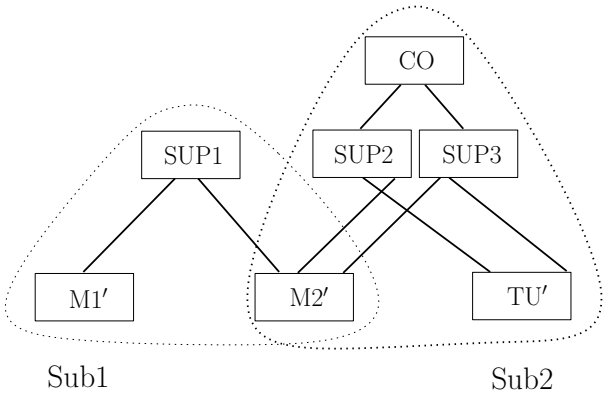
We have three decentralized supervisors, thus three modules.

- Module 1: **SUP1**, **M1**′, and **M2**′
- Module 2: **SUP2**, **M2**′, and **TU**′
- Module 3: **SUP3**, **M2**′, and **TU**′

We group these modules, according to their event-coupling relation, into two subsystems.



We verify the nonblocking property for each subsystem. Since Subsystem 1 has a single supervisor, it is necessarily nonblocking. In Subsystem 2, **SUP**2 and **SUP**3 turn out to be conflicting; we design a coordinator to ensure the nonblockingness of this subsystem.



	State #	Reduced State #
CO	6	2

♦

Step 4: Subsystem Model Abstraction

After step three, the system consists of several nonblocking subsystems. We now need to verify the nonconflicting property among these subsystems. Directly applying nonconflicting checks requires expensive computation; instead, we again bring in the model abstraction technique to simplify every subsystem, and test the nonconflictingness on the abstracted level. The procedure of this step is analogous to that of step one.

1. Determine the shared event set, denoted by Σ_{sub}, of these subsystems. Let $P_{sub} : (\Sigma'_o)^* \to (\Sigma_{sub})^*$ be the corresponding natural projection.

2. Ensure the OCC property: for each $\sigma \in \Sigma_{sub} \cap \Sigma_u$, add its nearest upstream controllable events to Σ_{sub}. Call the augmented alphabet Σ'_{sub}. Let $P'_{sub} : (\Sigma'_o)^* \to (\Sigma'_{sub})^*$.

3. Check if P'_{sub} is an observer for each subsystem. If yes, jump to (4).

4. Employ the MX algorithm to compute a reasonable extension of Σ'_{sub} that does define an observer for each subsystem. Denote the extended alphabet again by Σ'_{sub}, and the corresponding natural projection again by P'_{sub}.

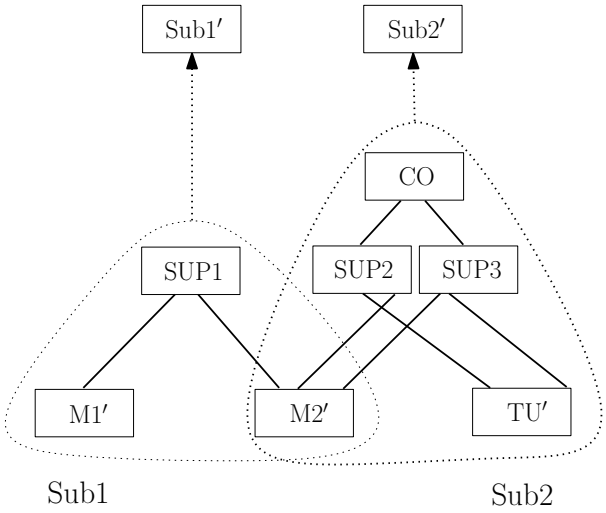
5. Compute model abstractions for each subsystem with P'_{sub}.

Example 3.1 (Continued)**.**

The two nonblocking subsystems **Sub**1 and **Sub**2 share the events of $\mathbf{M}2'$, and thus $\Sigma_{sub} = \{3, 4, 7\}$. Σ_{sub} is already OCC, but is not an observer for either subsystem. So we employ the MX algorithm to compute a reasonable extension of Σ_{sub}; the algorithm terminates with adding events $1, 6$, and 8 to Σ_{sub}. Denote the extended alphabet

$$\Sigma'_{sub} = \{1, 3, 4, 6, 7, 8\}$$

and the corresponding natural projection by P'_{sub}, with which we compute the subsystem model abstractions.



	Sub1	**Sub**1′	**Sub**2	**Sub**2′
State #	9	4	6	5

♦

Step 5: Abstracted Subsystem Decomposition and Coordination

After step four, we obtain several subsystem abstractions. We group these abstractions according to their interconnection dependencies (e.g. event-coupling). If these abstractions admit certain special structures, *control-flow nets* can again be applied.

Next for each group of subsystem abstractions, we perform a nonconflicting check to verify the nonblocking property. If a group fails to be nonblocking, we design a coordinator by using the method presented in Remark 3.2.

Example 3.1 (Continued)**.**

We have only two subsystem abstractions **Sub**1′ and **Sub**2′ left. Grouping them together, we check the nonblockingness: **Sub**1′ and **Sub**2′ turn out to be nonconflicting, and thus the group is nonblocking. ♦

Step 6: Higher-Level Abstraction

Repeat steps four and five until there remains a single group of subsystem abstractions in step five.

Step 7: Localization

The modular supervisory control design terminates at step six; we have obtained a hierarchy of decentralized supervisors and coordinators. We now apply the supervisor localization algorithm to localize each of these supervisors/coordinators to local controllers for the relevant agents.

To determine which agents are related to a supervisor/coordinator, we introduce the *control-coupling* relation. Given a plant $\mathbf{G} = (Y, \Sigma, \eta, y_0, Y_m)$, a component agent $\mathbf{G}_i = (\_, \Sigma_{c,i} \dot{\cup} \Sigma_{u,i}, \_, \_, \_)$ ($i \in I$, I an index set), and a supervisor $\mathbf{SUP} = (X, \Sigma, \xi, x_0, X_m)$, recall the function $D_i : X \to Pwr(\Sigma_{c,i})$, which was defined according to

$$D_i(x) = \{\sigma \in \Sigma_{c,i} | \neg \xi(x, \sigma)! \ \& \ (\exists s \in \Sigma^*)[\xi(x_0, s) = x \ \& \ \eta(y_0, s\sigma)!]\}$$

$D_i(x)$ is the set of controllable events in $\Sigma_{c,i}$ that must be disabled at x. Thus we say $\mathbf{G}_i$ is control-coupled to **SUP** if

$$(\exists x \in X) \ D_i(x) \neq \emptyset$$

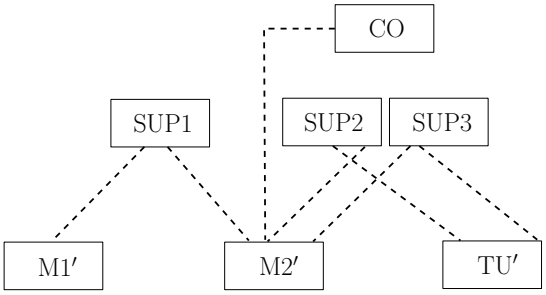
In other words, some controllable event(s) of $\mathbf{G}_i$ must be disabled at some state(s) of **SUP**. To determine the control coupling relation, we simply look up the table generated by **condat** in TCT [62].

Finally, we localize each supervisor/coordinator for its control-coupled components.

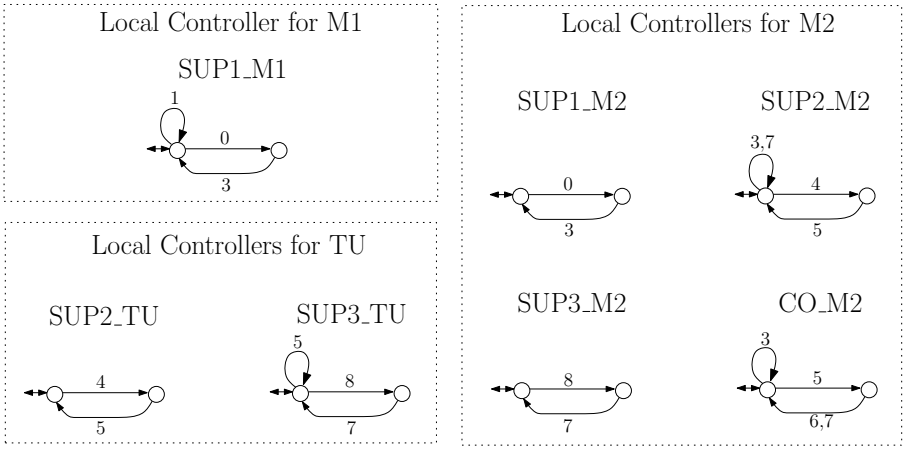
Example 3.1 (Continued)**.**

The modular supervisory control design generates three decentralized supervisors and one coordinator: $\mathbf{SUP}1, \mathbf{SUP}2, \mathbf{SUP}3$, and **CO**. For each of them, we determine their control-coupled components by looking up the corresponding **condat** tables. We show

the result below, with dashed lines denoting the control-coupling relation.



Along these dashed lines we apply the localization algorithm, with the results displayed below.



♦

Finally,

Theorem 3.2.

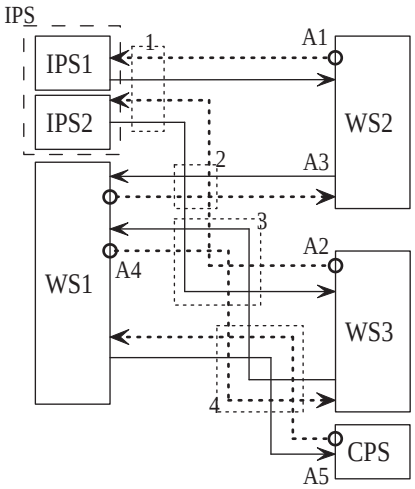
DASLP solves (⋇).

Proof. We sketch the proof as follows. At the end of step six of DASLP, we obtain a hierarchy of decentralized supervisors and coordinators. It has been proved [14] that the concurrent behavior of these modular supervisors is identical to the controlled behavior of the monolithic optimal nonblocking supervisor (for the special case of two specifications see Theorem 3.1; for a more general case see [17, Theorem 4].) Then in step seven, these modular supervisors are decomposed into local controllers for the relevant agents; the

identity, between the concurrent behavior of these local controllers and the concurrent behavior of the modular supervisors, is guaranteed by Proposition 2.1 and Theorem 2.1 in Chapter 2. Therefore by transitivity, we conclude that the concurrent behavior of the local controllers is identical to the controlled behavior of the monolithic optimal nonblocking supervisor. □

3.4 Example: Distributed Control of AGV System

We apply the decomposition-aggregation supervisor localization procedure to solve the distributed control problem of automatic guided vehicles (AGVs) serving a manufacturing workcell, adapted from [63].



The workcell consists of two input stations IPS1, IPS2 for parts of types 1, 2; three workstations WS1, WS2, WS3; and one completed parts station CPS. A team of five independent AGVs – AGV1,...,AGV5 – travel in fixed criss-crossing routes, loading/unloading and transporting parts in the cell. We model the AGV system as the plant to be controlled, on which three types of control specifications are imposed: the mutual exclusion (i.e. single occupancy) of shared zones, the capacity limit of workstations, and the mutual exclusion of the shared loading area of the input stations. The generator models of plant components and specifications are displayed in Figs. 3.1 and 3.2, respectively; readers are referred to [63, Section 4.7] for the detailed semantic description of events.

While the standard centralized approach generates a monolithic supervisor of 4406 states [63, Section 4.7], our distributed control objective is to design for each AGV a set of local strategies which as a whole realize performance identical to that achieved by the global supervisor.

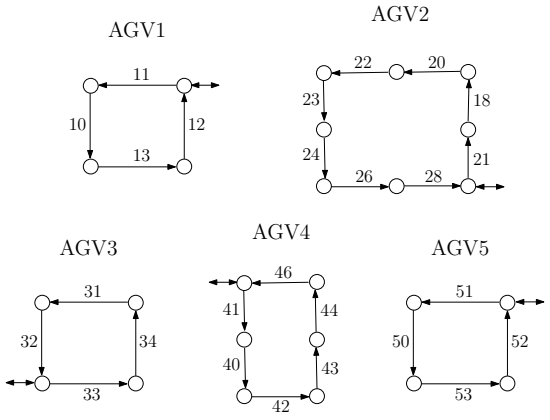


Figure 3.1: Generators of plant components

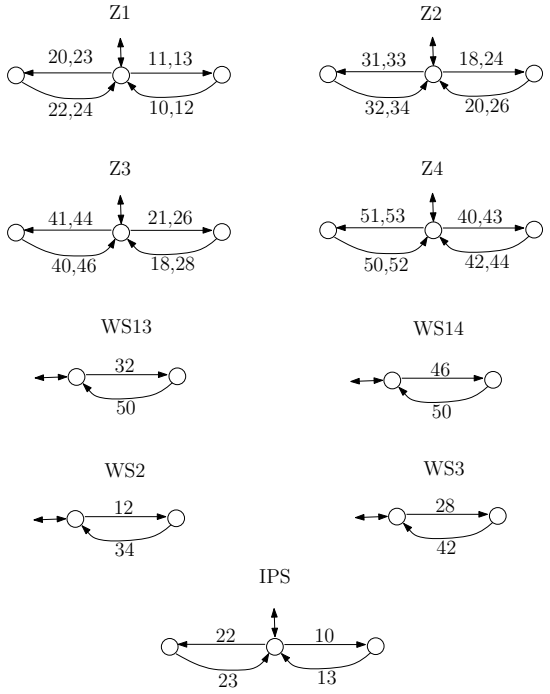


Figure 3.2: Generators of specifications

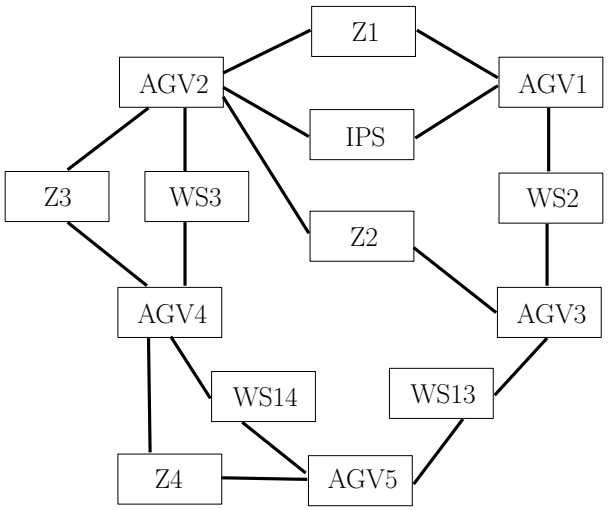
Step 1: Plant Model Abstraction

Let Σ and Σ_o denote the alphabets on which the overall plant and the overall specifi-

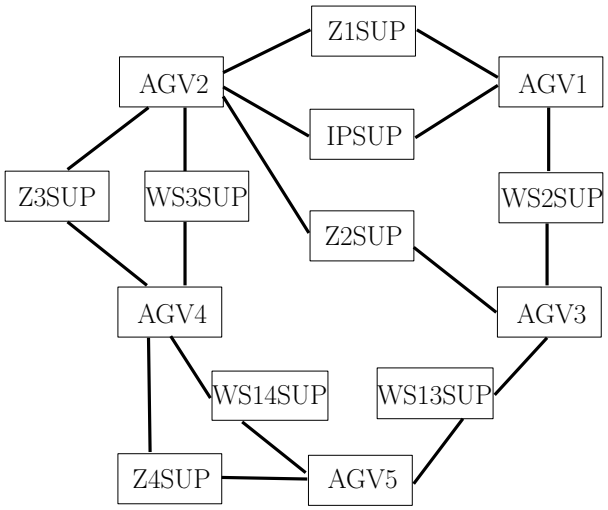
cation are defined. One can verify that $\Sigma = \Sigma_o$ in this example. Namely, all of the plant dynamics are crucial for the subsequent synthesis, and therefore no plant model can be simplified in this step.

Step 2: Decentralized Supervisor Synthesis

We group for each specification its event-coupled AGVs. The grouping is displayed as follows, with solid lines denoting event-coupling.



Then we compute a decentralized supervisor for each group, and in the figure above replace the specifications with the supervisors. In addition, the state sizes of these supervisors are listed in Table 3.1.

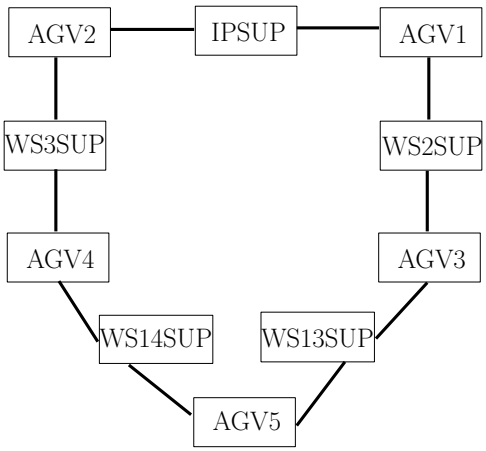


	State #	Reduced State #
Z1SUP	24	2
Z2SUP	24	2
Z3SUP	36	2
Z4SUP	18	2
WS13SUP	24	2
WS14SUP	34	2
WS2SUP	24	2
WS3SUP	62	2
IPSUP	24	2

Table 3.1: State sizes of decentralized supervisors

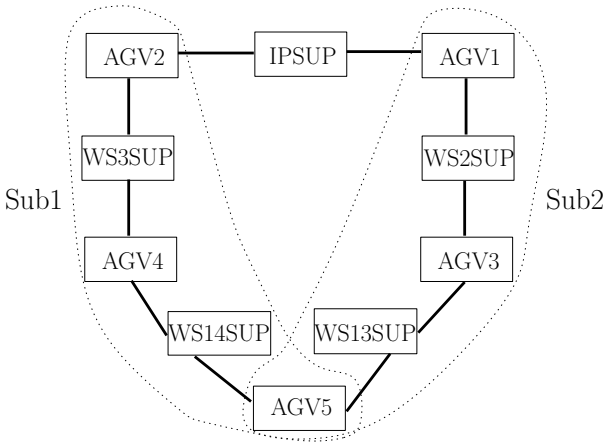
Step 3: Subsystem Decomposition and Coordination

We have nine decentralized supervisors, thus nine modules. The interconnection structure of these modules can be simplified by applying the control-flow nets approach. Specifically, the decentralized supervisors for the four zones – **Z1SUP** to **Z4SUP**, are *harmless* to the overall nonblocking property, and hence can be safely removed from the interaction structure [14, Section 4.6].



In the above simplified structure, there are two paths – **AGV1**, **WS2SUP**, **AGV3**, **WS13SUP** on the right and **AGV2**, **WS3SUP**, **AGV4**, **WS14SUP** on the left – that process workpieces of types 1 and 2, respectively. Thus the system can be naturally

decomposed into the following two subsystems.

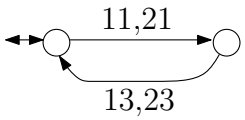


It is further verified that both subsystems are nonblocking on their own.

Step 4: Subsystem Model Abstraction

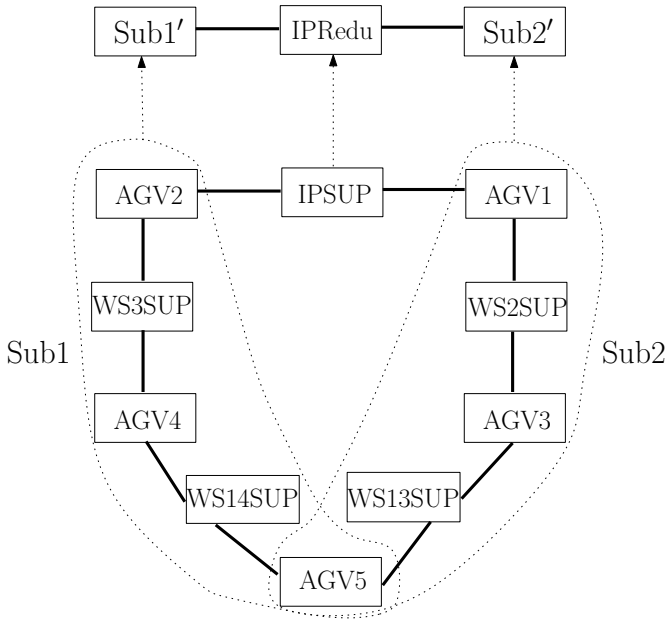
We now need to verify the nonconflicting property among the two subsystems **Sub1**, **Sub2**, and the decentralized supervisor **IPSUP**. First, we determine their shared event set, denoted by Σ_{sub}. **Sub1** and **Sub2** share all of the events in **AGV5**: 50, 51, 52, and 53. For the decentralized supervisor **IPSUP**, we consider its reduced generator:

IPRedu



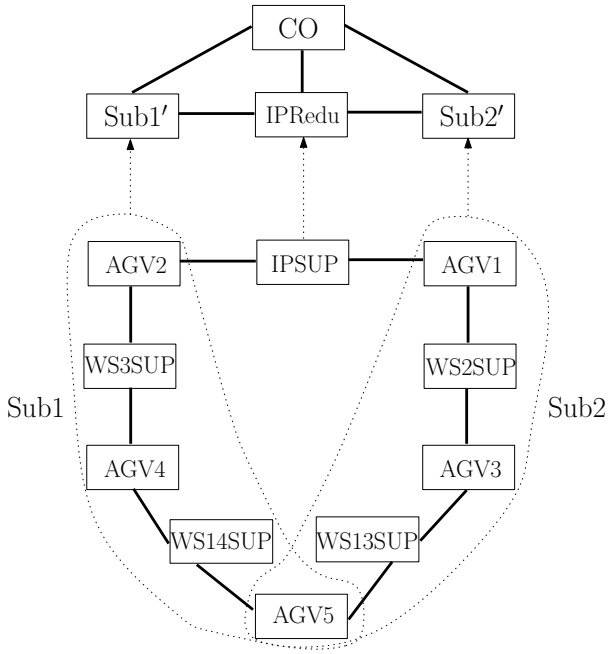
IPRedu share events 11, 13 with **Sub1**, and 21, 23 with **Sub2**. Thus we set $\Sigma_{sub} = \{11, 13, 21, 23, 50, 51, 52, 53\}$. It can then be verified that the corresponding natural projection $P_{sub} : \Sigma^* \to \Sigma^*_{sub}$ does enjoy the observer and OCC property. So that with P_{sub}, we can compute the subsystem model abstractions.

	Sub1	**Sub**1′	**Sub**2	**Sub**2′
State #	140	30	330	30



Step 5: Abstracted Subsystem Decomposition and Coordination

We treat **Sub1′**, **Sub2′**, and **IPRedu** as a single group, and directly check the non-blocking property. This group turns out to be blocking; a coordinator then has to be designed to resolve the conflict.



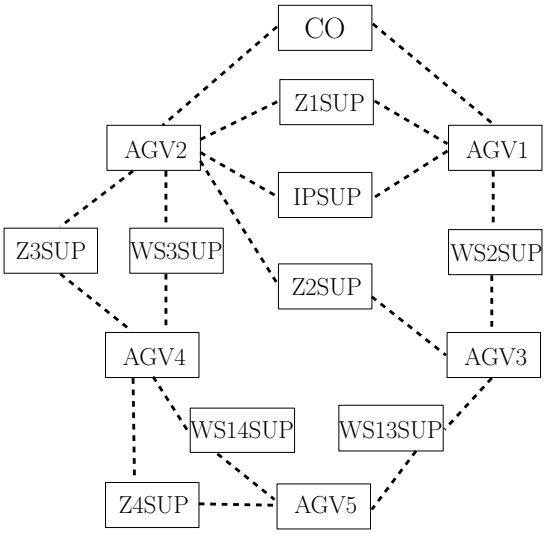
	State #	Reduced State #
CO	165	7

Step 6: Higher-Level Abstraction

The modular supervisory control design finishes at the last step.

Step 7: Localization

We start with determining the control-coupling relation through looking up the **con-dat** tables of each decentralized supervisor and the coordinator. We show the result below, with dashed lines denoting the control-coupling.



Notice that the coordinator is control-coupled only to **AGV1** and **AGV2**. Along these dashed lines, we apply the supervisor localization algorithm. The state sizes of the resultant local controllers are listed in Table 3.2, and the generator models of each controller are displayed in Figs. 3.3–3.7 (for clarity irrelevant selfloops are omitted), grouped with respect to individual AGVs. Thus we have established a purely distributed control architecture, wherein each of the AGV 'robots' pursues its independent 'lifestyle', while being coordinated implicitly with its fellows through their local shared observable events.

	AGV1 (#)	AGV2 (#)	AGV3 (#)	AGV4 (#)	AGV5 (#)
Z1SUP	Z1_1 (2)	Z1_2 (2)			
Z2SUP		Z2_2 (2)	Z2_3 (2)		
Z3SUP		Z3_2 (2)		Z3_4 (2)	
Z4SUP				Z4_4 (2)	Z4_5 (2)
WS13SUP			WS13_3 (2)		WS13_5 (2)
WS14SUP				WS14_4 (2)	WS14_5 (2)
WS2SUP	WS2_1 (2)		WS2_3 (2)		
WS3SUP		WS3_2 (2)		WS3_4 (2)	
IPSUP	IP_1 (2)	IP_2 (2)			
CO	CO_1 (7)	CO_2 (7)			

Table 3.2: Local controllers with state sizes

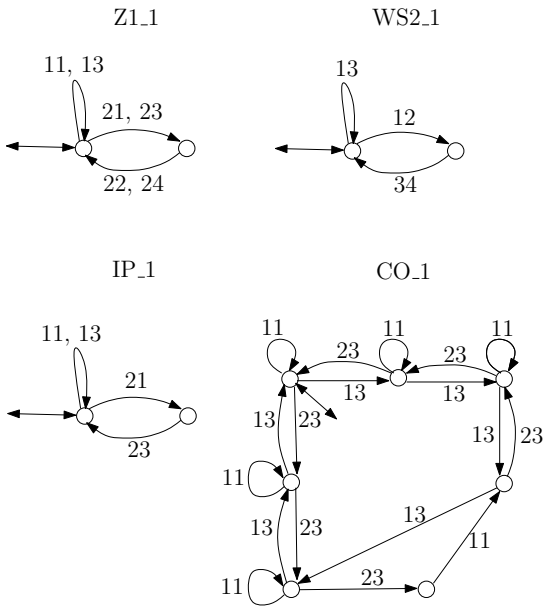


Figure 3.3: Local controllers for AGV1

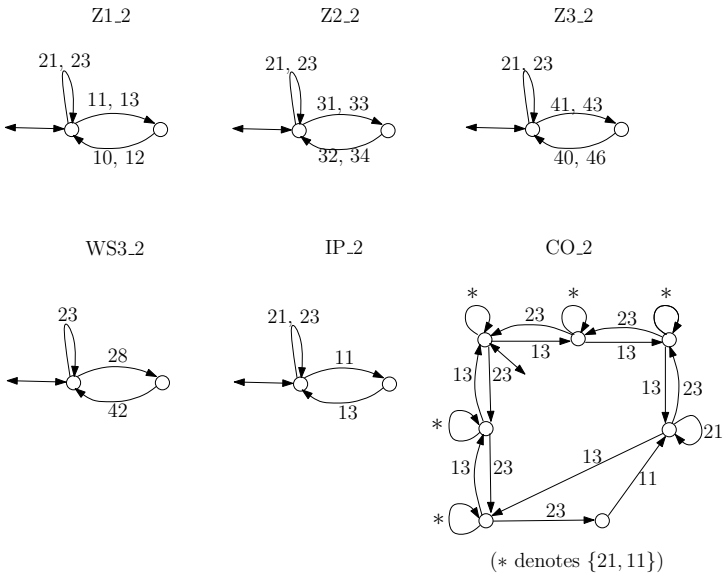


Figure 3.4: Local controllers for AGV2

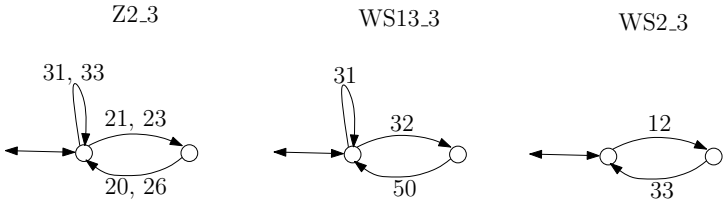


Figure 3.5: Local controllers for AGV3

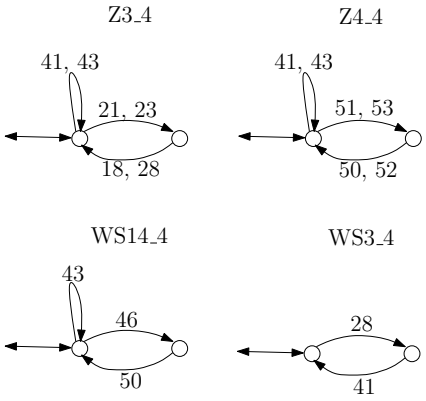


Figure 3.6: Local controllers for AGV4

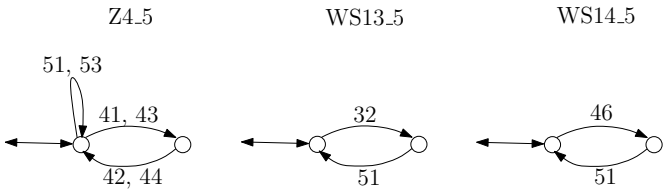
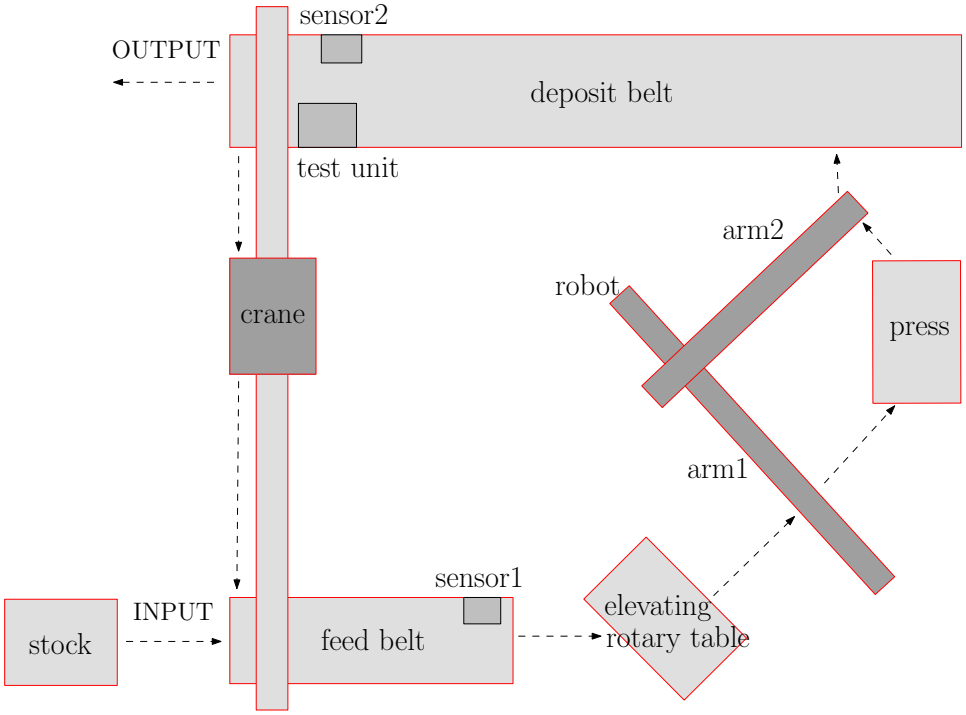


Figure 3.7: Local controllers for AGV5

3.5 Example: Distributed Control of Production Cell

As our second example of distributed control, we consider the following production cell problem taken from [15].



The cell consists of nine individual components: a stock, a feed belt, an elevating rotary table, a robot, arm 1, arm 2, a press, a deposit belt, and a crane. A work cycle of this cell is described as follows:

1. the stock inputs blanks to the system on the feed belt;
2. the feed belt forwards the blanks to the elevating rotary table;
3. the table lifts and rotates the blanks to the position where arm1 of the robot picks them up;
4. arm1 retracts/extends its length and meanwhile the robot rotates to the press, so that arm1 transfers the blanks to the press;
5. the blanks will be forged by the press;

6. after being forged, the blanks are picked up by arm2 of the robot;

7. arm2 retracts/extends its length and meanwhile the robot rotates to the deposit belt, so that arm2 transfers the forged blanks to the deposit belt;

8. the deposit belt forwards the blanks to the end point where a test unit is installed to measure if the forging is successful;

9. if a blank passes the test, it will be output from the system; otherwise, it will be picked up by the crane and moved to the feed belt for another forge.

The generator models of plant components and specifications are displayed in Figs. 3.8 and 3.9, respectively; readers are referred to [15, Section 4] for the detailed semantic description of events. According to the plant models in Fig. 3.8, this problem has state size at least of order 10^7. So it is rather computationally expensive even if still possible to synthesize the monolithic supervisor. Nevertheless, we will see that, by applying the decomposition-aggregation supervisor localization procedure, the largest state size we will encounter in computation is only of order 10^3, and the resulting local controllers as a whole are guaranteed to realize optimal nonblocking control.

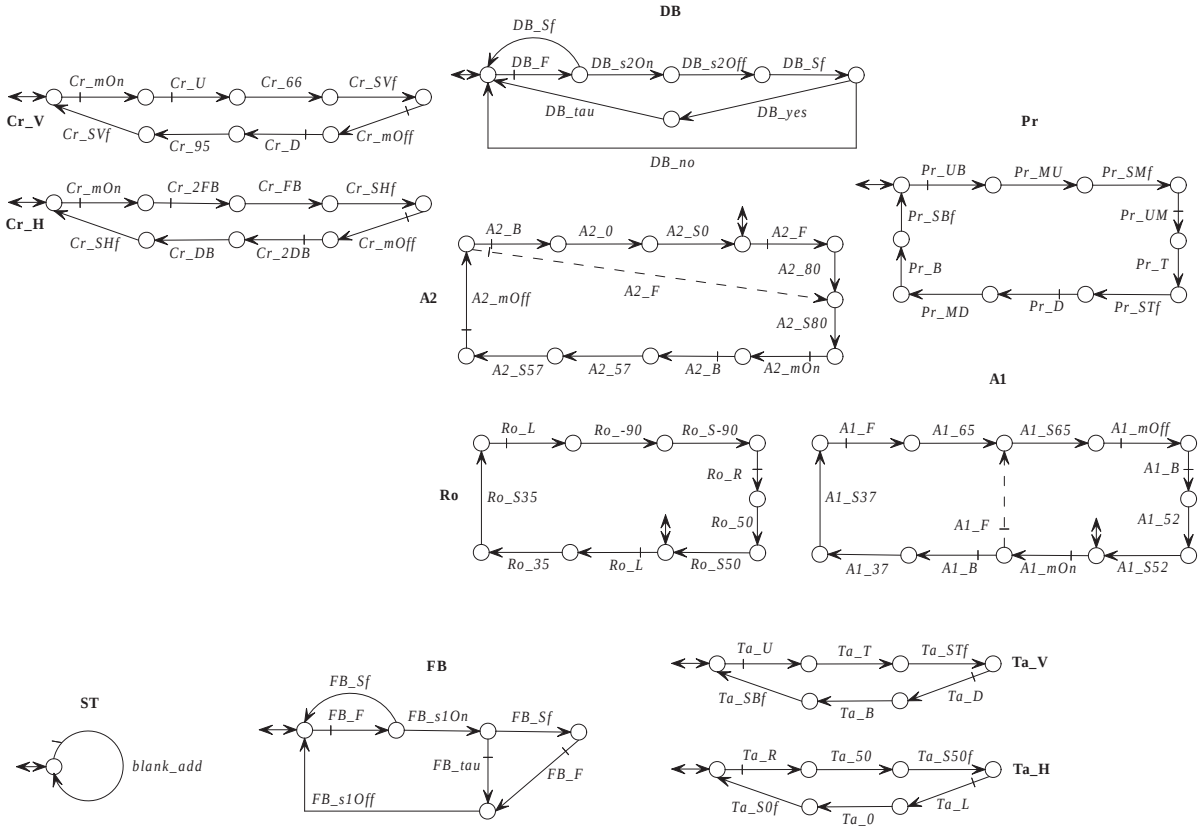


Figure 3.8: Generators of plant components

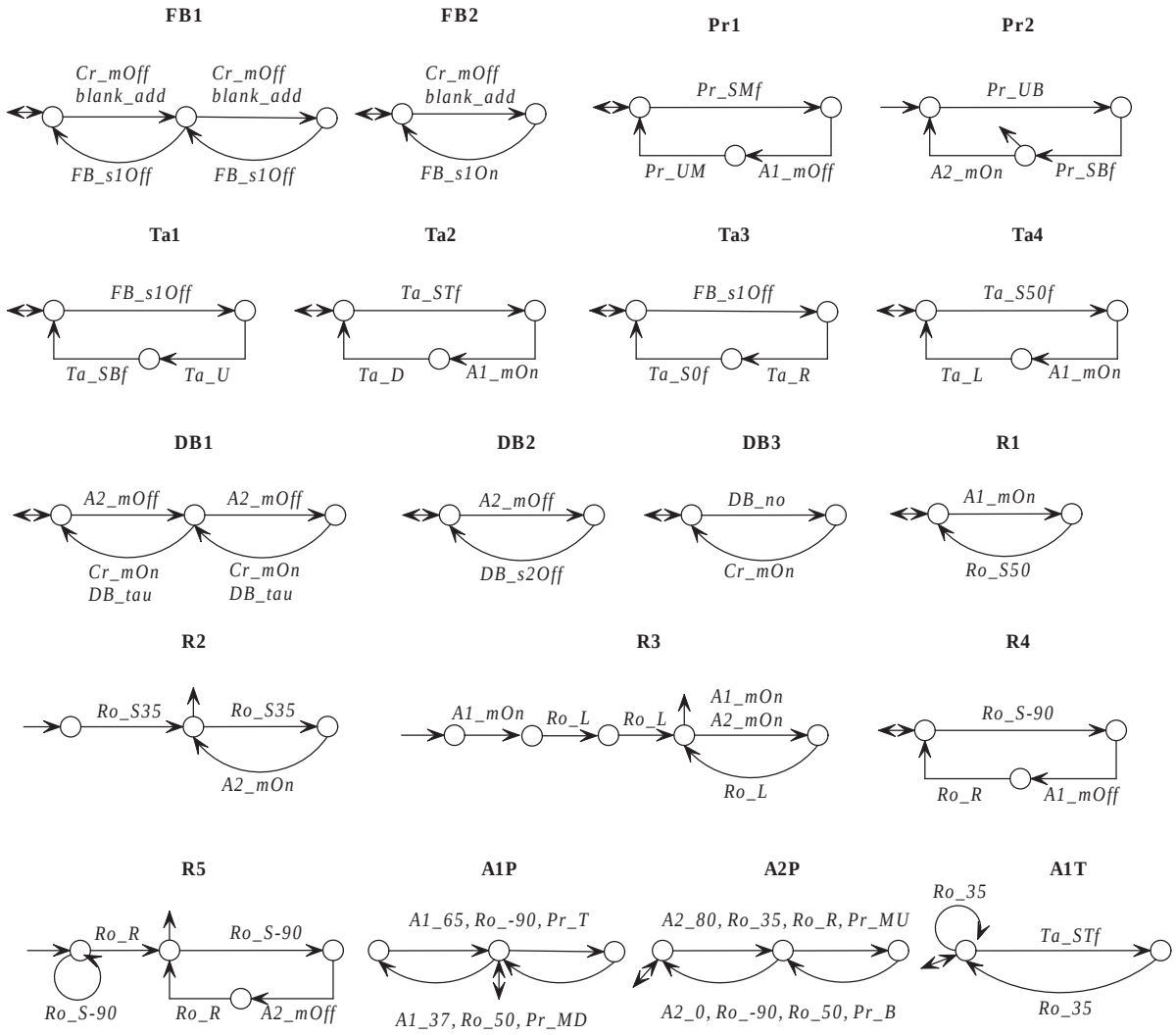
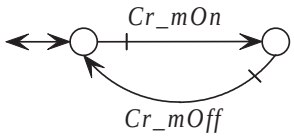


Figure 3.9: Generators of specifications

Step 1: Plant Model Abstraction

Unlike the previous AGV system, in this production cell model abstraction can effectively simplify the components' generators. Take the model of the crane, **Cr=Cr_V || Cr_H** (Fig. 3.8), for example. **Cr** is related to the specifications **DB1, DB2** via *Cr_mOn*, and to **FB1, FB2** via *Cr_mOff*. One can verify that the event set {*Cr_mOn, Cr_mOff*} defines a natural projection that is OCC and an observer for **Cr**. Other transitions in **Cr** are irrelevant to the subsequent control synthesis, and hence can be projected out. The model abstraction of the crane is the simple generator:



Similarly, we compute the model abstractions for other components, and show the result in Fig. 3.10. For economical display, we use numbers to label events; the correspondences with the original labels are listed in Table 3.5. Also note that three modifications have been made to the original models: (1) in **DB**, event DB_tau which was uncontrollable is set to be controllable (labeled 63); (2) in **A1**, event $A1\_F$ on the dashed line is distinguished from that on the solid line, with a new label $A1\_F'$ (89); (3) in **A2**, event $A2\_F$ on the dashed line is distinguished from that on the solid line, with a new label $A2\_F'$ (99). It will be shown that these moderate alterations make the resulting control logic more transparent than that in [15].

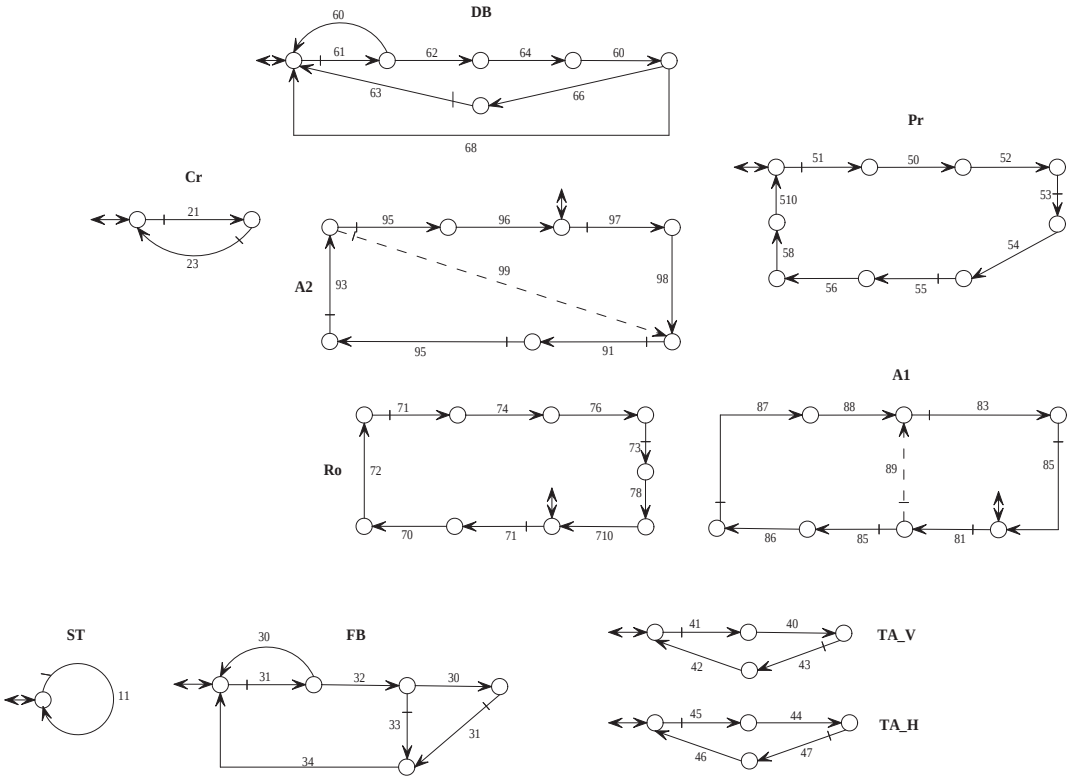


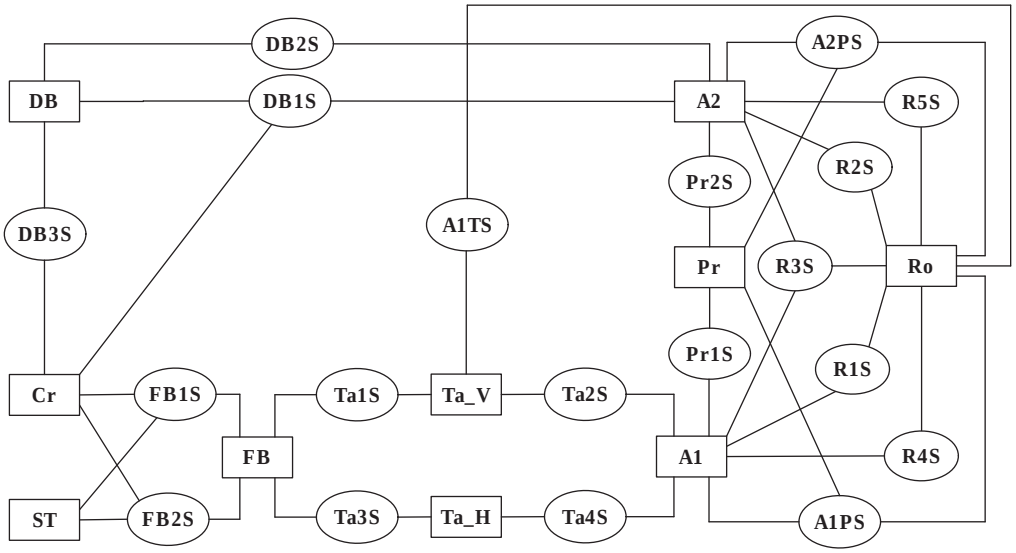
Figure 3.10: Model abstractions of plant components

blank_add	11	Pr_SMf	52	Ro_-90	74
Cr_mOn	21	Pr_UM	53	Ro_S-90	76
Cr_mOff	23	Pr_T	54	Ro_50	78
FB_Sf	30	Pr_D	55	Ro_S50	710
FB_F	31	Pr_MD	56	$A1\_mOn$	81
FB_s1On	32	Pr_B	58	$A1\_mOff$	83
FB_tau	33	Pr_SBf	510	$A1\_B$	85
FB_s1Off	34	DB_Sf	60	$A1\_37$	86
Ta_STf	40	DB_F	61	$A1\_F$	87
Ta_U	41	DB_s2On	62	$A1\_65$	88
Ta_SBf	42	DB_tau	63	$A1\_F'$	89
Ta_D	43	DB_s2Off	64	$A2\_mOn$	91
Ta_S50f	44	DB_yes	66	$A2\_mOff$	93
Ta_R	45	DB_no	68	$A2\_B$	95
Ta_S0f	46	Ro_35	70	$A2\_0$	96
Ta_L	47	Ro_L	71	$A2\_F$	97
Pr_MU	50	Ro_S35	72	$A2\_80$	98
Pr_UB	51	Ro_R	73	$A2\_F'$	99

Table 3.3: Original events vs. relabeled events

Step 2: Decentralized Supervisor Synthesis

First, for each specification we group its event-coupled components, and then compute a decentralized supervisor for each group. The interconnection structure is displayed as follows, with solid lines denoting event-coupling.



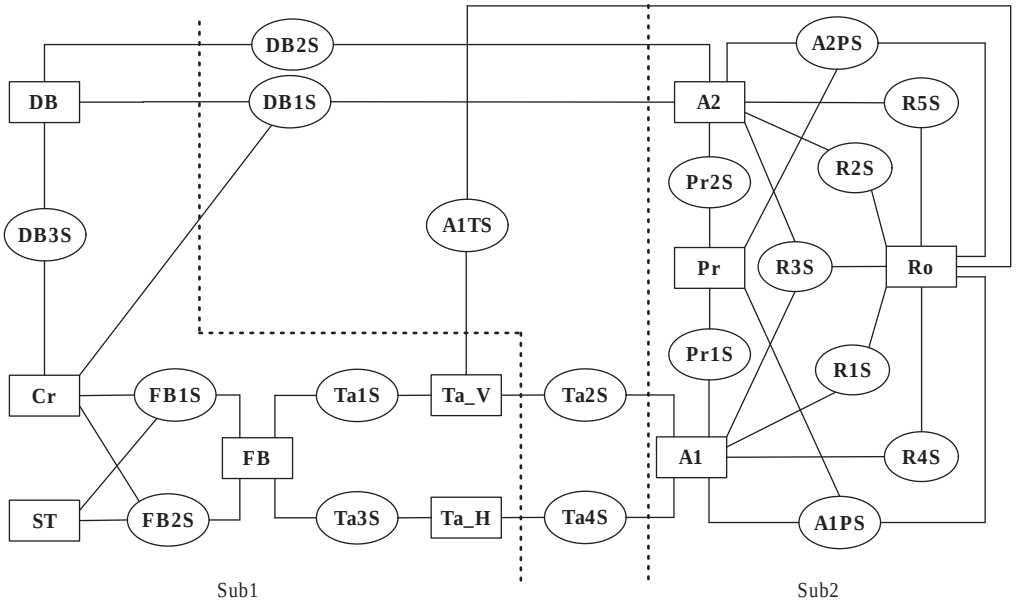
The state sizes of these supervisors are listed in the table below.

	State #	Reduced State #		State #	Reduced State #
FB1S	28	4	**DB3S**	14	2
FB2S	18	3	**R1S**	112	2
Ta1S	21	3	**R2S**	121	4
Ta2S	35	2	**R3S**	906	5
Ta3S	21	3	**R4S**	70	2
Ta4S	35	2	**R5S**	100	3
Pr1S	70	2	**A1PS**	495	6
Pr2S	70	2	**A2PS**	357	5
DB1S	252	3	**A1TS**	63	2
DB2S	70	2			

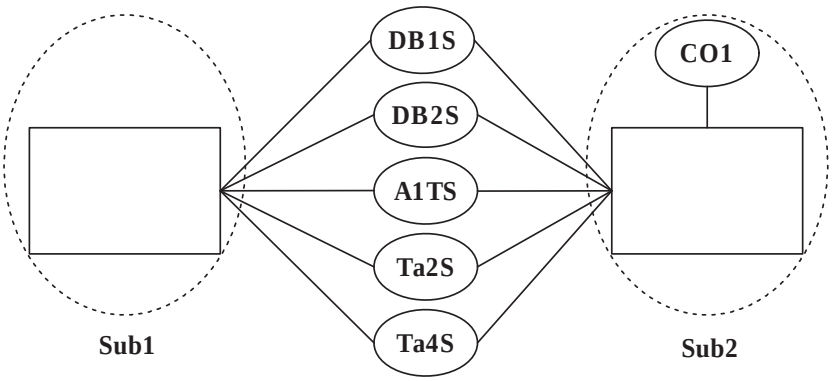
Step 3: Subsystem Decomposition and Coordination

We have nineteen decentralized supervisors, thus nineteen modules. For this structure, the control-flow nets approach fails to be applicable. Following [15], we decompose

the overall system into two subsystems, leaving five decentralized supervisors in between – **DB1S**, **DB2S**, **A1TS**, **Ta2S**, and **Ta4S**.



We now directly check the nonblocking property for each subsystem. While **Sub1** is nonblocking, **Sub2** turns out to be blocking. Thus a coordinator is designed to resolve the conflict in **Sub2**.



	State #	Reduced State #
CO1	650	3

Step 4: Subsystem Model Abstraction

We now need to verify the nonconflicting property among the two nonblocking subsystems, and the intermediate five decentralized supervisors. First, we determine their shared event set, denoted by Σ_{sub}. While **Sub1** and **Sub2** do not share events with each other, they do so with each of the five supervisors [2].

Reduced Supervisors	**SUB1**	**SUB2S**
DB1C	21, 61, 60, 64, 68, 63	93
DB2C	61, 64	93
Ta2C	43, 40	81
Ta4C	47, 44	81
A1TC	41, 40	70

So

$$\Sigma_{sub} = \{21, 40, 41, 43, 44, 47, 60, 61, 63, 64, 68, 70, 81, 93\}$$

To ensure the OCC property, we add events 45 and 71 to Σ_{sub}, since they are the immediately preceding controllable event of events 44 and 70, respectively. Denote the augmented alphabet

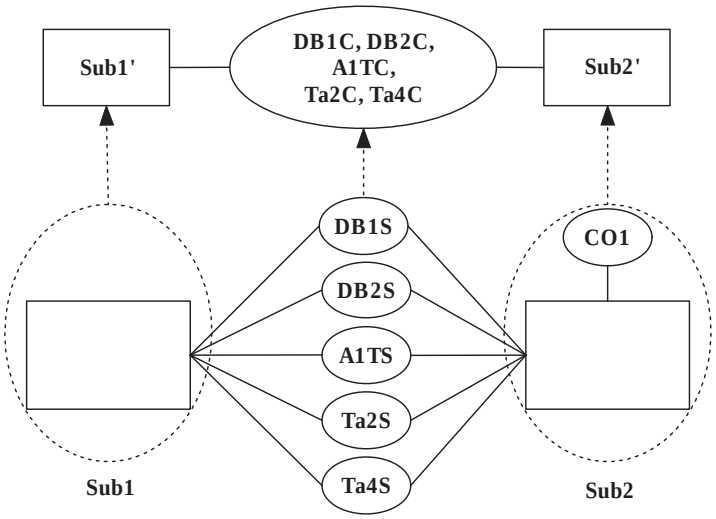
$$\Sigma'_{sub} = \{21, 40, 41, 43, 44, 45, 47, 60, 61, 63, 64, 68, 70, 71, 81, 93\}$$

Σ'_{sub} does not yet define an observer for either subsystem. Using the MX algorithm, we obtain a reasonable extension by adding events 11, 23, 62, 66, and 97 to Σ'_{sub}. We denote the extended alphabet again by

$$\Sigma'_{sub} = \{11, 21, 23, 40, 41, 43, 44, 45, 47, 60, 61, 62, 63, 64, 66, 68, 70, 71, 81, 93, 97\}$$

whose corresponding natural projection P'_{sub} is OCC and an observer for both subsystems. With P'_{sub}, we compute the subsystem model abstractions.

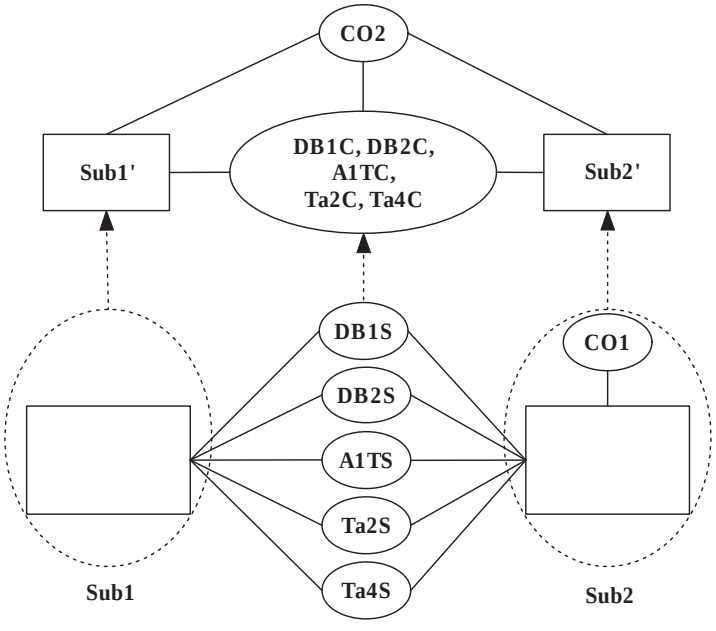
[2]Here we consider the reduced generators of these five supervisors



	Sub1	Sub1′	Sub2	Sub2′
State #	2478	644	650	13

Step 5: Abstracted Subsystem Decomposition and Coordination

We treat **Sub1′**, **Sub2′**, and the five reduced supervisors as a single group, and directly check the nonblocking property. This group turns out to be blocking; a coordinator then has to be designed to resolve the conflict.



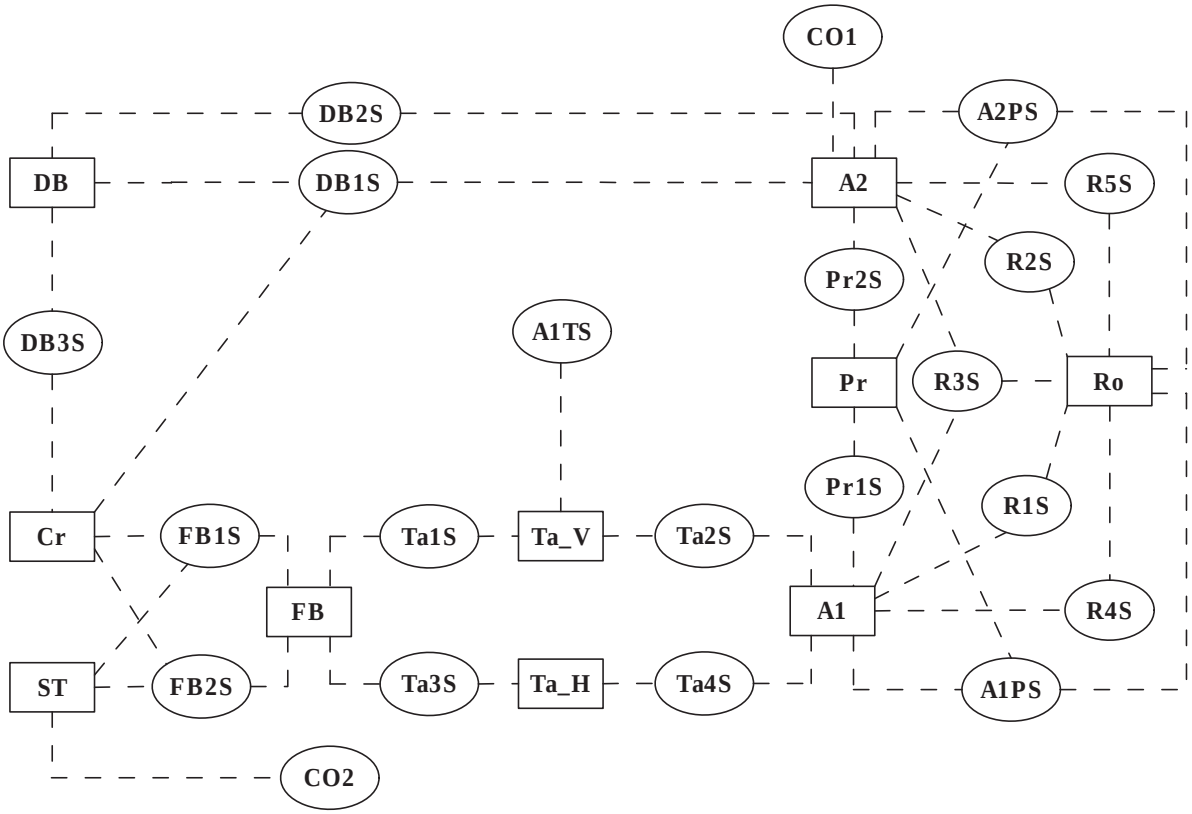
	State #	Reduced State #
CO2	6250	15

Step 6: Higher-Level Abstraction

The modular supervisory control design finishes at the last step.

Step 7: Localization

We start with determining the control-coupling relation through looking up the **condat** tables of each decentralized supervisor and coordinator. We show the result below, with dashed lines denoting the control-coupling.



First note that, although **A1TS** is event-coupled to both **TA** and **Ro**, it is control-coupled only to **TA**. Also notice that the two coordinators are control-coupled only to **A2** and **ST**, respectively. Along these dashed lines, we apply the supervisor localization algorithm. The state sizes of the resultant local controllers are listed in Table 2.4, and the

	ST #	FB #	TA #	PR #	DB #	CR #	RO #	A1 #	A2 #
FB1S	3	4				3			
FB2S	2	3				2			
TA1S		3	2						
TA2S			2					2	
TA3S		3	2						
TA4S			2					2	
Pr1S				2				2	
Pr2S				2					2
DB1S					3	3			3
DB2S					2				2
DB3S					2	2			
R1S							2	2	
R2S							3		4
R3S							4	4	4
R4S							2	2	
R5S							2		3
A1PS				5			6	5	
A2PS				5			3		5
A1TS			2						
CO1									3
CO2	15								

Table 3.4: State sizes of local controllers

generator models of each controller are displayed in Figs. 3.11–3.19 (for clarity irrelevant selfloops are omitted), grouped with respect to individual components. Thus we have established a purely distributed control architecture, wherein each of the component agents pursues its independent 'lifestyle', while being coordinated implicitly with its fellows through their local shared observable events.

Remark 3.3. The three mild modifications we made in step one enhance the comprehensibility of the resulting control logic. With the original setting, it was pointed out in [15] that even the reduced supervisors of **DB1S**, **A1PS**, and **A2PS** (of state sizes 7, 9, and 8, respectively) were too complicated to display. After modifying the models, however, the generators of the local controllers corresponding to the above three supervisors have state sizes ranging from 3 to 6, and hence they can be displayed readily. Further, with smaller state sizes, the control logic of these local controllers is more transparent than that of the corresponding decentralized supervisors in [15]. For example, the control logic of the local controller **DB1_A2** in Fig. 3.19 is simply that arm2 may unload a blank onto deposit belt (event 93) when there is no blank or only one on the belt.

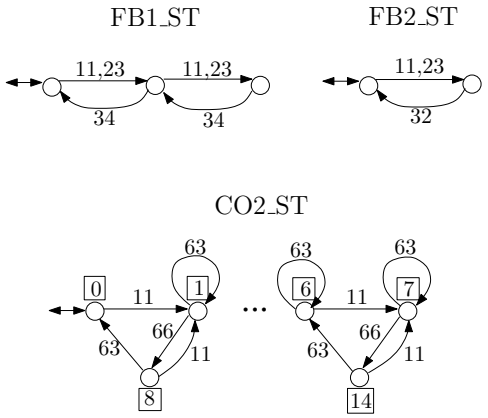


Figure 3.11: Local controllers for stock

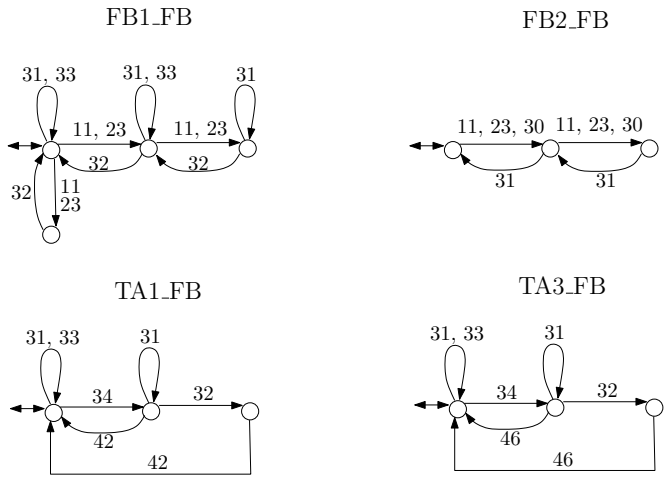


Figure 3.12: Local controllers for feed belt

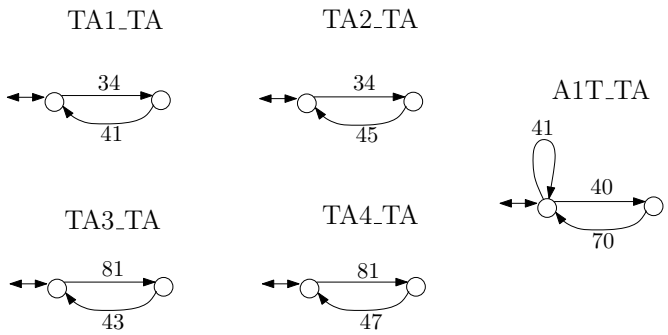


Figure 3.13: Local controllers for elevating rotary table

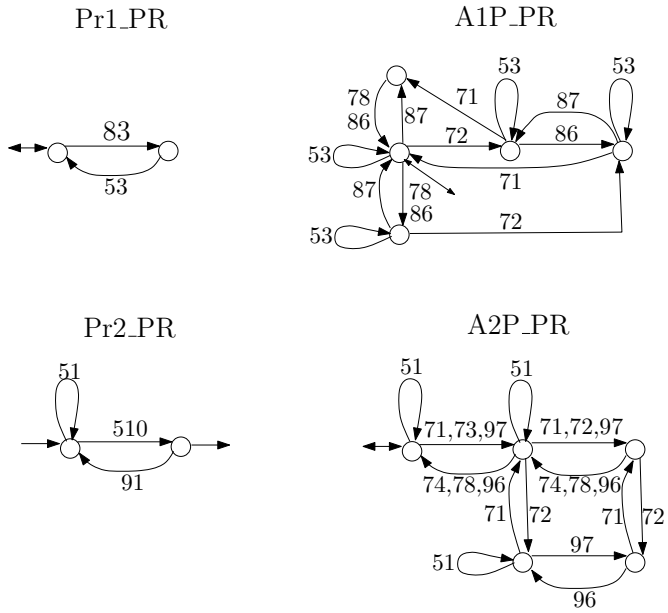


Figure 3.14: Local controllers for press

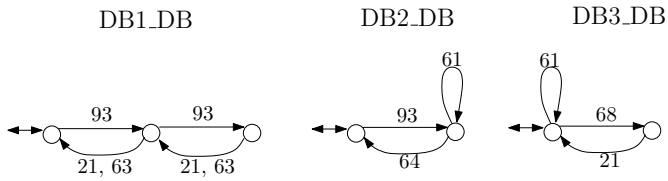


Figure 3.15: Local controllers for deposit belt

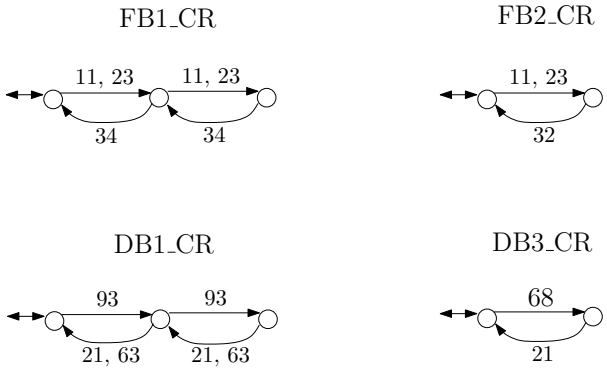


Figure 3.16: Local controllers for crane

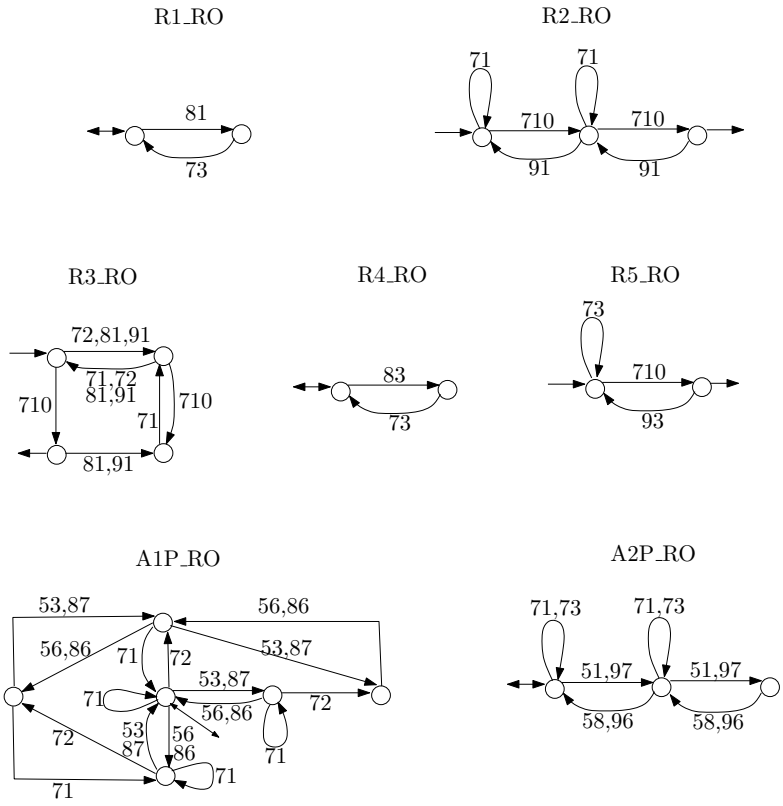


Figure 3.17: Local controllers for robot

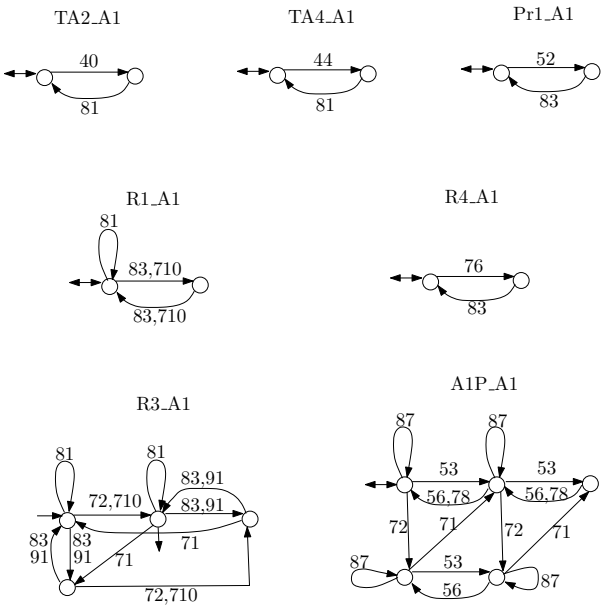


Figure 3.18: Local controllers for arm1

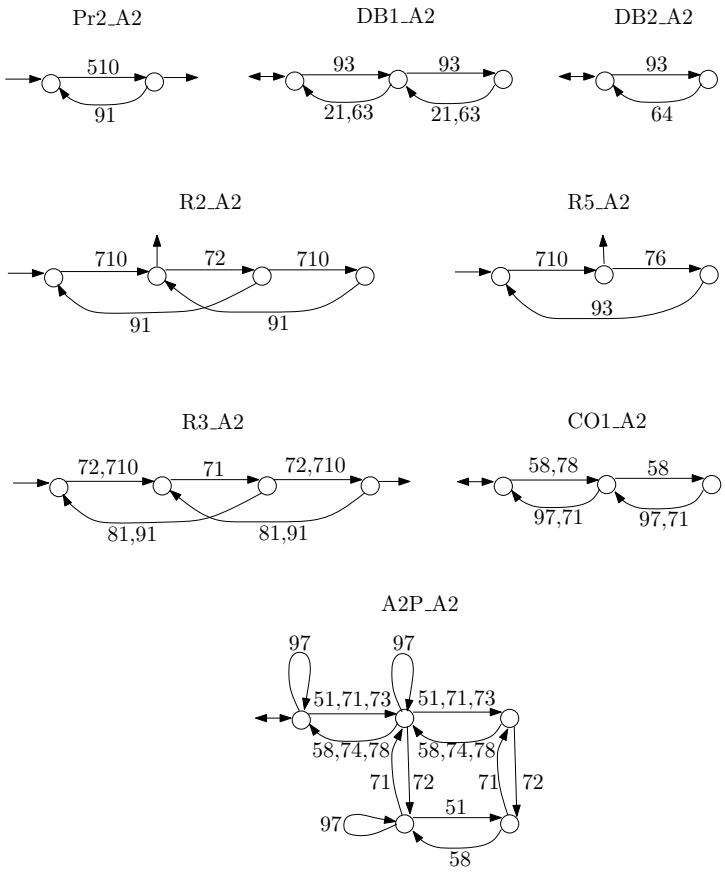


Figure 3.19: Local controllers for arm2

Chapter 4

State-Based Supervisor Localization

4.1 Introduction

So far we have studied supervisor localization in the language-based framework, and a decomposition-aggregation procedure is proposed therein to solve the distributed control problem of large-scale DES. In the present chapter we turn to a dual and more conventional viewpoint – the state-based framework – in which the counterpart supervisor localization concept and distributed control problem will be established; moreover, the framework of current concern opens up an alternative approach to tackle DES of large state size.

Specifically, we adopt the *state tree structure* (STS) ([57] [37]), a formalism that demonstrably is computationally efficient for monolithic supervisor synthesis. The efficiency is achieved first by modelling DES structurally using Statecharts [21], a graphical tool which offers economical representation of hierarchical and concurrent structure of the system state space. Thus a set of system states is organized into a hierarchy, or a *state tree*, equipped with *holon* modules describing system dynamics. In order to carry out symbolic computation, STS models are then encoded into predicates. The second feature underlying computational efficiency is exploitation of the *binary decision diagram*

(BDD) [9], a data structure which offers a compact representation of predicates. With BDD representation of encoded STS models, the computational complexity of supervisor synthesis becomes polynomial in the number of BDD nodes (|nodes|), rather than in the system state size (|states|). The encoding scheme is so designed that in many cases |nodes| $\ll$ |states|, thus achieving computational efficiency. As concrete evidence, it is claimed [37] that, based on the STS formalism, optimal nonblocking supervisory control design can be performed (in reasonable time and memory) for systems of state size 10^{24} and higher.

Stimulated by the hope of solving the distributed control problem for DES with the computational efficiency of STS, we develop the counterpart supervisor localization theory in the STS formalism. As an alternative to the decomposition-aggregation approach, the same top-down localization procedure as that in Chapter 2 can then be directly applied to deal with large, complex systems.

The setup of this chapter is the following. In Section 4.2 we provide a concise introduction to the STS formalism. In Sections 4.3 and 4.4, we establish the counterpart distributed control problem and supervisor localization theory, respectively. In Section 4.5 we present a symbolic localization algorithm, a counterpart to that in Section 2.4, and finally in Section 4.6 we illustrate the localization theory and algorithm with the familiar Transfer Line example.

4.2 Preliminaries

4.2.1 STS Modelling

STS models the state space of a DES as a state hierarchy, established by bringing in 'artificial' *superstates*. Let X be a finite state set. For $x \in X$, x is an *OR* (respectively, *AND*) *superstate* if there exists a nonempty subset $Y \subseteq X$ such that $x \notin Y$ and x can be represented by the *union* (respectively, *cartesian product*) of the states in Y. We call

each state in Y an *OR* (respectively, *AND*) *component* of x, and the states in X other than superstates *simple states*.

We introduce two useful functions associated with the hierarchical state space X. Define the *type* function $\mathcal{T} : X \to \{or, and, simple\}$ according to

$$\mathcal{T}(x) := \begin{cases} or, & \text{if } x \text{ is an OR superstate} \\ and, & \text{if } x \text{ is an AND superstate} \\ simple, & \text{if } x \text{ is a simple state} \end{cases}$$

Define the *expansion* function $\mathcal{E} : X \to Pwr(X)$ according to

$$\mathcal{E}(x) := \begin{cases} Y, & \text{if } \mathcal{T}(x) \in \{or, and\} \\ \emptyset, & \text{if } \mathcal{T}(x) = simple \end{cases}$$

Extend $\mathcal{E}$ to $\hat{\mathcal{E}}^1 : X \to Pwr(X)$ such that

$$\hat{\mathcal{E}}^1(x) := \mathcal{E}(x) \cup \{x\}$$

and consider the sequence of functions $\hat{\mathcal{E}}^n : X \to Pwr(X)$ given by

$$\hat{\mathcal{E}}^n(x) := \bigcup \{\mathcal{E}(y) \cup \{y\} | y \in \hat{\mathcal{E}}^{n-1}(x)\}, \quad n > 1$$

By construction we have $\hat{\mathcal{E}}^n(x) \subseteq \hat{\mathcal{E}}^{n+1}(x)$ for all $x \in X$ (i.e., the sequence is monotone, in the sense of subset inclusion). Since X is finite, the limit of this sequence, $\lim_{n\to\infty} \hat{\mathcal{E}}^n(x)$, must exist; we denote this limit by $\mathcal{E}^*$. In addition, we write $\mathcal{E}^+(x) := \mathcal{E}^*(x) - \{x\}$, and call each state in $\mathcal{E}^+(x)$ a *descendant* of x and x an *ancestor* of the states in $\mathcal{E}^+(x)$.

Definition 4.1. ([37, Definition 2.2])

Consider the 4-tuple $\mathbf{S}T = (X, x_0, \mathcal{T}, \mathcal{E})$, where X is a finite state set with $X = \mathcal{E}^*(x_0)$; $x_0 \in X$ is the *root state*; $\mathcal{T} : X \to \{or, and, simple\}$ is the type function; and

$\mathcal{E} : X \to Pwr(X)$ is the expansion function. $\mathbf{S}T$ is a *state tree* if

(1) (terminal case) $X = \{x_0\}$, or,

(2) (recursive case) $(\forall y \in \mathcal{E}(x_0))\ \mathbf{S}T^y = (\mathcal{E}^*(y), y, \mathcal{T}|_{\mathcal{E}^*(y)}, \mathcal{E}|_{\mathcal{E}^*(y)})$ is also a state tree such that

- $(\forall y, y' \in \mathcal{E}(x_0))\ y \neq y' \Rightarrow \mathcal{E}^*(y) \cap \mathcal{E}^*(y') = \emptyset$
- $\dot{\bigcup}\{\mathcal{E}^*(y) | y \in \mathcal{E}(x_0)\} = \mathcal{E}^+(x_0)$ ◊

Remark 4.1.

1. In the recursive case, $\mathbf{S}T^y$ is called a *child state tree* of x_0 rooted at y. Also notice that the set $\{\mathcal{E}^*(y) | y \in \mathcal{E}(x_0)\}$ partitions $\mathcal{E}^+(x_0)$.

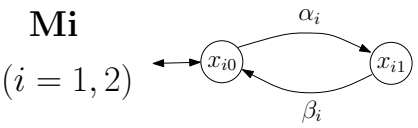
2. A state tree $\mathbf{S}T = (X, x_0, \mathcal{T}, \mathcal{E})$ is *well-formed* if

$$(\forall x, y \in X)\ \mathcal{T}(x) = and\ \&\ y \in \mathcal{E}(x) \Rightarrow \mathcal{T}(y) \neq simple$$

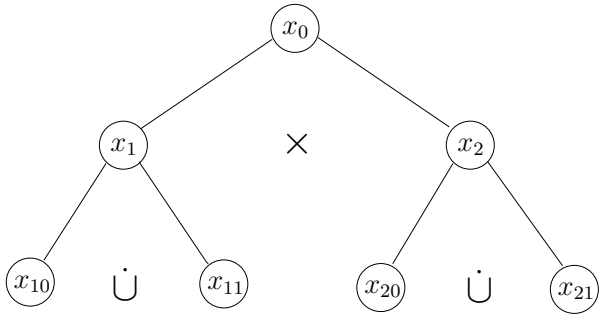
That is, no AND component can be a simple state.

Example 4.1.

Consider the Small Factory [63, Example 3.3.4] consisting of two machines **M1**, **M2**.



The entire state space of this system can be modelled as the state tree displayed in Fig. 4.1. Two OR superstates x_i $(i = 1, 2)$ are brought in as an index for the set of simple states $\{x_{i0}, x_{i1}\}$, and an AND superstate (also the root state), x_0, is introduced to model the synchronous product of **M1** and **M2**. This state tree is valid because the two child state trees of x_0 (rooted at x_1 and x_2, respectively) are state trees on their own,

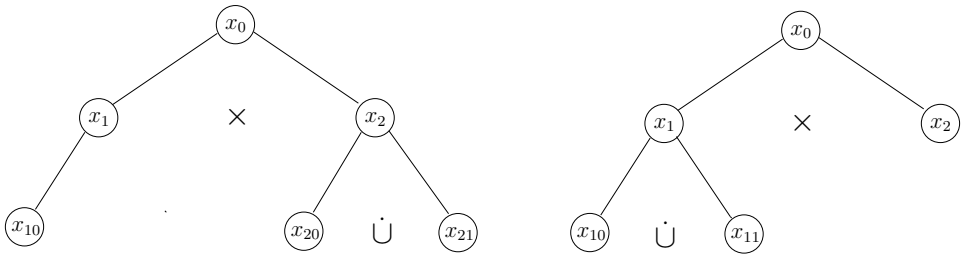


$\mathbf{S}T$

Figure 4.1: State tree model for small factory

and the set $\{\mathcal{E}^*(x_1), \mathcal{E}^*(x_2)\}$ partitions $\mathcal{E}^+(x_0)$. Besides, this state tree is well-formed, since the AND components of x_0, namely x_1 and x_2, are OR superstates. ♦

Let $\mathbf{S}T = (X, x_0, \mathcal{T}, \mathcal{E})$ be a well-formed state tree. A *sub-state-tree* of $\mathbf{S}T$ is also a well-formed state tree with x_0 as the root state, but contains only a nonempty subset of OR components, for every OR superstate in $\mathbf{S}T$. For example, in Fig. 4.2, $\mathbf{S}T_1$ is a sub-state-tree of $\mathbf{S}T$ in Example 4.1, while $\mathbf{S}T_2$ is not because it contains no OR components of x_2. We write $\mathcal{ST}(\mathbf{S}T)$ as the set of all sub-state-trees of $\mathbf{S}T$.



$\mathbf{S}T_1$ $\mathbf{S}T_2$

Figure 4.2: Example: sub-state-tree of $\mathbf{S}T$

In particular, if a sub-state-tree of $\mathbf{S}T$ contains exactly a singleton set of OR components for every OR superstate, we call it a *basic state tree* of $\mathbf{S}T$. For example, in Fig. 4.3, b_1 and b_2 are both basic state trees of $\mathbf{S}T$ in Example 4.1. We identify basic state trees in $\mathcal{ST}(\mathbf{S}T)$ because they correspond in turn to the generator states of the whole

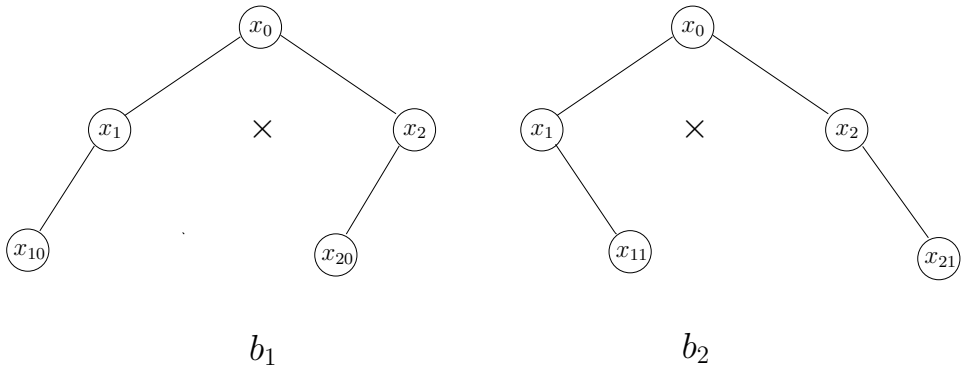


Figure 4.3: Example: basic state tree of $\mathbf{S}T$

system. Denote by $\mathcal{B}(\mathbf{S}T)$ the set of all basic state trees of $\mathbf{S}T$.

Having organized the state space into a state tree, we are left to establish the associated system dynamics – the transition structure of the state tree. We start with *holon*, a local transition structure that describes the inner and boundary dynamics of OR components.

Definition 4.2. ([37, Definition 2.13])

A *holon* H is a 5-tuple $H = (X, \Sigma, \delta, X_0, X_m)$, where

(1) X, the finite state set, is the disjoint union of the *external state set* X_E and the *internal state set* X_I.

(2) Σ, the event set, is the disjoint union of the *boundary event set* Σ_B and the *internal event set* Σ_I.

(3) $\delta : X \times \Sigma \to X$ (pfn), the local transition function, is the disjoint union [1] of the *internal transition structure* $\delta_I : X_I \times \Sigma_I \to X_I$ and the *boundary transition structure* δ_B; the latter is again the disjoint union of the *incoming* boundary transition structure $\delta_{BI} : X_E \times \Sigma_B \to X_I$ and the *outgoing* boundary transition structure $\delta_{BO} : X_I \times \Sigma_B \to X_E$.

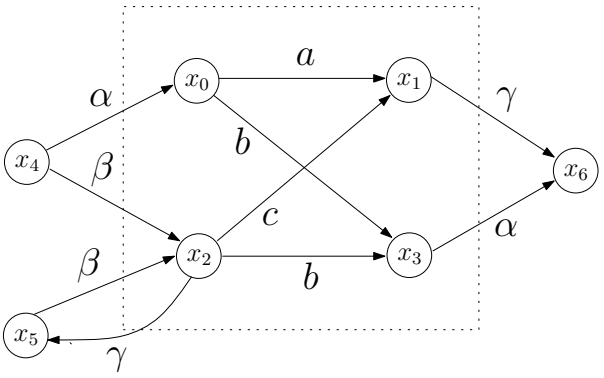
[1] Two transition functions $\delta_i : X \times \Sigma \to X$ $(i = 1, 2)$ are disjoint if the two sets $\{(x, \sigma, \delta_1(x, \sigma)) | \delta_1(x, \sigma)!\}$ and $\{(x, \sigma, \delta_2(x, \sigma)) | \delta_2(x, \sigma)!\}$ are disjoint, i.e., δ_1 and δ_2 have no transition in common.

(4) $X_0 \subseteq X_I$, the *initial state set*, contains those target states of incoming boundary transitions if δ_{BI} is defined. Otherwise, X_0 can be selected to be any nonempty subset of X_I.

(5) $X_m \subseteq X_I$, the *marked state set*, contains those source states of outgoing boundary transitions if δ_{BO} is defined. Otherwise, X_m can be selected to be any nonempty subset of X_I. ◊

Example 4.2.

A holon H



A typical holon, $H = (X, \Sigma, \delta, X_0, X_m)$, is displayed above. We determine its components.

(1) The state set $X = X_I \dot{\cup} X_E$, where $X_I = \{x_0, x_1, x_2, x_3\}$ and $X_E = \{x_4, x_5, x_6\}$.

(2) The event set $\Sigma = \Sigma_I \dot{\cup} \Sigma_E$ where $\Sigma_I = \{a, b, c\}$ and $\Sigma_B = \{\alpha, \beta, \gamma\}$.

(3) The internal transitions are $\delta_I(x_0, a) = x_1$, $\delta_I(x_0, b) = x_3$, $\delta_I(x_2, b) = x_3$, and $\delta_I(x_2, c) = x_1$; the incoming boundary transitions are $\delta_{BI}(x_4, \alpha) = x_0$, $\delta_{BI}(x_4, \beta) = x_2$, and $\delta_{BI}(x_5, \alpha) = x_0$; finally, the outgoing boundary transitions are $\delta_{BO}(x_2, \gamma) = x_5$, $\delta_{BO}(x_1, \gamma) = x_6$, and $\delta_{BO}(x_3, \alpha) = x_6$.

(4) The initial state set $X_0 = \{x_0, x_2\}$.

(5) The marked state set $X_m = \{x_0, x_1, x_2\}$. ♦

Now we match holons to their corresponding OR superstates in a state tree. This operation should respect two constraints – *boundary consistency* and *local coupling* [37, Chapter 2]. Informally, boundary consistency requires compatible inner and boundary behaviors between any adjacent pair of holons in the vertical direction; local coupling requires that internal events be shared by only those holons in the horizontal direction that have a common adjacent AND ancestor.

Also, we extend the local internal transition function δ_I to $\bar{\delta}_I$ [37, Definition 2.17] to handle cases where transitions involve superstates as the source or target states. This extension allows the construction of global transition structures. Define the global transition function $\Delta : \mathcal{ST}(\mathbf{S}T) \times \Sigma \to \mathcal{ST}(\mathbf{S}T)$ such that for all $T \in \mathcal{ST}(\mathbf{S}T)$ and $\sigma \in \Sigma$,

$$\Delta(T, \sigma) := \text{replace\_source}_{\mathbf{G},\sigma}(T \wedge \text{Elig}_{\mathbf{G}}(\sigma))$$

where $\text{Elig}_{\mathbf{G}}(\sigma) \in \mathcal{ST}(\mathbf{S}T)$ is the largest sub-state-tree of $\mathbf{S}T$ where σ can occur; letting $T_E := T \wedge \text{Elig}_{\mathbf{G}}(\sigma)$, $\text{replace\_source}_{\mathbf{G},\sigma}(T_E)$ replaces all of the argument's (child) sub-state-trees T_E^x (rooted at x) by $\bar{\delta}_I^x(T_E^x, \sigma)$, whenever $\bar{\delta}_I^x(T_E^x, \sigma)!$.

Finally, we can state

Definition 4.3. (State Tree Structure [37, Definition 2.16])

Consider the 6-tuple $\mathbf{G} = (\mathbf{S}T, \mathcal{H}, \Sigma, \Delta, \mathbf{S}T_0, \mathbf{S}T_m)$, where $\mathbf{S}T = (X, x_0, \mathcal{T}, \mathcal{E})$ is a state tree; $\mathcal{H} = \{H^a | \mathcal{T}(a) = or \ \& \ H^a = (X^a, \Sigma^a, \delta^a, X_0^a, X_m^a)\}$ is the set of holons that are matched to the OR superstates in $\mathbf{S}T$; $\Sigma = \bigcup\{\Sigma_I^a | H^a \in \mathcal{H}\}$ is the set of internal events of $\mathcal{H}$; $\Delta : \mathcal{ST}(\mathbf{S}T) \times \Sigma \to \mathcal{ST}(\mathbf{S}T)$ is the global transition function; $\mathbf{S}T_0 \in \mathcal{ST}(\mathbf{S}T)$ is the initial state tree; and $\mathbf{S}T_m \subseteq \mathcal{ST}(\mathbf{S}T)$ is the set of marker state trees. $\mathbf{G}$ is a *state tree structure* (STS) if both boundary consistency and local coupling hold when matching $\mathcal{H}$ with $\mathbf{S}T$. ◊

Remark 4.2. For STS synthesis, a *backward global transition function* is needed. Following a dual route, we define $\Gamma : \mathcal{S}T(\mathbf{S}T) \times \Sigma \to \mathcal{S}T(\mathbf{S}T)$ such that for all $T \in \mathcal{S}T(\mathbf{S}T)$ and $\sigma \in \Sigma$,

$$\Gamma(T, \sigma) := \text{replace\_target}_{\mathbf{G},\sigma}(T \wedge \text{Next}_{\mathbf{G}}(\sigma))$$

where $\text{Next}_{\mathbf{G}}(\sigma) \in \mathcal{S}T(\mathbf{S}T)$ is the largest sub-state-tree of $\mathbf{S}T$ that σ targets; letting $T_N := T \wedge \text{Next}_{\mathbf{G}}(\sigma)$, $\text{replace\_target}_{\mathbf{G},\sigma}(T_N)$ replaces all of the argument's (child) sub-state-trees T_N^x (rooted at x) by T_N^y, where $T_N^x = \bar{\delta}_I^x(T_N^y, \sigma)$.

4.2.2 Symbolic Representation of STS

Having discussed the STS modelling for a DES, we are now ready to represent the model symbolically. This step is fundamental because it is the basis for symbolic computation on STS. Our particular focus is symbolic computation for supervisory control synthesis.

We begin with encoding state trees. Let $\mathbf{S}T = (X, x_0, \mathcal{T}, \mathcal{E})$ be a state tree. A *predicate* P defined on $\mathcal{B}(\mathbf{S}T)$ is a function $P : \mathcal{B}(\mathbf{S}T) \to \{0, 1\}$; thus P is the *characteristic function* of the set $B_P := \{b \in \mathcal{B}(\mathbf{S}T) | P(b) = 1\}$ [2]. That is, for $b \in \mathcal{B}(\mathbf{S}T)$, $b \models P$ iff $b \in B_P$. Similarly, for $T \in \mathcal{S}T(\mathbf{S}T)$, $T \models P$ iff $\mathcal{B}(T) \subseteq B_P$, and the extension $P : \mathcal{S}T(\mathbf{S}T) \to \{0, 1\}$ follows accordingly. We write $Pred(\mathbf{S}T)$ for the set of all predicates defined on $\mathcal{S}T(\mathbf{S}T)$, and define the propositional connectives $\wedge, \vee$, and $\neg$ in the usual way. We also introduce a partial order on $Pred(\mathbf{S}T)$: for $P_1, P_2 \in Pred(\mathbf{S}T)$ define $P_1 \preceq P_2$ (say P_1 is a *subpredicate* of P_2) iff $P_1 \Rightarrow P_2$. With this partial order, $(Pred(\mathbf{S}T), \preceq)$ is a complete lattice [37, Section 3.1].

The following function specifies the mechanism that assigns every sub-state-tree T of $\mathbf{S}T$ to the predicate P it satisfies, i.e., $T \models P$.

Definition 4.4. ([37, Definition 4.1])

Let $\mathbf{S}T = (X, x_0, \mathcal{T}, \mathcal{E})$ be a state tree and $T = (Y, x_0, \mathcal{T}|_Y, \mathcal{E}|_Y) \in \mathcal{S}T(\mathbf{S}T)$. Associate

[2] The satisfaction relation $P(b) = 1$ will often be written $b \models P$.

to each OR superstate x a *state variable* v_x ranging over $\mathcal{E}(x)$. Define $\Theta : \mathcal{S}T(\mathbf{S}T) \to Pred(\mathbf{S}T)$ recursively by

$$\Theta(T) := \begin{cases} \bigvee\{(v_{x_0} = y) \wedge \Theta(T^y) \mid y \in \mathcal{E}|_Y(x_0)\}, & \text{if } \mathcal{T}(x_0) = or \\ \bigwedge\{\Theta(T^y) \mid y \in \mathcal{E}|_Y(x_0)\}, & \text{if } \mathcal{T}(x_0) = and \\ 1, & \text{if } \mathcal{T}(x_0) = simple \end{cases}$$

where "=" in $(v_{x_0} = y)$ is the assignment operator, and $(v_{x_0} = y)$ returns value 1 iff v_{x_0} is assigned value y. ◊

Remark 4.3.

1. If all of the components of an OR superstate x are on the sub-state-tree T, then the predicate $\Theta(T)$ is independent of the state variable v_x; namely, the following is a tautology:

$$\left(\bigvee\{v_x = y | y \in \mathcal{E}(x)\}\right) \equiv 1 \tag{4.1}$$

   where "≡" denotes logical equivalence. For example, in Fig. 4.2,

$$\begin{aligned} \Theta(\mathbf{S}T_1) &:= (v_{x_1} = x_{10}) \wedge (v_{x_2} = x_{20} \ \vee \ v_{x_2} = x_{21}) \\ &\equiv (v_{x_1} = x_{10}) \end{aligned}$$

   where v_{x_1}, v_{x_2} are the state variables for the OR superstates x_1 and x_2, respectively. Notice that the predicate $\Theta(\mathbf{S}T_1)$ is simplified by applying the tautology (4.1) to v_{x_2}.

2. [37, Definition 4.2] Let v_x be a state variable appearing in a predicate P, and for $y \in \mathcal{E}(x)$ denote by $P[y/v_x]$ the resulting predicate after assigning y to v_x. We define $\exists v_x P := \bigvee\{P[y/v_x] \mid y \in \mathcal{E}(x)\}$, which adds all of the components of the OR superstate x back onto the sub-state-tree satisfying P. Hence, it again follows from the tautology (4.1) that the variable v_x will not appear in $\exists v_x P$. Continuing

the above example,

$$\exists v_{x_1}\Theta(\mathbf{S}T_1) := v_{x_1} = x_{10} \ \vee \ v_{x_1} = x_{11}$$
$$\equiv 1$$

Next given an STS $\mathbf{G}$ defined over Σ, we encode its backward global transition function Γ. First we bring in some notation. Associate to every OR superstate x in $\mathbf{G}$ a *normal* state variable v_x (respectively, a *prime* state variable v'_x) if x is a target (respectively, source) state in a transition. Then for a predicate P, we write $P(\mathbf{v})$ to mean that P is defined over $\mathbf{v}$, a set of normal state variables. Denote by $P(\mathbf{v})[\mathbf{v} \rightarrow \mathbf{v}']$ the replacement of $\mathbf{v}$ by $\mathbf{v}'$ in $P(\mathbf{v})$; the resulting predicate is defined over $\mathbf{v}'$, i.e., $P(\mathbf{v}')$.

For $\sigma \in \Sigma$ let the triple $(\mathbf{S}, \sigma, \mathbf{T})$ represent the entire set of transitions in $\mathbf{G}$ labeled with σ, where $\mathbf{S}$ and $\mathbf{T}$ are the predicates of the source sub-state-trees and the target sub-state-trees, repectively. Denote by $\mathbf{v}_{\sigma,\mathbf{S}}$ and $\mathbf{v}_{\sigma,\mathbf{T}}$ the set of variables over which $\mathbf{S}$ and $\mathbf{T}$ are defined. Then we can derive a predicate N_σ which characterizes the transition set $(\mathbf{S}, \sigma, \mathbf{T})$ [3]; this predicate N_σ is defined over $\mathbf{v}'_{\sigma,\mathbf{S}}$ and $\mathbf{v}_{\sigma,\mathbf{T}}$.

Definition 4.5. ([37, Definition 4.3])

Let $\sigma \in \Sigma$ be an event and $P \in Pred(\mathbf{S}T)$ be a predicate. Define $\hat{\Gamma} : Pred(\mathbf{S}T) \times \Sigma \rightarrow Pred(\mathbf{S}T)$ according to

$$\hat{\Gamma}(P, \sigma) := (\exists \mathbf{v}_{\sigma,\mathbf{T}}(P \wedge N_\sigma))[\mathbf{v}'_{\sigma,\mathbf{S}} \rightarrow \mathbf{v}_{\sigma,\mathbf{S}}]$$

$\Diamond$

Informally, $\hat{\Gamma}(P, \sigma)$ returns a predicate characterizing the largest (source) set of basic state trees, each of which can reach a basic state tree in B_P by a one-step transition σ.

To compute $\hat{\Gamma}(P, \sigma)$, we first compute $P \wedge N_\sigma$ that holds for those transitions in

[3] For the detailed derivation of N_σ, see [37, Section 4.2.2]

$(\mathbf{S}, \sigma, \mathbf{T})$ whose target sub-state-trees satisfy P. With $P \wedge N_\sigma$, we quantify out all variables in $\mathbf{v}_{\sigma,\mathbf{T}}$ by the $\exists$ operator, thus obtaining the source sub-state trees; the resulting predicate $\exists \mathbf{v}_{\sigma,\mathbf{T}}(P \wedge N_\sigma)$ is defined only on $\mathbf{v}'_{\sigma,\mathbf{S}}$. Lastly we replace $\mathbf{v}'_{\sigma,\mathbf{S}}$ by $\mathbf{v}_{\sigma,\mathbf{S}}$ in order to let the final predicate be defined over normal variables.

4.2.3 Optimal Nonblocking Supervisory Control of STS

Given an STS $\mathbf{G} = (\mathbf{S}T, \mathcal{H}, \Sigma, \Delta, P_0, P_m)$ [4] and a predicate $P \in Pred(\mathbf{S}T)$ with B_P denoting the set of *illegal* basic state trees, our objective is to synthesize the *largest* subpredicate of $\neg P$ which is *(weakly) controllable* and *nonblocking* (as defined below).

We define a *state feedback control* (SFBC) [63, Chapter 7] for $\mathbf{G}$ to be the total function

$$f : \mathcal{B}(\mathbf{S}T) \to \Pi$$

where $\Pi := \{\Sigma' \subseteq \Sigma | \Sigma_u \subseteq \Sigma'\}$. Thus f 'attaches' to each basic state tree of $\mathbf{G}$ a subset of events that always contains the uncontrollable events. The event σ is *enabled* at $b \in \mathcal{B}(\mathbf{S}T)$ if $\sigma \in f(b)$, and is *disabled* otherwise. For $\sigma \in \Sigma$ we define a *control function* $f_\sigma : \mathcal{B}(\mathbf{S}T) \to \{0, 1\}$ according to $f_\sigma(b) = 1$ iff $\sigma \in f(b)$. Thus the control actions of f can be fully distributed to the set $\{f_\sigma | \sigma \in \Sigma\}$. The closed-loop global transition function induced by f is given by

$$\Delta^f(b, \sigma) := \begin{cases} \Delta(b, \sigma), & \text{if } f_\sigma(b) = 1 \\ \emptyset, & \text{if } f_\sigma(b) = 0 \end{cases}$$

We write $\mathbf{G}^f = (\mathbf{S}T, \mathcal{H}, \Sigma, \Delta^f, P_0^f, P_m)$ for the closed-loop STS formed from $\mathbf{G}$ and f, with Δ^f as above and $P_0^f \preceq P_0$.

Let $Q \in Pred(\mathbf{S}T)$ be a predicate. The *reachability* predicate $R(\mathbf{G}, Q)$ is defined to hold precisely on those basic state trees that can be reached in $\mathbf{G}$ from B_{P_0} via basic state

[4] Here $P_0 := \Theta(\mathbf{S}T_0)$ and $P_m := \bigvee\{\Theta(\mathbf{S}T_i) | \mathbf{S}T_i \in \mathbf{S}T_m\}$

trees satisfying Q. For $\sigma \in \Sigma$ the *weakest liberal precondition* is the predicate transformer $M_\sigma : Pred(\mathbf{S}T) \to Pred(\mathbf{S}T)$ defined by

$$b \models M_\sigma(Q) \text{ iff } \Delta(b, \sigma) \models Q, \quad b \in \mathcal{B}(\mathbf{S}T)$$

We say a predicate $Q \in Pred(\mathbf{S}T)$ is *weakly controllable* (with respect to $\mathbf{G}$) if

$$(\forall \sigma \in \Sigma_u)\ Q \preceq M_\sigma(Q)$$

It can then be shown that, if $Q \wedge P_0 \neq false$ and Q is weakly controllable, there exists a SFBC f such that $R(\mathbf{G}, Q) = R(\mathbf{G}^f, true)$ [37, Theorem 3.1].

Now suppose Q is not weakly controllable. Denote by $\mathcal{CP}(Q)$ the set of all weakly controllable subpredicates of Q. Then [37, Proposition 3.2] $\mathcal{CP}(Q)$ contains a (unique) supremal element, denoted by $sup\mathcal{CP}(Q)$.

It is left to ensure the nonblocking property. To this end, we introduce the *coreachability predicate* $CR(\mathbf{G}, P)$ defined recursively as follows:

1. $(b_m \models P_m \wedge P) \Rightarrow (b_m \models CR(\mathbf{G}, P))$
2. $(b \models CR(\mathbf{G}, P)\ \&\ \sigma \in \Sigma\ \&\ \Delta(b', \sigma) = b\ \&\ b' \models P) \Rightarrow (b' \models CR(\mathbf{G}, P))$

We say a predicate $Q \in Pred(\mathbf{S}T)$ is *coreachable* (with respect to $\mathbf{G}$) if

$$Q \preceq CR(\mathbf{G}, Q)$$

Also, we say a SFBC f for $\mathbf{G}$ is *nonblocking* if

$$R(\mathbf{G}^f, true) \preceq CR(\mathbf{G}^f, true)$$

Then we have the result that, if $Q \wedge P_0 \neq false$ and Q is weakly controllable and coreachable, then there exists a nonblocking SFBC f such that $R(\mathbf{G}, Q) = R(\mathbf{G}^f, true)$

[37, Theorem 3.2].

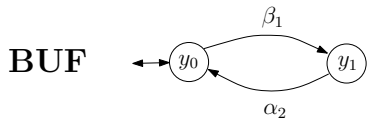
Again suppose Q is either not weakly controllable or not coreachable. Denote by $\mathcal{C}^2\mathcal{P}(Q)$ the set of all weakly controllable and coreachable subpredicates of Q. Then [37, Proposition 3.5] $\mathcal{C}^2\mathcal{P}(Q)$ contains a (unique) supremal element, denoted by $sup\mathcal{C}^2\mathcal{P}(Q)$.

To conclude, on returning to the original given STS $\mathbf{G}$ and predicate P, we solve the corresponding supervisory control problem by synthesizing the supremal weakly controllable and coreachable subpredicate of $\neg P$, denoted by $sup\mathcal{C}^2\mathcal{P}(\neg P)$; this we know can be implemented by a nonblocking SFBC f.

Remark 4.4.

In the language-based framework, a control problem is typically given in terms of a plant generator $\mathbf{P}$ and a specification generator $\mathbf{S}$ that imposes a behavioral constraint on $\mathbf{P}$. We show how to convert this pair $(\mathbf{P}, \mathbf{S})$ into an STS model $\mathbf{G}$ with a predicate P specifying the illegal basic state trees.

First, to construct $\mathbf{G}$ we bring in an AND (root) superstate and 'link' both $\mathbf{P}$ and $\mathbf{S}$ to it. To illustrate, continuing Example 4.1 we let the following one-slot buffer be the specification. Then the STS model is obtained as shown in Fig. 4.4. So it is the entire



control problem that the STS $\mathbf{G}$ models, instead of merely the uncontrolled plant.

Next we need to determine the predicate P specifying those illegal basic state trees that $\mathbf{G}$ is prohibited from visiting. Notice that the control requirement imposed by the specification generator $\mathbf{S}$ is expressed implicitly through its partial transition function. It is this implicit requirement that helps identify the illegal basic state trees. For example, the generator **BUF** above conveys two elementary requirements: disabling event α_2 at state y_0 and disabling event β_1 at state y_1. While the former disablement does not render

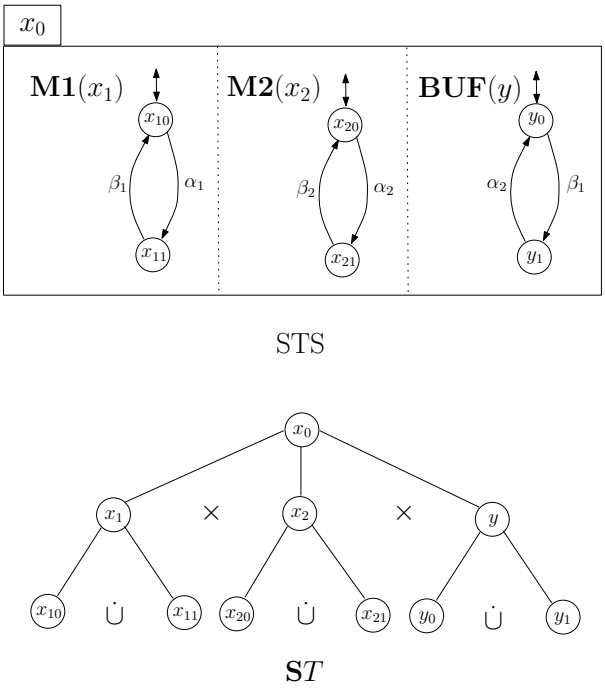


Figure 4.4: STS model for small factory

any basic state tree illegal because α_2 is controllable [5], the latter requirement does make the basic state tree in Fig. 4.5 illegal since β_1 is uncontrollable. Thus $P := v_{x_1} = x_{11} \wedge v_y = y_1$, where v_y is the state variable of the superstate y.

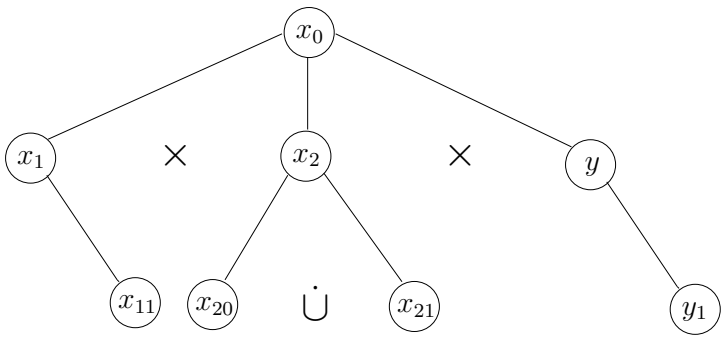


Figure 4.5: Illegal basic state tree

Finally, it is important to note that the control requirement – disabling event α_2 at state y_0 – is in fact 'embedded' in the STS model owing to the synchronization of **P** and **S**. We call this disablement *preliminary control*, and thus distinguish it from

[5]This disablement could cause blocking, which will nevertheless be resolved when achieving a non-blocking SFBC implementation.

those control actions obtained from supervisor synthesis. In general, let $\mathbf{v_S}$ be the set of state variables of the specification $\mathbf{S}$; and for $\sigma \in \Sigma_c$ define the *preliminary disablement predicate* $PD_\sigma : \mathcal{B}(\mathbf{S}T) \to \{0,1\}$ according to

$$PD_\sigma := \big(\neg Elig_{\mathbf{G}}(\sigma)\big) \wedge \big(\exists \mathbf{v}_m Elig_{\mathbf{G}}(\sigma)\big)$$

Thus PD_σ is the characteristic function of the set of basic state trees where σ is not eligible to occur in $\mathbf{G}$, but can occur when considering the uncontrolled plant alone. For Small Factory with the buffer specification above, we have

$$\neg Elig_{\mathbf{G}}(\alpha_2) = (v_{x_2} = x_{20} \ \wedge \ v_y = y_0) \ \vee \ (v_{x_2} = x_{21} \ \wedge \ v_y = y_0) \ \vee \ (v_{x_2} = x_{21} \ \wedge \ v_y = y_1)$$

and

$$\begin{aligned} \exists v_y Elig_{\mathbf{G}}(\alpha_2) &:= \exists v_y (v_{x_2} = x_{20} \ \wedge \ v_y = y_1) \\ &\equiv (v_{x_2} = x_{20} \ \wedge \ y_0 = y_1) \ \vee \ (v_{x_2} = x_{20} \ \wedge \ y_1 = y_1) \\ &\equiv v_{x_2} = x_{20} \end{aligned}$$

Therefore,

$$\begin{aligned} PD_{\alpha_2} &:= \neg Elig_{\mathbf{G}}(\alpha_2) \ \wedge \ \exists v_y Elig_{\mathbf{G}}(\alpha_2) \\ &\equiv (v_{x_2} = x_{20} \ \wedge \ v_y = y_0) \ \vee \ false \ \vee \ false \\ &\equiv v_{x_2} = x_{20} \ \wedge \ v_y = y_0 \end{aligned}$$

By inspection of STS, Fig. 4.4, one can verify that PD_{α_2} is the characteristic function of the basic state tree where α_2 is blocked when synchronizing **M2** and **BUF**.

4.3 Problem Statement

Given a plant generator $\mathbf{P}$ to be controlled, consider the case where $\mathbf{P}$ consists of component agents $\mathbf{P}^k$ defined over pairwise disjoint alphabets Σ^k ($k \in K, K$ an index set):

$$\Sigma = \dot{\bigcup}\{\Sigma^k | k \in K\}$$

With $\Sigma = \Sigma_c \,\dot{\cup}\, \Sigma_u$ we assign control structure to each agent:

$$\Sigma_c^k = \Sigma^k \cap \Sigma_c \ , \ \Sigma_u^k = \Sigma^k \cap \Sigma_u$$

Also we assume a specification generator $\mathbf{S}$ is given that (as usual) imposes a behavioral constraint on $\mathbf{P}$.

As demonstrated in Remark 4.4, we convert this pair $(\mathbf{P}, \mathbf{S})$ into an STS model $\mathbf{G}$ with a predicate P specifying the illegal basic state trees; we then synthesize the supremal weakly controllable and coreachable subpredicate of $\neg P$, denoted by $sup\mathcal{C}^2\mathcal{P}(\neg P)$. Let

$$C := R(\mathbf{G}, sup\mathcal{C}^2\mathcal{P}(\neg P))$$

Thus C is the optimal nonblocking supervisor for the control problem $(\mathbf{G}, P)$. On the one hand, C is the characteristic function of the subset of basic state trees

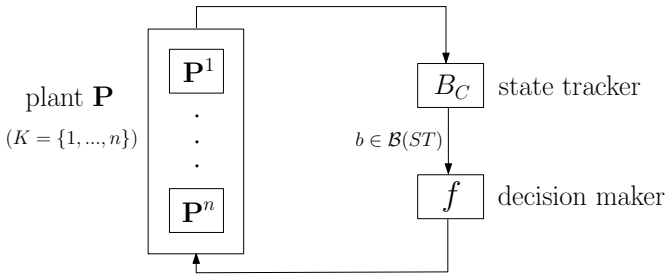
$$B_C := \{b \in \mathcal{B}(\mathbf{S}T) \mid b \models C\};$$

on the other hand, for C there exists a nonblocking SFBC $f : \mathcal{B}(\mathbf{S}T) \to \Pi$, where $\Pi := \{\Sigma' \subseteq \Sigma | \Sigma_u \subseteq \Sigma'\}$, such that

$$R(\mathbf{G}^f, true) = C$$

We call B_C a (monolithic) *state tracker* which reports state evolution of the controlled

system, and call f a (monolithic) *decision maker* which issues disablement commands based on the current state B_C reports. With B_C and f, the control actions of C can be implemented in a centralized fashion, as displayed below. Therefore, supervisor local-



ization in the present STS setting involves localizing both the state tracker B_C and the decision maker f.

The decision maker localization follows immediately from the fact that a SFBC f can be fully distributed to a set of control functions $\{f_\sigma | \sigma \in \Sigma\}$, where $f_\sigma : \mathcal{B}(\mathbf{S}T) \to \{0, 1\}$ is defined according to $f_\sigma(b) = 1$ iff $\sigma \in f(b)$. Since $f_\sigma(b)$ always holds for $\sigma \in \Sigma_u$, we will consider only the set $\{f_\sigma | \sigma \in \Sigma_c\}$. Let $\sigma \in \Sigma_c$, and recall that $Next_{\mathbf{G}}(\sigma)$ denotes the largest sub-state-tree of $\mathbf{S}T$ in $\mathbf{G}$ that is targeted by σ. Following [37, Section 4.4], we first divide the predicate $\Theta(Next_{\mathbf{G}}(\sigma))$ into the following two subpredicates:

$$N_{good} := \Theta(Next_{\mathbf{G}}(\sigma)) \wedge C$$

the legal subpredicate of $\Theta(Next_{\mathbf{G}}(\sigma))$, and

$$N_{bad} := \Theta(Next_{\mathbf{G}}(\sigma)) \wedge \neg C$$

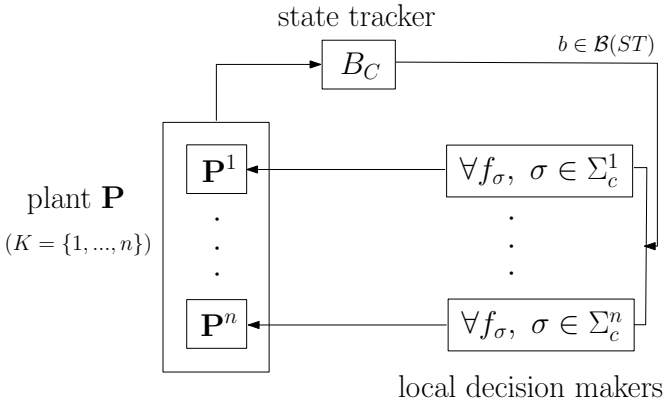
the illegal subpredicate of $\Theta(Next_{\mathbf{G}}(\sigma))$. Then we define the control function $f_\sigma : \mathcal{B}(\mathbf{S}T) \to \{0, 1\}$ by

$$f_\sigma := \hat{\Gamma}(N_{good}, \sigma);$$

namely, for every basic state tree $b \in \mathcal{B}(\mathbf{S}T)$

$$f_\sigma(b) = \begin{cases} 1, & \text{if } \Delta(b,\sigma) \models N_{good} \\ 0, & \text{if either } \Delta(b,\sigma) \models N_{bad} \text{ or } \Delta(b,\sigma) = \emptyset \end{cases}$$

With this set of localized decision makers $\{f_\sigma | \sigma \in \Sigma_c\}$ and the monolithic state tracker B_C, the supervisory control can now be implemented as follows.



We still need to localize the state tracker B_C. In analogy to the approach in Chapter 2, for each $k \in K$ we will establish a *control cover* on B_C, denoted by

$$\mathcal{C}^k := \{B^k_{i^k} \subseteq B_C | i^k \in I^k\}$$

where $B^k_{i^k}$ is a cell of $\mathcal{C}^k$ labeled i^k, and I^k is an index set. Thus B_C with $\mathcal{C}^k$ can be viewed as another state tracker, written B^k_C, that reports system state evolution in terms of cells (subsets) of basic state trees in $\mathbf{G}$, rather than just singleton basic state trees; to put it another way, B^k_C can distinguish only different cells of $\mathcal{C}^k$, but not different basic state trees in the same cell.

To be compatible with this state tracker B^k_C, the foregoing local decision makers f_σ must be extended to handle subsets of basic state trees. Such an extension makes sense only when it is defined over those subsets of basic state trees whose elements have *consistent control information*. With this in mind, for $\sigma \in \Sigma_c$ we define the extended

control function $\hat{f}_\sigma : Pwr(\mathcal{B}(\mathbf{S}T)) \to \{0,1\}$ (pfn) such that for all $B \in Pwr(\mathcal{B}(\mathbf{S}T))$

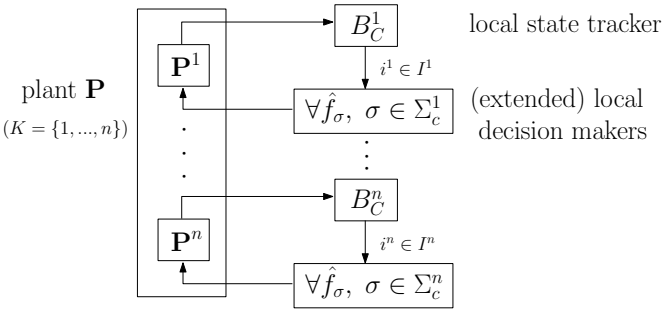
$$\hat{f}_\sigma(B) := \begin{cases} 1, & \text{if } \big[(\forall b \in B) b \models f_\sigma \vee \neg Elig_G(\sigma)\big] \ \& \ \big[(\exists b' \in B) b' \models f_\sigma\big] \\ 0, & \text{if } (\forall b \in B) b \models \neg f_\sigma \\ \text{undefined}, & \text{otherwise, i.e., } \big[(\exists b \in B) b \models f_\sigma\big] \ \& \ \big[(\exists b' \in B) b' \models Elig_G(\sigma) \wedge \neg f_\sigma\big] \end{cases}$$

Thus $\hat{f}_\sigma$ is not defined for any B having two elements (b and b'), at one of which σ must be enabled ($b \models f_\sigma$) while at the other σ must be disabled ($b' \models Elig_G(\sigma) \wedge \neg f_\sigma$). Otherwise $\hat{f}_\sigma$ is defined: B is evaluated to be false if all of its members fail to satisfy f_σ ($b \models \neg f_\sigma$); B is evaluated to be true if all of its members satisfy f_σ ($b \models f_\sigma$.) In addition, we know that if σ is not eligible at a basic state tree b (i.e. $b \models \neg Elig_G(\sigma)$), then b can be regarded as having consistent control information with any other basic state tree. Thus we also declare $\hat{f}_\sigma(B) = 1$ in case B contains a nonempty subset of elements that satisfy f_σ, and at the remaining elements of B, σ is not eligible.

Subsequently, we say B_C^k is a *local state tracker* for agent $\mathbf{P}^k$ if for all $\sigma \in \Sigma_c^k$, $\hat{f}_\sigma$ is defined for every cell of $\mathcal{C}^k$; namely,

$$(\forall \sigma \in \Sigma_c^k, \forall i^k \in I^k)\ \hat{f}_\sigma(B_{i^k}^k) \text{ is defined}$$

So with a set of local state trackers $\{B_C^k | k \in K\}$ and the set of extended local decision makers $\{\hat{f}_\sigma | \sigma \in \Sigma_c\}$, the supervisory control can be implemented in the following distributed manner.



Of central importance for this distributed implementation is to preserve the optimality and nonblocking properties of the monolithic supervisory control. Let $k \in K$ and $\sigma \in \Sigma_c^k$. Suppose the controlled system is currently visiting a basic state tree $b \in B_C$; thus there must exist a cell $B_{i^k}^k$ of the cover $\mathcal{C}^k$ to which b belongs. As displayed in Fig. 4.6, on the one hand, the monolithic state tracker reports b to f_σ which then makes the control decision $f_\sigma(b)$; on the other hand, a local state tracker reports the whole cell $B_{i_k}^k$ to $\hat{f}_\sigma$ which then makes the control decision $\hat{f}_\sigma(B_{i_k}^k)$. We say these two pairs (B_C, f_σ) and

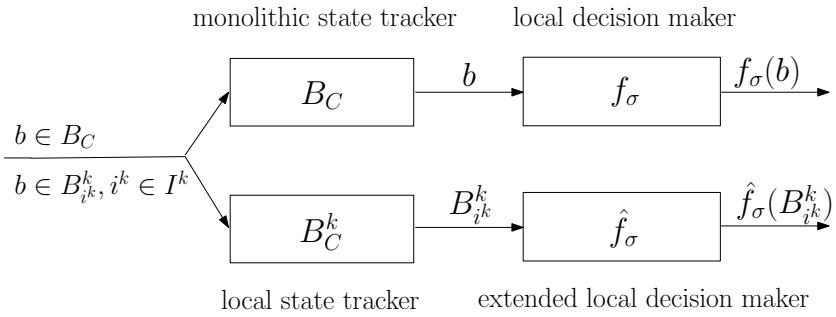


Figure 4.6: Control equivalence in STS framework

$(B_C^k, \hat{f}_\sigma)$ are *control equivalent* whenever the following holds:

$$\Delta(b, \sigma) \neq \emptyset \quad \Rightarrow \quad f_\sigma(b) = 1 \text{ iff } \hat{f}_\sigma(B_{i_k}^k) = 1$$

Now we can formulate the *Distributed Optimal Nonblocking Control Problem* (⋇):

> Given a (plant, specification) pair $(\mathbf{P}, \mathbf{S})$, obtain its STS counterpart $(\mathbf{G}, P)$ and the corresponding optimal nonblocking supervisory predicate C (implemented by B_C and $\{f_\sigma | \sigma \in \Sigma_c\}$). Construct a set of local state trackers $\{B_C^k | k \in K\}$ with a corresponding set of (extended) local decision makers $\{\hat{f}_\sigma | \sigma \in \Sigma_c\}$ such that for all $k \in K$ and all $\sigma \in \Sigma_c^k$, the two pairs (B_C, f_σ) and $(B_C^k, \hat{f}_\sigma)$ are control equivalent.

4.4 Supervisor Localization

We solve (⋇) by developing a supervisor localization procedure closely analogous to that in Chapter 2.

It follows from $\Sigma = \dot{\bigcup}\{\Sigma^k | k \in K\}$ that the set $\{\Sigma_c^k \subseteq \Sigma_c | k \in K\}$ forms a partition on Σ_c. Fix an element $k \in K$. We first establish a *control cover* on B_C based only on control information pertaining to Σ_c^k, as captured by the following two functions. Let $\sigma \in \Sigma_c^k$. First define $E_\sigma : B_C \to \{0,1\}$ by

$$E_\sigma := \hat{\Gamma}(N_{good}, \sigma) \wedge C$$

Thus E_σ is the characteristic function of the set of basic state trees in B_C where σ is enabled. Notice that E_σ is actually the restriction of the control function f_σ from $\mathcal{B}(\mathbf{S}T)$ to B_C. Next define $D_\sigma : B_C \to \{0,1\}$ by

$$D_\sigma := \left[PD_\sigma \vee \hat{\Gamma}(N_{bad}, \sigma)\right] \wedge C$$

That is, D_σ is the characteristic function of the set of basic state trees in B_C where σ must be disabled, either by preliminary disablement in the STS model $\mathbf{G}$ or by the supervisory control of C.

Definition 4.6.

We define a binary relation $\mathcal{R}^k$ on B_C as follows. Let $b, b' \in B_C$. We say b and b' are *control consistent* (with respect to Σ_c^k), and write $(b, b') \in \mathcal{R}^k$, if

$$(\forall \sigma \in \Sigma_c^k) \;\; E_\sigma(b) \wedge D_\sigma(b') \equiv false \equiv E_\sigma(b') \wedge D_\sigma(b)$$

$\Diamond$

Informally, a pair of basic state trees (b, b') is in $\mathcal{R}^k$ if there is no event in Σ^k that is

enabled at b but is disabled at b', or vice versa.

Like its counterpart definition in the language-based framework, $\mathcal{R}^k$ is a tolerance relation on B_C, namely it is reflexive and symmetric but in general need not be transitive. Thus, $\mathcal{R}^k$ is generally not an equivalence relation. This fact leads to the following definition of control cover (with respect to Σ_c^k).

Definition 4.7.

Let I^k be some index set, and $\mathcal{C}^k = \{B_{i^k}^k \subseteq B_C | i^k \in I^k\}$ be a cover on B_C. $\mathcal{C}^k$ is a *control cover* on B_C (with respect to Σ_c^k) if

(i) $(\forall i^k \in I^k, \forall b, b' \in B_{i^k}^k)\ (b, b') \in \mathcal{R}^k$

(ii) $(\forall i^k \in I^k, \forall \sigma \in \Sigma)\big[(\exists j^k \in I^k)(\forall b \in B_{i^k}^k)\Delta(b,\sigma) \neq \emptyset\ \&\ \Delta(b,\sigma) \in B_C \Rightarrow \Delta(b,\sigma) \in B_{j^k}^k\big]$ $\Diamond$

Informally, a control cover $\mathcal{C}^k$ groups basic state trees in B_C into (possibly overlapping) cells $B_{i^k}^k$ ($i^k \in I^k$). According to (i) all basic state trees that reside in a cell $B_{i^k}^k$ have to be pairwise control consistent; and (ii) for each event $\sigma \in \Sigma$, all basic state trees that can be reached from any basic state trees in $B_{i^k}^k$ by a one-step transition σ have to be covered by a certain cell $B_{j^k}^k$. Recursively, two basic state trees b, b' belong to a common cell in $\mathcal{C}^k$ if and only if (1) b and b' are control consistent; and (2) two future states that can be reached from b and b', respectively, by the same string are again control consistent. In addition we say that a control cover $\mathcal{C}^k$ is a *control congruence* if $\mathcal{C}^k$ happens to be a partition on B_C.

Thus B_C with a control cover $\mathcal{C}^k$ can be viewed as a state tracker, written B_C^k, that reports system state evolution in terms of cells (subsets) of basic state trees. We proceed to derive the dynamics of B_C^k through constructing the induced generator $B_C^k = (I^k, \Sigma, \kappa^k, i_0^k, I_m^k)$ from B_C and $\mathcal{C}^k$ as follows:

(i) $i_0^k \in I^k$ such that $b_0 \in B_{i_0^k}^k$

(ii) $I^k_m = \{i^k \in I^k | B^k_{i^k} \cap \mathcal{B}(\mathbf{S}T_m) \neq \emptyset\}$

(iii) $\kappa^k : I^k \times \Sigma \to I^k$(pfn) with $\kappa^k(i^k, \sigma) = j^k$ if

$$(\exists b \in B^k_{i^k})\Delta(b, \sigma) \in B^k_{j^k} \;\&$$
$$(\forall b' \in B^k_{i^k}) \left[\Delta(b', \sigma) \neq \emptyset \;\&\; \Delta(b', \sigma) \in B_C \Rightarrow \Delta(b', \sigma) \in B^k_{j^k}\right]$$

Here b_0 is the initial basic state tree, and $\mathcal{B}(\mathbf{S}T_m)$ is the set of marked basic state trees. Note that, owing to overlapping, the choices of i^k_0 and κ^k may not be unique, and consequently B^k_C may not be unique. In that case we pick an arbitrary instance of B^k_C. Clearly if $\mathcal{C}^k$ happens to be a control congruence, then B^k_C is unique.

Recall that an extended local decision maker $\hat{f}_\sigma$ is a partial function defined on $Pwr(\mathcal{B}(\mathbf{S}T))$. Our first result shows that for all $\sigma \in \Sigma^k_c$, $\hat{f}_\sigma$ is defined for every cell of $\mathcal{C}^k$, namely B^k_C is a *local state tracker* for agent $\mathbf{P}^k$.

Proposition 4.1.

Let $B^k_C = (I^k, \Sigma, \kappa^k, i^k_0, I^k_m)$ be induced from B_C and $\mathcal{C}^k$. Then for all $\sigma \in \Sigma^k_c$ and all $i^k \in I^k$, $\hat{f}_\sigma(B^k_{i^k})$ is defined.

Proof.

Let $\sigma \in \Sigma^k$ and $i^k \in I^k$. We suppose $\hat{f}_\sigma(B^k_{i^k})$ is *not* defined. Then by the (structural) definition of $\hat{f}_\sigma$, there exist $b, b' \in B^k_{i^k}$ such that

$$b \models f_\sigma \text{ and } b' \models Elig_{\mathbf{G}}(\sigma) \wedge \neg f_\sigma$$

i.e.,

$$b \models \hat{\Gamma}(N_{good}, \sigma) \text{ and } b' \models \hat{\Gamma}(N_{bad}, \sigma)$$

It follows from $b, b' \in B^k_{i^k} \subseteq B_C$ that

$$b \models \hat{\Gamma}(N_{good}, \sigma) \wedge C \text{ and } b' \models \hat{\Gamma}(N_{bad}, \sigma) \wedge C$$

Hence $E_\sigma(b) = 1$ and $D_\sigma(b') = 1$. So $E_\sigma(b) \wedge D_\sigma(b') \equiv true$, which implies $(b, b') \notin \mathcal{R}^k$. This contradicts that $b, b' \in B^k_{i^k}$, and therefore $\hat{f}_\sigma(B^k_{i^k})$ is defined after all. □

Now let $\{B^k_C | k \in K\}$ be a set of local state trackers for the partition $\{\Sigma^k_c \subseteq \Sigma_c | k \in K\}$. Then $\{B^k_C | k \in K\}$ with a corresponding set of extended local decision makers $\{\hat{f}_\sigma | \sigma \in \Sigma_c\}$ solves (⋇).

Proposition 4.2.

For all $k \in K$ and all $\sigma \in \Sigma^k_c$, the two pairs (B_C, f_σ) and $(B^k_C, \hat{f}_\sigma)$ are control equivalent.

Proof.

Let $k \in K$ and $\sigma \in \Sigma^k_c$. Pick a basic state tree $b \in B_C$ such that $\Delta(b, \sigma) \neq \emptyset$; then there must exist a cell $B^k_{i^k}$ in a control cover $\mathcal{C}^k$ on B_C (with respect to Σ^k_c) such that $b \in B^k_{i^k}$. It will be shown that

$$f_\sigma(b) = 1 \text{ iff } \hat{f}_\sigma(B^k_{i_k}) = 1$$

(if) Assume $\hat{f}_\sigma(B^k_{i_k}) = 1$. Then by the definition of $\hat{f}_\sigma$, there must exist $b' \in B^k_{i^k}$ such that $b' \models f_\sigma$, which implies that $E_\sigma(b') = 1$. Since b is also in $B^k_{i^k}$, $(b, b') \in \mathcal{R}^k$. Then it follows from $E_\sigma(b') \wedge D_\sigma(b) \equiv false$ that $D_\sigma(b) = 0$, i.e.,

$$\begin{aligned} b &\models \neg \Big(\big(PD_\sigma \vee \hat{\Gamma}(N_{bad,\sigma}) \big) \wedge C \Big) \\ &\models \big(\neg PD_\sigma \wedge \neg \hat{\Gamma}(N_{bad,\sigma}) \big) \vee \neg C \end{aligned}$$

We have that b is in B_C (i.e. $b \models C$); so

$$b \models \neg PD_\sigma \wedge \neg \hat{\Gamma}(N_{bad,\sigma})$$

Besides, it follows from the definition of f_σ that

$$\neg\hat{\Gamma}(N_{good},\sigma) \equiv \hat{\Gamma}(N_{bad},\sigma) \vee \neg Elig_G(\sigma)$$

Equivalently,

$$\hat{\Gamma}(N_{bad},\sigma) \equiv \neg\hat{\Gamma}(N_{good},\sigma) \wedge Elig_G(\sigma)$$

Hence,

$$\begin{aligned} b &\models \neg PD_\sigma \wedge \neg\big(\neg\hat{\Gamma}(N_{good},\sigma) \wedge Elig_G(\sigma)\big) \\ &\models \neg PD_\sigma \wedge \big(\hat{\Gamma}(N_{good},\sigma) \vee \neg Elig_G(\sigma)\big) \end{aligned}$$

We know from the hypothesis $\Delta(b,\sigma) \neq \emptyset$ that σ is not preliminarily disabled at b ($b \models \neg PD_\sigma$), and σ is actually defined at b (i.e. $b \models Elig_G(\sigma)$). Therefore $b \models \hat{\Gamma}(N_{good},\sigma)$; namely $f_\sigma(b) = 1$.

(only if) Assume $f_\sigma(b) = 1$ (i.e. $b \models \hat{\Gamma}(N_{good},\sigma)$). If follows from $b \models C$ that $E_\sigma(b) = 1$. Let b' be an arbitrary element in $B^k_{i^k}$ that is distinct from b. Then $(b,b') \in \mathcal{R}^k$ and $E_\sigma(b) \wedge D_\sigma(b') \equiv false$. So $D_\sigma(b') = 0$, and as above,

$$b' \models \neg PD_\sigma \wedge \big(\hat{\Gamma}(N_{good},\sigma) \vee \neg Elig_G(\sigma)\big)$$

Thus in this cell $B^k_{i^k}$, we have $b \models f_\sigma$ and all other elements $b' \models \hat{\Gamma}(N_{good},\sigma) \vee \neg Elig_G(\sigma)$ (they are not preliminarily disabled). By the definition of $\hat{f}_\sigma$, we conclude that $\hat{f}_\sigma(B^k_{i^k}) = 1$. □

Next we investigate whether or not the converse is true: for $k \in K$ and $\sigma \in \Sigma^k_c$, if a pair $(B^k_C, \hat{f}_\sigma)$ is control equivalent to the pair (B_C, f_σ), can the local state tracker B^k_C always be induced from a suitable control cover on B_C? In response, we bring in the notion of *normality*.

Definition 4.8.

A local state tracker $B_C^k = (I^k, \Sigma, \kappa^k, i_0^k, I_m^k)$ is *normal* (with respect to B_C) if

(i) $\{B_{i^k}^k | i^k \in I^k\}$ is a cover on B_C

(ii) $i_0^k \in I^k$ such that $b_0 \in B_{i_0^k}^k$

(iii) $I_m^k = \{i^k \in I^k | B_{i^k}^k \cap \mathcal{B}(\mathbf{S}T_m) \neq \emptyset\}$

(iv) $\kappa^k : I^k \times \Sigma \to I^k$(pfn) with $\kappa^k(i^k, \sigma) = j^k$ if

$$(\exists b \in B_{i^k}^k)\Delta(b, \sigma) \in B_{j^k}^k \;\&$$
$$(\forall b' \in B_{i^k}^k)\ \left[\Delta(b', \sigma) \neq \emptyset \;\&\; \Delta(b', \sigma) \in B_C \Rightarrow \Delta(b', \sigma) \in B_{j^k}^k\right]$$

Here b_0 is the initial basic state tree and $\mathcal{B}(\mathbf{S}T_m)$ is the set of marked basic state trees.

$\Diamond$

Informally, a local state tracker will be normal (with respect to B_C) whenever it is induced from some cover on B_C. Under normality, we have the following result in response to the converse question posed above.

Theorem 4.1.

Suppose that, for all $k \in K$ and $\sigma \in \Sigma_c^k$, a normal local state tracker $B_C^k = (I^k, \Sigma, \kappa^k, i_0^k, I_m^k)$ with a corresponding extended local decision maker $\hat{f}_\sigma$ is control equivalent to the pair (B_C, f_σ). Then there exists a control cover on B_C from which B_C^k can be induced.

Proof.

Let $k \in K$ and $\sigma \in \Sigma_c^k$. By normality, B_C^k is induced from a cover $\mathcal{C}^k = \{B_{i^k}^k | i^k \in I^k\}$ on B_C. It will be shown that $\mathcal{C}^k$ is a control cover.

First, we prove the second condition in the definition of control cover. Let $i^k \in I^k$, and $b \in B_C$ with $\Delta(b, \sigma) \neq \emptyset$ and $\Delta(b, \sigma) \in B_C$. So by (iv) of normality, the transition $\kappa^k(i^k, \sigma)$ is defined, and there exists $j^k \in I^k$ such that $\Delta(b, \sigma) \in B_{j^k}^k$.

We are left to show that $(b, b') \in \mathcal{R}^k$ whenever $b, b' \in B^k_{i^k}$ ($i^k \in I^k$). If σ is not defined at either b or b', or both of them, then they are trivially control consistent. Otherwise (i.e. $\Delta(b, \sigma) \neq \emptyset$ and $\Delta(b', \sigma) \neq \emptyset$), by the assumption that (B_C, f_σ) and $(B^k_C, \hat{f}_\sigma)$ are control equivalent, we derive that $f_\sigma(b) = 1$ iff $\hat{f}_\sigma(i^k) = 1$ and $f_\sigma(b') = 1$ iff $\hat{f}_\sigma(i^k) = 1$. Hence $f_\sigma(b) = 1$ iff $f_\sigma(b') = 1$, which implies that $E_\sigma(b) = 1$ iff $E_\sigma(b') = 1$. It then follows that $E_\sigma(b) \wedge D_\sigma(b') \equiv false \equiv E_\sigma(b') \wedge D_\sigma(b)$, i.e., $(b, b') \in \mathcal{R}^k$. We conclude that $\mathcal{C}^k$ is a control cover. □

To summarize, every set of control covers generates a solution to (⋇) (Proposition 4.2); and every solution to (⋇) can be induced from some set of control covers (Theorem 4.1). In particular, a set of *state-minimal* local state trackers can be induced from a set of suitable control covers. In agreement with the conclusion in Chapter 2, however, such a set is in general not unique, and the problem of finding such a set is NP-hard.

4.5 Symbolic Localization Algorithm

Following the idea of the localization algorithm presented in Section 2.4, we propose another polynomial-time algorithm in the present STS setting that can accomplish state tracker localization.

Let $(\mathbf{G}, P)$ be the STS counterpart of a given control problem $(\mathbf{P}, \mathbf{S})$, and assume that $\mathbf{G}$ is defined over $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. For $\sigma \in \Sigma_c$ let PD_σ be the preliminary disablement predicate; and let C be the monolithic supervisory predicate synthesized from P, with $B_C = \{b_0, b_1, ..., b_{n-1}\}$ the corresponding monolithic state tracker. Then our objective is to localize this B_C: for every agent $k \in K$ with controllable event set Σ_c^k, generate a control cover (a control congruence in our algorithm) on B_C with respect to the control information pertaining to Σ_c^k.

Recall that $\mathcal{R}^k$ is the control consistency binary relation (with respect to Σ_c^k) on B_C; for $b_1, b_2 \in B_C$, $(b_1, b_2) \in \mathcal{R}^k$ if for all $\sigma \in \Sigma_c^k$,

$$E_\sigma(b_1) \wedge D_\sigma(b_2) \equiv false \equiv E_\sigma(b_2) \wedge D_\sigma(b_1)$$

where

$$\begin{aligned} E_\sigma &= \hat{\Gamma}(N_{good},\ \sigma) \wedge C \\ &= \hat{\Gamma}(\Theta(Next_{\mathbf{G}}(\sigma)) \wedge C,\ \sigma) \wedge C \end{aligned}$$

and

$$\begin{aligned} D_\sigma &= \left[PD_\sigma \vee \hat{\Gamma}(N_{bad},\ \sigma)\right] \wedge C \\ &= \left[PD_\sigma \vee \hat{\Gamma}(\Theta(Next_{\mathbf{G}}(\sigma)) \wedge \neg C,\ \sigma)\right] \wedge C \end{aligned}$$

Next we define a predicate $\mathscr{R}^k : Pwr(B_C) \to \{0, 1\}$ such that for all $B \in Pwr(B_C)$, $\mathscr{R}^k(B) = 1$ iff $(\forall b, b' \in B)(b_1, b_2) \in \mathcal{R}^k$. We symbolically implement $\mathscr{R}^k$ instead of $\mathcal{R}^k$

(see lines 11-13 in the pseudocode below); thus a subset of basic state trees can be tested for control consistency in a single predicate evaluation.

Notation: wl is a list of subsets of basic state trees whose mergibility is pending.

Symbolic Localization Algorithm (SLA)

```
1  int main()
2  C^k = {c_0, c_1, ..., c_{n-1}} (initialize C^k with c_l = b_l for l ∈ [0, n-1])
3  for i : 0 to |C^k| - 2 do
4      for j : i + 1 to |C^k| - 1 do
5          cell = c_i ∨ c_j;
6          wl = ∅;
7          if Check_Mergibility(cell, wl, i, C^k) = true then
           C^k = (C^k ∪ wl) − {x|(∃y ∈ wl)x ≺ y};
8      end
9  end
10 bool Check_Mergibility(cell, wl, i, C^k)
11 for each pair of basic state trees b_1, b_2 ∈ {b ∈ B_C|b ⊨ cell} do
12     if (b_1, b_2) ∉ R^k then return false
13 end
14 wl = (wl ∪ {cell}) − {x|x ≺ cell} ;
15 for each σ ∈ Σ with Δ(cell, σ) ∧ C ≠ 0 do
16     if (Δ(cell, σ) ∧ C) ⪯ x for some x ∈ C^k ∪ wl then continue;
17     if (Δ(cell, σ) ∧ C) ∧ x_r ≠ 0 for some r < i then return false;
18     new_cell = (Δ(cell, σ) ∧ C) ∨ (⋁_{x∈(C^k∪wl) & x∧(Δ(cell,σ)∧C)≠0} x) ;
19     if Check_Mergibility(new_cell, wl, i, C^k) = false then return false;
20 end
21 return true;
```

Proposition 4.3.

SLA terminates and the resulting $\mathcal{C}^k$ is a control congruence.

Proof. Lines 6, 14 and 18 guarantee that each cell of wl is the union of cells of $\mathcal{C}^k$. So whenever two cells of $\mathcal{C}^k$ can be merged together, the size of the updated $\mathcal{C}^k$ is nonincreasing (see line 7). Hence, the algorithm must terminate.

It is left to show that the resulting $\mathcal{C}^k$ is a control congruence. Initially, $\mathcal{C}^k$ is the set of singleton basic state trees of B_C, thus is trivially a control congruence. Notice that $\mathcal{C}^k$

is updated only at line 7 if the function *Check_Mergibility* returns *true*. In that case, wl must have the following properties:

1. By lines 11 and 12 every cell of wl satisfies the predicate $\mathscr{R}^k$.

2. By line 16 every cell's downstream cell $(\Delta(cell, \sigma) \wedge C)$ must reside in a cell either of $\mathcal{C}^k$ or of wl.

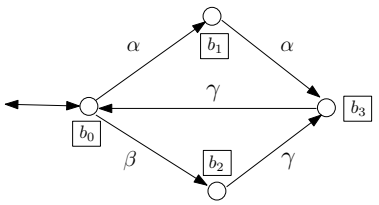
3. By lines 14 and 18 every two cells of wl must be disjoint.

4. Again by lines 14 and 18 every cell of wl is the union of some cells of $\mathcal{C}^k$.

Thus, properties 1 and 2 ensure that the updated $\mathcal{C}^k$ is a control cover; properties 3 and 4 ensure that every two cells of the updated $\mathcal{C}^k$ must be disjoint. Therefore, $\mathcal{C}^k$ is a control congruence. □

Remark 4.5. The size of the initial $\mathcal{C}^k$ is n. In the worst case $\frac{1}{2}n(n-1)$ calls can be made to the function *Check_Mergibility*, which can then make $n-2$ calls to itself. Therefore SLA has the time complexity $O(n^3)$, slightly more efficient than the localization algorithm in Section 2.4 which is $O(n^4)$. This improvement is due to the fact that by the symbolic approach we can check the mergibility directly for a pair of cells, rather than just a pair of singleton basic state trees.

Example 4.3.



$\Sigma_c^k = \{\alpha\}$
α is disabled at b_3

$B_C = \{b_0, b_1, b_2, b_3\}$ $\qquad f_\alpha(b_n) := \begin{cases} 0, & \text{if } n = 0, 1 \\ 1, & \text{if } n = 2, 3 \end{cases}$

To illustrate SLA, we again use Example 2.4, but in the STS setting.

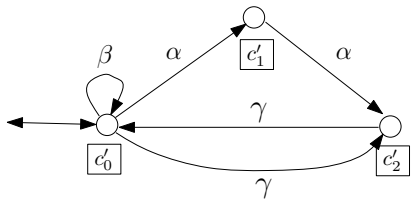
(0) Initially, $\mathcal{C}^k_{init} = \{c_0, c_1, c_2, c_3\}$ with $c_l = b_l$ for $l \in [0, 3]$. Thus at line 3, the index i ranges from 0 to $|\mathcal{C}^k_{init}| - 2$, i.e., from 0 to 2.

(1) (c_0, c_1) cannot be merged: they pass lines 11 and 12 because $c_0 \vee c_1 \models \mathscr{R}^k$, but they fail at line 19 since

$$c_0 \vee c_1 \vee \Delta(c_0, \alpha) \vee \Delta(c_1, \alpha) \equiv c_0 \vee c_1 \vee c_2 \not\models \mathscr{R}^k;$$

(c_0, c_2) can be merged: they pass lines 11 and 12 because $c_0 \vee c_2 \models \mathscr{R}^k$, and they trivially pass line 15 since there is no common event defined on them, so that no further control consistency needs to be verified; (c_0, c_3) cannot be merged: they fail at line 12, for $c_0 \vee c_2 \not\models \mathscr{R}^k$. So $\mathcal{C}^k_1 = \{c'_0, c'_1, c'_2\}$ with $c'_0 := c_0 \vee c_2, c'_1 := c_1$, and $c'_2 := c_3$. Now at line 3, the index i ranges from 1 to $|\mathcal{C}^k_{init}| - 2$, i.e., just 1.

(2) (c'_1, c'_2) cannot be merged: they failed at line 12, since $c'_1 \vee c'_2 \not\models \mathscr{R}^k$. Thus the final cover is $\mathcal{C}^k_2 = \mathcal{C}^k_1 = \{b_0 \vee b_2, b_1, b_3\}$, as displayed below.

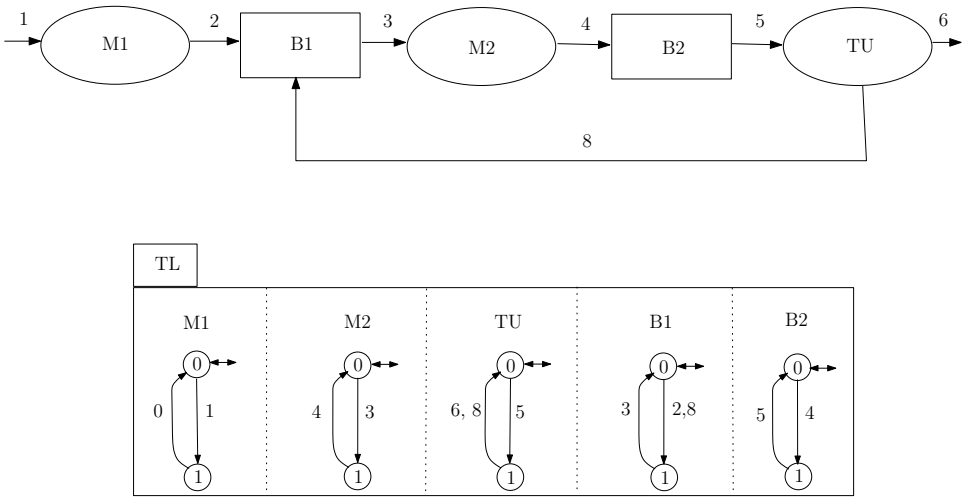


$$B^{loc}_C = \{c'_0, c'_1, c'_2\} = \{b_0 \vee b_2, b_1, b_3\} \qquad \hat{f}_\alpha(c'_n) := \begin{cases} 0, & \text{if } n = 2 \\ 1, & \text{if } n = 0, 1 \end{cases}$$

This result is the same as that of Example 2.4, being achieved with one less call to *Check_Mergibility* than with the algorithm in the language-based framework. ♦

4.6 Example: Transfer Line

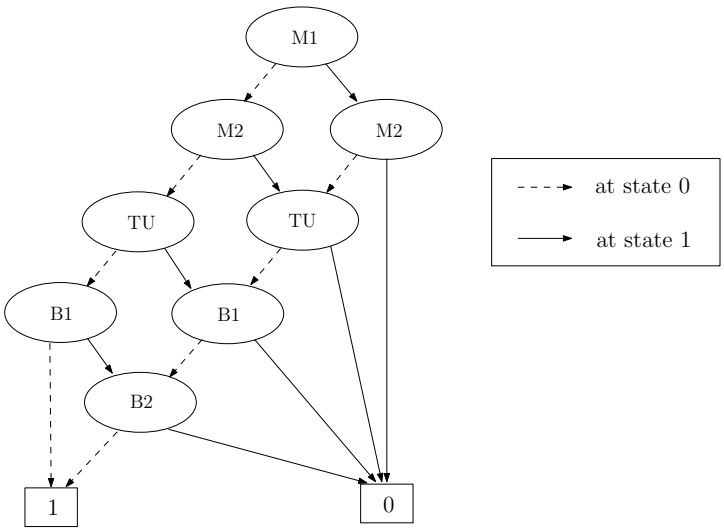
Let us revisit the transfer line system we discussed in Chapter 2, with one difference that here we let **B1** be a one-slot buffer. The STS model of this control problem is displayed



STS Model **G**

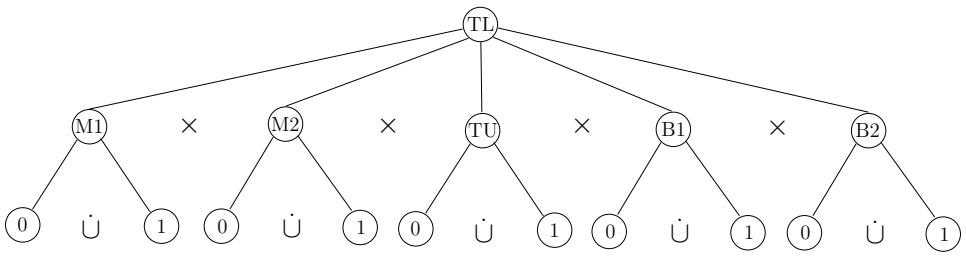
above. In the present state-based framework, the distributed control objective is to design for each component – **M1**, **M2**, and **TU** – a local state tracker with a corresponding (extended) local decision maker.

By the centralized symbolic synthesis [37], we first obtain the optimal nonblocking supervisory predicate $C = R(\mathbf{G}, sup\mathcal{C}^2\mathcal{P}(\neg P))$, where P is the illegal predicate. The BDD representation of C is the following.



The monolithic supervisor C can be implemented by the monolithic state tracker B_C and (simplified) local decision makers [37, Section 4.5.2], as shown in Figs. 4.7 and 4.8.

Now we employ the symbolic localization algorithm to compute for each agent a local



$\mathbf{S}$T

Thus $B_C := \{b \in \mathcal{B}(\mathbf{S}T) \mid b \models C\}$

Figure 4.7: Monolithic state tracker

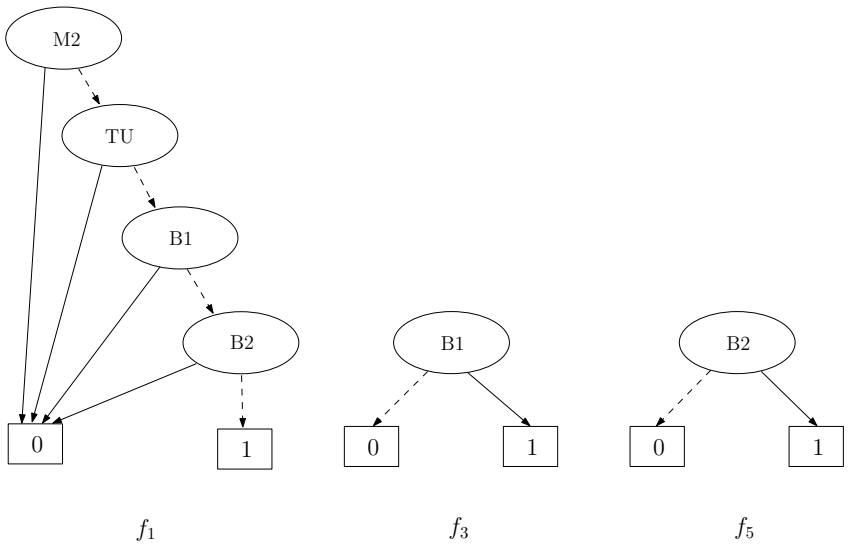


Figure 4.8: Local decision makers

state tracker from the global one. The resultant state trackers with their corresponding extended local decision makers are displayed in Fig. 4.9. Thus we have built a purely distributed control architecture, wherein every agent tracks system state evolution locally and makes corresponding decisions. Notice that, with the local state tracker B_C^1, the control logic of $\hat{f}_1$ is much simpler than that of f_1.

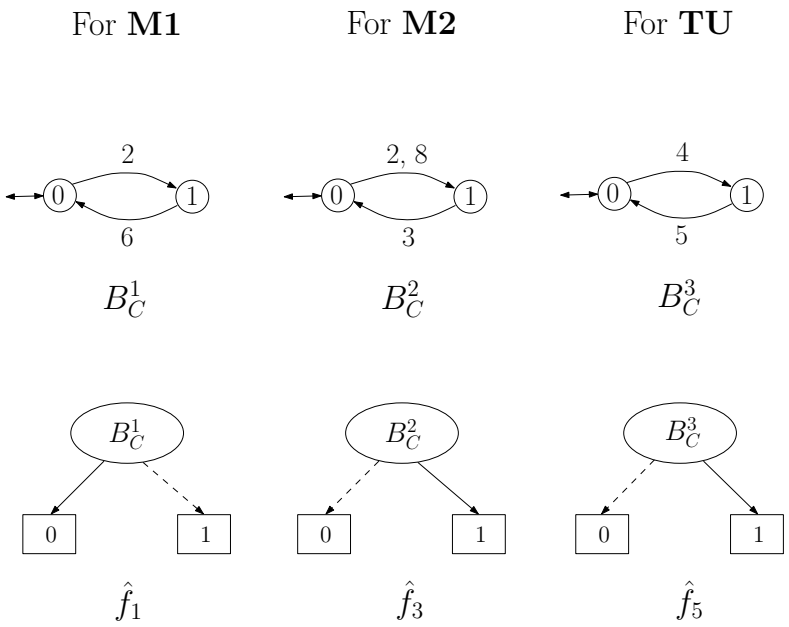


Figure 4.9: Local state trackers and extended local decision makers

Chapter 5

Conclusion

5.1 Thesis Summary

This thesis has initiated the study of distributed control design for DES in the SCT framework, DES that consist of independent agents whose coupling is due solely to imposed specifications. The central problem investigated is how to synthesize local controllers for individual agents such that these local controllers collectively realize controlled behavior identical to that achieved by an external (monolithic or modular) supervisor. The investigation has been carried out in both language- and state-based models.

In the language-based setting, a supervisor localization algorithm has been established that solves the problem in a top-down fashion: first compute the monolithic optimal nonblocking supervisor, and then decompose it into local controllers while preserving optimality and nonblockingness. Our localization algorithm generalizes a known supervisor reduction algorithm, with the new feature that it is conducted based solely on local control information. Furthermore, to tackle the case of large-scale DES where the monolithic supervisor is in general not feasibly computable owing to state explosion, we have proposed combining the (language-based) localization algorithm with an efficient modular control theory. This combination led us to a language-based decomposition-aggregation

procedure that systematically solves the large-system problem in an alternative top-down manner: first, design an organization of modular supervisors that achieves optimal nonblocking control, then decompose each of these modular supervisors into local controllers for the relevant agents.

Finally, the large-system problem was addressed in a state-based setting, specifically the state tree structure (STS), a formalism that is already known to be efficient for monolithic supervisor synthesis. In the thesis, a state-based counterpart to our language-based, top-down solution was obtained in the form of an STS-based supervisor localization algorithm.

5.2 Future Research

We suggest a few topics for future research arising from this thesis.

In Chapter 2 we developed a supervisor localization algorithm that not only preserves the optimality and nonblocking properties of monolithic control, but aims also to minimize the state size of the resulting local controllers in an effort to make their logic more comprehensible. Minimizing state size does not, however, directly address perhaps more intriguing issues regarding the observation scope of individual agents, namely identifying quantitative tradeoffs between information and control. Of particular interest would be to find the minimal amount of information (in some sense) necessary for individual agents collectively to achieve optimal nonblocking control. One approach could be to design an alternative supervisor localization algorithm that aimed to minimize the number of events observed by the resulting local controllers.

In Chapter 3 we proposed a systematic decomposition-aggregation procedure to tackle distributed control design for large-scale DES. A shortcoming of this procedure is that the decomposition steps rely heavily on heuristic analysis of the components' interconnection structure, and different ways of decomposition may well affect the efficiency of the ap-

proach. So it is desirable, both as a practical matter and as one of theoretical interests, to develop an effective decomposition method that can handle a rather general type of interconnection structure, thereby automating the decomposition-aggregation procedure as a whole.

In Chapter 4 we studied distributed control design in the state tree structure (STS) framework, in the hope of endowing our solution with STS's computational power. We established a counterpart supervisor localization algorithm that solves the distributed control problem in the same top-down fashion as that in Chapter 2. While monolithic supervisor synthesis can be performed very efficiently (even for systems of state size 10^{24} or more), the localization algorithm itself has time complexity $O(n^3)$, where n is the state size of the supervisor. This fact renders our solution inefficient when faced with large-scale DES. One direction of future work could be to design a localization algorithm that is polynomial in the number of BDD nodes of the monolithic supervisor, rather than in the number of its (flat) states. An alternative could be to adapt the decomposition-aggregation procedure directly to the STS formalism, thus tackling large problems systematically from the ground up.

Finally, our investigation on distributed control design for DES has added "purely distributed" architecture to the family consisting of "monolithic" and "modular" architectures. What are the advantages of our distributed architecture over the other two that could serve to motivate our efforts? As already remarked in the Introduction, it would be more convincing if we rigorously validated those intuitively envisaged benefits. To put it more generally, given a specific system with a particular task, we need to analyze quantitatively the tradeoffs among these three architectures, in such a way that we could soundly infer which one was the best suited to the task at hand. Such a "theory of architecture" would seem to be an ultimate objective of SCT.

Bibliography

[1] In P. Maes, editor, *Designing Autonomous Agents: Theory and Practice from Biology to Engineering and Back*. MIT Press, 1991.

[2] T. Arai, E. Pagello, and L. E. Parker. Guest editorial: Advances in multirobot systems. *IEEE Transactions on Robotics and Automation*, 18(5):655–661, Oct 2002.

[3] R. C. Arkin. Motor schema-based mobile robot navigation. *International Journal of Robotics Research*, 8(4):92–112, 1989.

[4] T. Balch. "Emergent" is not a four-letter word. In *Abstract accompanying the author's talk at the 2003 Block Island Workshop on Cooperative Control*, Block Island, Rhode Island, Jun 2003.

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

[6] M. Ben-Ari. *Principles of Concurrent and Distributed Programming*. Prentice Hall, 1990.

[7] Y. Brave and M. Heymann. Control of discrete event systems modeled as hierarchical state machines. *IEEE Transactions on Automatic Control*, 38(12):1803–1819, Dec 1993.

[8] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of robotics and automation*, RA-2(1):14–23, Mar 1986.

[9] R. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[10] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative mobile robotics: antecedents and directions. *Autonomous Robots*, 4(1):7–23, Mar 1997.

[11] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[12] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988.

[13] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, 2004.

[14] L. Feng. *Computationally Efficient Supervisory Design for Discrete-Event Systems*. PhD thesis, ECE Department, University of Toronto, 2007.

[15] L. Feng, K. Cai, and W. M. Wonham. A structural approach to the nonblocking supervisory control of discrete-event systems. *International Journal of Advanced Manufacturing Technology*, to appear, 2008.

[16] L. Feng and W. M. Wonham. Computationally efficient supervisory design: Abstraction and modularity. In *Proc. Int. Workshop Discrete Event Systems (WODES06)*, pages 3–8, Ann Arbor, Michigan, U.S.A., Jul 2006.

[17] L. Feng and W. M. Wonham. Supervisory control architecture for discrete-event systems. *IEEE Transactions on Automatic Control*, to appear, Jun 2008.

[18] Peyman Gohari and W. M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man, and Cybernetics, Special Issue on DES*, 30(5):643–652, 2000.

[19] H. Goldstein. Cure for the multicore blues. *IEEE Spectrum*, 44(1):40–43, Jan 2007.

[20] X. Guan and L. E. Holloway. Distributed discrete event control structures with controller interactions. In *Proc. of the American Control Conference*, pages 3151–3156, Seattle, Washington, U.S.A., 1995.

[21] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

[22] B. S. Heck, L. M. Wills, and G. J. Vachtsevanos. Software technology for implementing reusable, distributed control systems. *IEEE Control Systems Magazine*, 23(1):21–35, Feb 2003.

[23] R. Hill and D. Tilbury. Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In *Proc. Int. Workshop Discrete Event Systems (WODES06)*, pages 399–406, Ann Arbor, Michigan, U.S.A., Jul 2006.

[24] K. Hiraishi. On solvability of an agent-based control problem under dynamic environment. In *Proc. Int. Workshop Discrete Event Systems (WODES04)*, pages 91–96, Reims, France, Sep 2004.

[25] A. Jadbabaie, J. Lin, and A. S. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[26] A. Khoumsi. Coordination of components in a distributed discrete-event system. In *Proc. of the 4th International Symposium on Parallel and Distributed Computing*, pages 299–306, Jul 2005.

[27] S. Lafortune. On decentralized and distributed control of partially-observed discrete event systems. In L. Marconi C. Rossi C. Bonivento, A. Isidori, editor, *Advances*

in Control Theory and Applications, volume 353, pages 171–184. Springer Berlin / Heidelberg, 2007.

[28] R. J. Leduc, M. Lawford, and W. M. Wonham. Hierarchical interface-based supervisory control - parallel case. *IEEE Transactions on Automatic Control*, 50(9):1336–1348, 2005.

[29] Y. Li and W. M. Wonham. Controllability and observability in the state-feedback control of discrete-event systems. In *Proc. of the 27th Conferences on Decision and Control*, pages 203–208, Austin, Texas, U.S.A., Dec 1988.

[30] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[31] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44:173–198, 1988.

[32] F. Lin and W. M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions on Automatic Control*, 35(12):1330–1337, 1990.

[33] Z. Lin. *Coupled Dynamic Systems: From Structure Towards Stability and Stabilizability*. PhD thesis, ECE Department, University of Toronto, 2006.

[34] Z. Lin, M. Broucke, and B. A. Francis. Local control strategies for groups of mobile autonomous agents. *IEEE Transactions on Automatic Control*, 49(4):622–629, 2004.

[35] Z. Lin, B. A. Francis, and M. Maggiore. Necessary and sufficient graphical conditions for formation control of unicycles. *IEEE Transactions on Automatic Control*, 50(1):121–127, 2005.

[36] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[37] C. Ma and W. M. Wonham. *Nonblocking Supervisory Control of State Tree Structures*. LNCIS 317. Springer-Verlag, Berlin Heidelberg, 2005.

[38] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[39] A. Mannani, Y. Yang, and P. Gohari. Distributed extended finite-state machines: communication and control. In *Proc. Int. Workshop Discrete Event Systems (WODES06)*, pages 161–167, Ann Arbor, Michigan, U.S.A., Jul 2006.

[40] J. A. Marshall. *Coordinated Autonomy: Pursuit Formations of Multivehicle Systems*. PhD thesis, ECE Department, University of Toronto, 2005.

[41] J. P. Muller. Architectures and applications of intelligent agents: a survey. *The Knowledge Engineering Review*, 13(4):353–380, 1998.

[42] P. M. Pardalos and J. Xue. The maximum clique problem. *Global Optimization*, 4(3):301–328, Apr 1994.

[43] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[44] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of IEEE*, 77(1):81–98, Jan 1989.

[45] C. W. Reynolds. Flocks, herds, and schools: a distributed behavioral model. *Computer Graphics*, 21(4):25–34, Jul 1987.

[46] S. L. Ricker and K. Rudie. Incorporating communication and knowledge into decentralized discrete-event systems. In *Proc. of the 38th Conference on Decision and Control*, pages 1326–1332, Phoenix, Arizona, U.S.A., Dec 1999.

[47] K. Rudie, S. Lafortune, and F. Lin. Minimal communication in a distributed discrete-event system. *IEEE Transactions on Automatic Control*, 48(6):957–975, Jun 2003.

[48] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, 1992.

[49] K. Schmidt, T. Moor, and S. Park. A hierarchical architecture for nonblocking control of decentralized discrete event systems. In *Proc. 13th Mediterranean Conference on Control and Automation*, pages 902–907, Limassol, Cyprus, Jun 2005.

[50] D. D. Siljak. *Large-Scale Dynamic Systems: Stability and Structure*. North-Holland New York, 1978.

[51] H. A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, 106:467–482, 1962.

[52] H. A. Simon and A. Ando. Aggregation of variables in dynamic systems. *Econometrica*, 29:111–138, 1961.

[53] R. Su and W. M. Wonham. Supervisor reduction for discrete-event systems. *Discrete Event Dynamic Systems*, 14(1):31–53, Jan 2004.

[54] J. G. Thistle. Supervisory control of discrete-event systems. *Mathematical and Computer Modelling*, 23(11-12):25–53, 1996.

[55] S. Tripakis. Decentralized control of discrete-event systems with bounded or unbounded delay communication. *IEEE Transactions on Automatic Control*, 49(9):1489–1501, Sep 2004.

[56] A. F. Vaz and W. M. Wonham. On supervisor reduction in discrete-event systems. *International Journal of Control*, 44(2):475–491, Aug 1986.

[57] B. Wang. Top-Down Design for RW Supervisory Control Theory. Master's thesis, ECE Department, University of Toronto, 1995.

[58] Y. Willner and M. Heymann. Supervisory control of concurrent discrete-event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[59] K. C. Wong and J. H. van Schuppen. Decentralized supervisory control of discrete-event systems with communication. In *Proc. Int. Workshop Discrete Event Systems (WODES96)*, pages 284–289, Edinburgh, U.K., Aug 1996.

[60] K. C. Wong and W. M. Wonham. Hierarchical control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 6(3):241–273, 1996.

[61] K. C. Wong and W. M. Wonham. Modular control and coordination of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 8(3):247–297, 1998.

[62] W. M. Wonham. Design software: XPTCT. Systems Control Group, ECE Dept, University of Toronto, http://www.control.toronto.edu/DES, Version 119, Windows XP, updated July 1, 2007.

[63] W. M. Wonham. Supervisory control of discrete-event systems. Systems Control Group, ECE Dept, University of Toronto, http://www.control.toronto.edu/DES, updated July 1, 2008.

[64] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

[65] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems*, 1(1):13–30, 1988.

[66] T. S. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 12(3):335–377, 2002.

[67] H. Zhong and W. M. Wonham. On the consistency of hierarchical supervision in discrete-event systems. *IEEE Transactions on Automatic Control*, 35(10):1125–1134, Oct 1990.