INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

HIERARCHICAL SUPERVISORY CONTROL SYSTEMS

Paul Hubbard

Departement of Electrical and Computer Engineering
McGill University, Montréal

January 2000

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy

Résumé

Cette thèse présente deux approches pour la commande supervisée hiérarchique des systèmes à évènements discrets(DES), modélisés comme des automates déterministes finis.

On présente en premier, une théorie de commande hiérarchique, basée sur l'ensemble (totalité) des états, différente de l'approche classique qui utilise des méthodes de language formel.

Un modèle d'automate déterministe de haut niveau est caracterisé par son utilisation de notions de consistence dynamique(DC) dans sa définition des transitions de haut niveau. La condition *Trace-DC* est ainsi définie sur toute partition en éspace d'état qui garantie: (i) Les trajectoires dans les systèmes bas niveau sont représentées dans les systemes haut niveau. (ii) Les trajectoires dans les modèles haut niveau sont realisées utilisant les trajectoires dans les systèmes bas niveau. Il est aussi montré que la condition *Trace-DC* assure que les comportements controlables bas niveau sont représentés comme comportements controlables haut niveau.

La condition *Non-Blocking In-Block-controllability* sur les partitions est définie dans le but de garantir que les comportements, qui sont controlables dans les modèles haut niveau, peuvent être atteints par la commande des transitions haut niveau entre les blocs combinés avec les commandes locales à retour d'état dans chaque bloc. Un algorithme appelé *Vocalised lifting* est proposé pour la construction des partitions *Trace-DC*. Il est prouvé de la littérature des langages formels basés sur la théorie de la commande hierarchique supervisée, que la formulation de la supervision hiérarchique satisfait les conditions de consistence hiérarchique

Notre approche est illustrée avec plusieurs exemples de chaines de production manufacturières, incluant un tranfert ré-entrant et une chaine à deux queues séquentielles.

Une application pour la commande des appareils de maison (Machine à laver), motivant notre approche est présentée.

En deuxième volet de cette thèse, une formulation de la notion des systèmes d'automates déterministes interagis, basée sur le produit multi-agent(MA) est présentée. La relation de transition du produit MA est montrée qu'elle représente l'intersection des relations de transition de deux produit associés: le vecteur de produit synchrone à état dependant et le produit stimulant. Il est aussi prouvé que la langage accepeté par le produit MA est l'intersection des langages *embedded constraint* de chaque agent.

La supervision d'un agent par autre est ainsi considérée. Il est montré que la réalisation standard d'un automate fini peut être utlisée comme modèle pour l'agent superviseur.

Une direction de recherche basée sur la supervision centralisée, décentralisée et hiérarchique des systèmes produit MA est initiée. Il est montré que les comportements de produit MA, qui sont individuèllement et indépendement controlables avec le respect de chaque agent, forment un comportement controlable dans le système produit MA.

Abstract

This thesis presents two approaches to the hierarchical supervisory control of discrete event systems (DES) modelled as finite deterministic automata.

First, a hierarchical control theory based on the *aggregation of states* is presented that differs from the standard approach to hierarchical supervision which mainly employs formal language methods. A high-level (i.e. aggregated) deterministic automaton model is defined that uses notions of *dynamical consistency (DC)* in the definition of high-level transitions. A *Trace-DC* condition is then defined on any given state-space partition which ensures that (i) trajectories in the low-level system are always represented in the high-level system, and (ii) trajectories in the high-level model are realized by trajectories in the low-level system. It is also shown that the *Trace-DC* condition ensures low-level controllable behaviours (languages) are represented as high-level controllable behaviours.

The *(Non-Blocking) In-Block-Controllability* condition on partitions is then defined that ensures that behaviours that are expressly controllable in the high-level model can be achieved through the control of high-level transitions between blocks combined with local state - feedback controls in each block. A so-called *vocalised lifting* algorithm is proposed for the construction of Trace-DC partitions. It is shown that this formulation of hierarchical supervision satisfies conditions of hierarchical consistency from the literature on formal language-based hierarchical supervisory control theory.

The approach is illustrated with several examples of manufacturing production lines, including a re-entrant transfer line and a line with two sequential queues. An application to the embedded control of home appliances (washing machines) which motivates the approach is summarised.

Second, a notion of systems of interacting finite deterministic automata is formulated via a newly proposed *multi-agent (MA) product.* The transition relation of the MA product is shown to be the intersection of the transition relations of two associated products, the *vector state-dependent synchronous product* and the *simultaneous product.* It is shown also that the language accepted by the MA product is the intersection of the so-called *embedded constraint* language of each agent.

The supervision of one agent by another is then considered and it is shown that the standard finite automata realization for a supervisor can be employed as the model for the supervising agent.

A line of research is then initiated on the centralised, decentralised and hierarchical supervision of MA product systems. It is shown that the MA product of behaviours that are individually and independently controllable with respect to each agent forms a controllable behaviour in the MA product system.

Acknowledgements

First and foremost, I am indebted to my thesis supervisor, Professor Peter E. Caines for his technical advice and moral support. He is also the co-author of the papers on which this thesis is based. I would like to thank the other members of the systems and control research group, with whom I shared the process of research and discovery, for their friendship and support. These are Carlos Martinez-Martínez-Mascarúa, Thomas Mackling, Ekaterina Lemch, Shen Gang, Michael Glaum, Charalambous Charalambos and Benoit Boulet. The staff at the Centre for Intelligent Machines (CIM), Kathleen VanderNoot, Ornella Cavaliere and Marlene Gray, were an endless source of encouragement and administrative assistance. I must add my thanks to Jan Binder and others involved in the CIM computer systems administration which was invariably excellent.

I also greatly appreciate the time and effort put in by Dr. Tim Johnson and Vivek Badami at General Electric R& D.

The sources of financial support that made this research possible are the National Science and Engineering Research Council (NSERC), General Electric R&D and the Bank of Montreal. I gratefully acknowledge their support.

Finally, thanks to my parents for their love and gracious hospitality during the final stages of this thesis and thanks to my wife, Ann, for everything. Words do no justice.

Paul Hubbard

Montréal, Québec, August 1999.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

An essential element in the design of a complex system, whether it be a manufacturing plant, a communications network or an enterprise management system, is a high level description of component functionality, logical dependence and information flow. A common form for such a description is a graphical representation such as a flow chart or a finite state machine. Independent of the complexity of the real system, there is an informal maximum complexity (measured perhaps, by the size or number of elements in the flow chart) for such a description to be a meaningful tool for reasoning about the design. Hence there is a need for a formal tie between an abstract description and the true system so that reasoning at the abstract level can be effectively translated to the true system.

In this thesis, two forms of abstract model are proposed in the setting of a specific formalism; that of supervisory control of discrete-event systems modelled as finite deterministic automata. The first is based on the aggregation of states to form an abstract model in which the abstract states now correspond to sets of states in the true system. The second is based on a model for the concurrent evolution of multiple agents. The product of the agent state spaces gives the true system state and the dynamics are provided by a specific definition for the interaction of the agents.

Complexity

The aspects of complexity that are at issue are organisational and constitutional complexity ([**79**]) rather than computational complexity. That is, the cause of the problems at the design stage is not the length of time of calculations, but rather the need for an organised approach to combining subsystems that have extensive

interactions. Computational complexity, on the other hand, may be of importance if an automated tool is used to aid the design and becomes much more important during implementation.

It should be noted that the approach here is not that of disguising complexity as uncertainty as is done in robust control or fuzzy logic. Rather, it is assumed that exact models (deterministic or stochastic) are in principle available at any level of refinement and the complexity must be organised to form a meaningful abstraction for design.

The most natural response to the issue of organisational complexity is hierarchical organisation. It has been argued that hierarchies are universally present in natural and synthetic complex systems [**84, 45**]. Characteristics of multi-level hierarchical structures, their vertical arrangement, and subsystem prioritisation have been defined in various contexts [**65, 90**], and more recently in settings similar to this thesis in [**44, 72, 13, 99, 46**]

Discrete Event Systems

Historically, discrete state systems and the complexity they engender have been studied, at least indirectly, in many fields including game theory [**98**], operations research [**24**], graph theory [**36**], discrete mathematics [**11**] and digital design [**63**].

Motivated by the almost universal use of computation-based control and the spread of systems-based philosophies to application areas with complex discrete dynamics, the field of Discrete Event Dynamical Systems (DEDS or sometimes simply DES, see [**4**]) has emerged as the application of systems and control concepts to formal computer science. Because of this dual heritage, there is a dichotomy in perspective regarding DES. The systems-theoretic view of this field is that DESs are most often derived from a continuous state space, e.g. by cell-wise decomposition. As a result, many system-theoretic notions (for instance controllability) gain new interpretations in the DES field in terms of reachability or connectedness [**36, 19**]. The formal computer science view, on the other hand, is motivated by issues such as decidability [**39**] and formal verification [**70**] which regard DESs as purely computational processes.

The dichotomy is also present in the closely related field of hybrid systems (see for instance,[**18, 26, 75, 50**] and [**7, 8**]).

Other fields have emerged with similar characteristics and motivations to DES. These include intelligent systems and expert systems [**3, 86**], multi-agent systems [**32**], decision systems [**74**], etc. All attempt in some way to make systems of enormous complexity amenable to human understanding and (re-)organisation, either by abstraction or by creating tools that allow for formal verification.

Supervisory Control Theory

Within the DES field, there are many approaches to system modelling. These include Markov chains, automata and finite state machines, Max+ Algebra and Petri Nets (see [**22**] and references therein, and for a short comparison of these models for a specific queueing example see [**21**]). The contents and contributions of this thesis lie in the automata-theoretic and finite state machine setting, specifically in the supervisory control framework.

In the supervisory control framework, DESs are modelled as generators of a formal language. These are untimed logical models (see [**78**] and references therein, in particular the original articles [**77**] and [**101**] and the longer works [**91, 47**]). Feedback control is applied through the observation of system events and the inhibition of future events. System behaviours are also described by languages (i.e. sets of strings of events) and the theory seeks to determine, among other things, which behaviours can be achieved through the inhibition of future events by an external supervisor.

There is an extensive literature for supervisory control. A partial list of topics includes observability [**71, 60**], control based on partial observation [**25, 47**], modular control [**76, 102**], decentralised control [**80, 58, 59**], nondeterminism [**38, 62**], timed aspects [**12**], infinite strings [**91**], vector systems [**55, 56**], concurrency [**95, 57**] and supervisory control of Petri Nets [**35, 85**].

Hierarchical Strategies in Supervisory Control

Complexity issues arise in the supervisory framework due to combinatorial explosion in the number of states (for computational complexity considerations see [**94**] and for complex examples see [**81, 82**]). Hierarchical supervisory control formulations in a linguistic framework have appeared in [**103**] and [**99**] and an approach based on supervisor reduction appeared in [**96**].

Two approaches to hierarchical supervision are presented in this thesis. The first approach, presented in Chapter 2, is to clump states into blocks in order to

form an abstract supervisory model. This is motivated by hierarchies based on state aggregation that appeared in [**19, 15, 83**] and [**33**] and by the expected generalisation to hybrid systems [**17, 52**]. The main results for the chapter are Theorems 4.1 and 4.2 in which properties of consistency between the low level system and the abstraction are proved. Where possible, the work is compared with the hierarchical supervisory control theory in [**103, 99**].

The second approach, presented in Chapter 4, is a modular approach based on interacting agents. This is motivated partially by the general work in [**67**], the multi-agent formalisms from computer science in [**32**], and the concurrency ideas in [**68**]. The main results for the chapter are the development of an algebraic description of the language accepted by a so-called multi-agent (MA) product system and an initial analysis of the supervisory control problem for multi-agent systems.

DES Applications Areas

Chapter 3 of this thesis contains several worked examples and a discussion of a motivating application areas. The intention is to demonstrate the theory and prove the worth of the analysis for the design of much larger systems. The emphasis in the worked examples is on manufacturing systems and discrete-event production lines [**28**] but the analysis might well be exported to applications such as communication network architecture designs and resource allocation.

Manufacturing Systems

The fields of Operations Research and Expert Systems, among others, focus on the control and optimisation of manufacturing models. In particular, [**90**] and the other papers in the same collection discuss coordination and conflict in the manufacturing systems setting and [**64**] contains a survey of expert systems and manufacturing techniques.

Within the DES field, applications in manufacturing systems are also common. In particular, [**34**] discusses a probabilistic model in which the relative frequency of events yields a hierarchical structure, [**103, 29, 49**] contains an analysis of re-entrant flow production lines in the supervisory control setting and [**10**] provides a logical synthesis from a flow chart through automata to integrated circuit layouts. The supervision of batch processes modelled with Petri Nets appeared in [**92**] and supervisory control in the context of programmable logic controllers was examined in [**31**].

Embedded Systems

With the advent of automated integrated circuit design, embedded controllers are now pervasive in many industries, for instance in the home appliances industry, the automotive industry and the telecommunications industry. A pertinent discussion of the design of embedded systems and the need for modular design based on pre-verified components can be found in [**30**].

The candidate performed a brief part of the research for this thesis at General Electric R & D in Schenectady, NY on the control of washing machines with an embedded system (see[**9**]). A discussion of this exploratory phase can be found in Chapter 3.

Other Applications Areas and DES software

Other application areas for DES include failure diagnosis (see [**81, 82**] and [**97**]), communication systems (for a discussion of Internet topology see [**20**] and feature interaction in telephony see [**23**]). Larger distributed applications include automotive traffic systems (see [**66**] for a multi-origin multi-destination network model) and air traffic control [**73**], [**93**].

Many software applications packages exist for DESs. These include packages developed in the academic setting such as [**5, 6**] and a host of commercially available software such as Matlab Stateflow [**2**]. These offer varying degrees of generality or simulation environments and hence inversely varying degrees of formal verification tools.

CHAPTER 2

State Aggregation and Hierarchical Supervisory Control

1. Introduction

The supervisory control framework for modelling discrete event systems (DES) is an untimed logical model that is expressed in terms of the observation and inhibition of events. Within this framework, system behaviours are described by languages (i.e. sets of strings of events) and the theory seeks to determine which behaviours can be achieved via the inhibition of a subset of the system's events (see [**78, 77, 101**] and [**91, 47**]).

Because of the generally accepted role of hierarchies in system complexity reduction, the immediate complexity issues that arise for DESs due to combinatorial explosion (for examples, see [**23, 82**]) motivate a hierarchical approach to supervisory control.

In this chapter, which follows closely the work in [**16, 40, 41, 43**], a hierarchical control theory based upon state aggregation is presented. The general idea is illustrated Figure 2.1. An abstraction of the system model is created by partitioning the system state-space and forming an abstract model in which the abstract states now correspond to sets of states in the true system.

Definitions of relevant conditions for partitions and high-level dynamics for a hierarchical supervisory control automaton are then given. A motivating problem for the entire formulation is that of ensuring (non-blocking) accessibility of the designated states; it is shown that this problem may be decomposed and solved in a hierarchical

fashion with local state-feedback supervision in conjunction with supervision at the aggregated level.

Other work on a hierarchical theory for supervisory control has appeared in the DES literature. Within a linguistic framework, [**103**] and [**99**] develop notions of *output control consistency* and *hierarchical consistency* which capture conditions for effective high-level control. Links and comparisons will be made with these notions in the current work. An aggregated modular approach to controller reduction based on covers of the state set of the supervisor also appeared in [**96**]. In [**72**], the authors consider an approach in which high-level events represent specific sequences of low-level events ("tasks") which may need to be tracked or repeated numerous times. An aggregation-based theory appeared in [**13**] in the context of statechart models. An analysis of Petri Nets based on refinement and abstraction appeared in [**88**], with a corresponding analysis of policies that enforce liveness being given in [**85**]. A discussion of state-space partitions can be found in [**89**] and a condition is presented which guarantees, for a given specification, the existence of a unique maximal partition on which to base control.

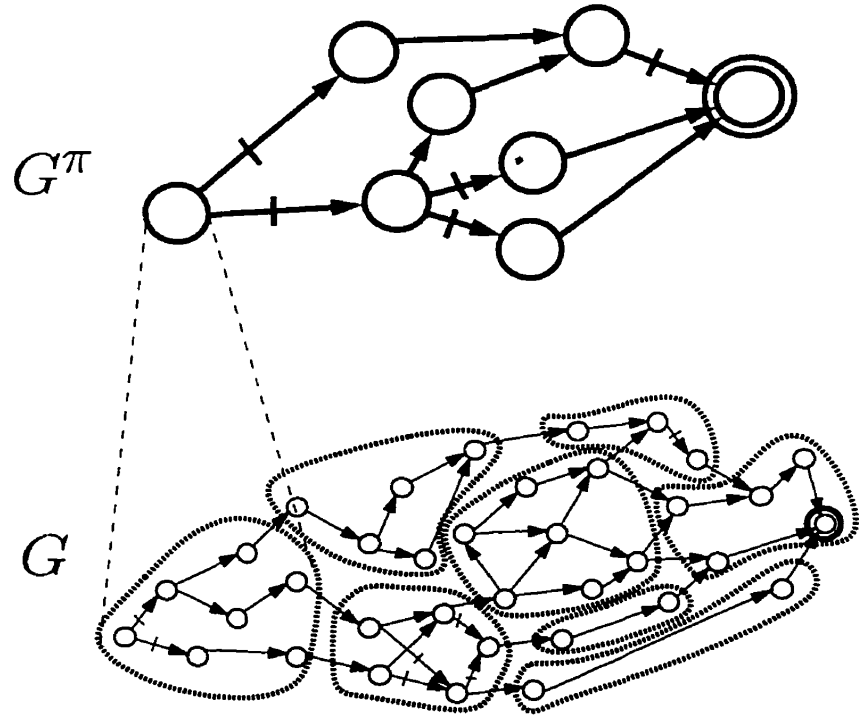


FIGURE 2.1. Aggregation and the π-partition automaton.

The distinct characteristic of the present work is that the state aggregation is based upon *Dynamical Consistency* (DC). This is a notion developed in the context of forced event (positive-imperative) control in [**19, 15**], generalised to differential control systems in [**17, 18, 50, 52, 51**] and which, further, is related to notions in a purely graph theoretic setting (see [**36**]) and in computer science (see [**7, 37**]).

State-feedback control in the supervisory framework has been considered previously in [**54, 48**].

Relevant definitions from the literature concerning languages, supervision, controllability, non-blocking, etc., are given concurrently with the development of the state aggregation theory. The following chapter contains several worked illustrative examples.

2. State Aggregation in the System Model

A supervisory automaton G is a five-tuple,

$$G = (X, \Sigma_u \cup \Sigma_c, \delta, Q_0, Q_m), \tag{2.2.1}$$

where X is a set of states, $\Sigma = \Sigma_u \cup \Sigma_c$ is an alphabet of event labels (where $\Sigma_u \cap \Sigma_c = \emptyset$ by assumption) and $\delta : X \times \Sigma \longrightarrow X$ is a (partially-defined) transition function. All (observed) admissible initial states q_0 lie in Q_0, and Q_m is the set of marked goal states.

The notion of a *set* of initial start states, Q_0, as defined in (2.2.1) differs from the standard single start state defined in [**77**]. This is purely notational and is motivated by a desire to maintain a self-similar layering in a hierarchy of abstractions in which multiple start states must be considered. It is assumed the initial state $q_0 \in Q_0$ is identified *a priori* for any execution of the model. At present only completely state-observable systems are considered. It is also assumed that all states in X are reachable from some initial state and co-reachable from a state in Q_m. The alphabet of event labels is composed of a set of controllable labels Σ_c which may be disabled by an external control (a supervisor) and a set of uncontrollable labels Σ_u which may not be disabled.

An abstraction of the system can be constructed via a partition π of the state set. Accordingly, let $\pi = \{X_1, X_2, .., .X_N\}$ with $\bigcup X_i = X$, $X_i \neq \emptyset$, $1 \leq i \leq |\pi|$, and $X_i \cap X_j = \emptyset$ for $i \neq j$.

Motivated by the analysis in [**15**] regarding the dynamics between partition blocks, the following definitions are made.

Definition 2.1. $I(X_i, Q_0)$

The *In-set* $I(X_i, Q_0)$ of a partition block X_i is the set of states in X_i that are either in the initial state set or directly accessible (i.e. one step accessible) from the complement of the block, i.e.

$$x \in I(X_i, Q_0) \iff [\, x \in Q_0 \cap X_i \quad \vee \\ \exists x' \in X_i^C . \exists u \in \Sigma . \delta(x', u) = x].$$

□

Definition 2.2. $\langle X_i, X_j \rangle_u$

The relation $\langle X_i, X_j \rangle_u$ holds for $i \neq j$ whenever, for each state in the in-set x of X_i, there exists an uncontrollable path with initial state x, intermediate states uniquely in X_i and terminal state in the in-set of X_j, i.e.

$$\langle X_i, X_j \rangle_u \Longleftrightarrow \forall x \in I(X_i, Q_0) . \exists \sigma \in \Sigma_u^* . \\ \delta(x, \sigma) \in X_j \wedge \forall \sigma' < \sigma, \delta(x, \sigma') \in X_i .$$

where $<$ is the prefix partial order, i.e. $s' < s$ if s' is a strict prefix of s. □

Definition 2.3. $\langle X_i, X_j \rangle_d$

The relation $\langle X_i, X_j \rangle_d$ holds for $i \neq j$ whenever, for each state x in the in-set of X_i, there exists a path with initial state x, intermediate states uniquely in X_i and terminal state in the in-set of X_j and, furthermore, all such paths contain a controllable transition, i.e.

$$\langle X_i, X_j \rangle_d \iff \forall x \in I(X_i, Q_0) . \exists \sigma \in \Sigma^* . \\ \delta(x, \sigma) \in X_j \wedge \forall \sigma' < \sigma . \delta(x, \sigma') \in X_i \\ \wedge \forall x \in I(X_i, Q_0) . \nexists \sigma \in \Sigma_u^* . \\ \delta(x, \sigma) \in X_j \wedge \forall \sigma' < \sigma . \delta(x, \sigma') \in X_i .$$

□

Note that $\langle X_i, X_j \rangle_d$ and $\langle X_i, X_j \rangle_u$ cannot hold simultaneously. The abstract model, which is itself a supervisory automaton, may now be formally defined.

Definition 2.4. π-Partition Automaton

The *π-partition automaton* is defined as,

$$G^\pi \triangleq (\pi, \Sigma_u^\pi \cup \Sigma_c^\pi, \delta^\pi, Q_0^\pi, Q_m^\pi)$$

where $\pi = \{X_1, ...X_N\}$ is the set of states. When $\langle X_i, X_j \rangle_d$ holds between two blocks, a π-level disableable transition U_i^j is defined. Similarly, when $\langle X_i, X_j \rangle_u$ holds, an undisableable π-level transition V_i^j is defined. The disjoint sets Σ_u^π and Σ_c^π are the collections, respectively, of the defined symbols U_i^j and V_i^j.

The (partially-defined) *π-level transition function*, $\delta^\pi : \pi \times \Sigma^\pi \to \pi$, is defined such that when a transition exists, it forms a directed edge between the associated states, i.e. $\langle X_i, X_j \rangle_d \Longrightarrow \delta^\pi(X_i, U_i^j) = X_j$, and $\langle X_i, X_j \rangle_u \Longrightarrow \delta^\pi(X_i, V_i^j) = X_j$. The set of *$\pi$-level initial states* is $Q_0^\pi \stackrel{\text{def}}{=} \{X_i \in \pi | X_i \cap Q_0 \neq \emptyset\}$. The set of *$\pi$-level goal states* is $Q_m^\pi \stackrel{\text{def}}{=} \{X_i \in \pi | X_i \cap Q_m \neq \emptyset\}$. □

The selection of a high-level representation which is itself a supervisory automaton (i.e. a model in which control is applied through inhibition) is again motivated by a desire for self-similarity between layers in the hierarchy. In practice, this allows for a single set of algorithms to be used, independently of hierarchical layer, for tasks such as building partitions, optimisation (for instance maximal controllable subset calculations) and instantiating control actions. There are clearly many other candidates for the higher-level descriptions. One possibility is a forced-event model. The hierarchical theory developed in [**19**] then applies directly (except at the bottom layer, which remains a supervisory automaton). Another possibility is a continuous description rather than a granular model [**50**].

Definition 2.5. *Trace* Dynamical Consistency (*Trace*-DC)

A π-partition automaton G^π is *Trace-DC* iff

$$\forall\, i \neq j, 1 \leq i, j \leq |\pi|.$$
$$\{\langle X_i, X_j \rangle_u \ \vee \ \langle X_i, X_j \rangle_d \ \vee \ \neg[\exists w \in \Sigma. \exists x \in X_i. \exists x' \in X_j.\ \delta(x, w) = x']\}.$$

□

The *Trace-DC* definition ensures that π-level transitions are realized by trajectories in the low-level automaton and also that low-level trajectories are always represented in the π-partition automaton. It must be stressed that the *Trace-DC* property

of a π-partition automaton depends upon the properties of the partition π and the low-level automaton.

A condition appeared in a similar context in [**7, 68**] termed *bisimulation.* The *Trace*-DC condition for a partition automaton is weaker than the more strict bisimulation equivalence condition for two reasons. In the latter, a high-level transition would represent a single-transition connection from all states in one partition block to a state in another and further, all such transitions from a given block to another would be required to have identical labels. In the *Trace*-DC setting, longer strings of transitions are permitted and these need not have the same event labels.

The unmarked language, i.e. the set of strings of event labels accepted by the automaton G but not necessarily ending at the goal set, is denoted as $L(G)$. The marked language is denoted $L_m(G)$. Languages may be defined on the high-level alphabet Σ^π in a similar fashion, i.e. $L(G^\pi), L_m(G^\pi)$ and when necessary, a superscript, e.g. $K^\pi \subseteq L(G^\pi)$ will be included to emphasise the layer at which the language is defined.

2.1. Maps from Low-Level Systems to High-Level Systems

The canonical map (for states), $\Theta_\pi : X \longrightarrow \pi$, is defined as

$$\Theta_\pi(x) = X_i \text{ if } x \in X_i, \qquad 1 \leq i \leq |\pi|,$$

and extended in the natural way to domains of successive greater complexity:

(state-sets) $\Theta_\pi : 2^X \longrightarrow 2^\pi$,

$$\Theta_\pi(R) = \{X \in \pi | X \cap R \neq \emptyset\}, \quad R \subseteq X$$

(strings of event labels) $\Theta_\pi : L(G) \longrightarrow L(G^\pi)$,

$$\Theta_\pi(\epsilon) = \epsilon,$$

$$\Theta_\pi(\sigma w) = \begin{cases} \Theta_\pi(\sigma)U_i^j & \text{if } \delta(x_0,\sigma) \in X_i,\ \delta(x,\sigma w) \in X_j \text{ and } \langle X_i, X_j\rangle_d \\ & \text{for some } 1 \leq i,j \leq |\pi|, \\ \Theta_\pi(\sigma)V_i^j & \text{if } \delta(x_0,\sigma) \in X_i,\ \delta(x,\sigma w) \in X_j \text{ and } \langle X_i, X_j\rangle_u \\ & \text{for some } 1 \leq i,j \leq |\pi|, \\ \Theta_\pi(\sigma) & \text{if } \delta(x_0,\sigma) \in X_i \text{ and } \delta(x,\sigma w) \in X_i \text{ for some } 1 \leq i \leq |\pi|, \end{cases}$$

where $\sigma \in \Sigma^*$ and $w \in \Sigma$.

Note that $\Theta_\pi : L(G) \longrightarrow L(G^\pi)$ is well-defined only when G^π is *Trace-DC.*

(languages) $\Theta_\pi : 2^{L(G)} \longrightarrow 2^{L(G^\pi)}$,

$$\Theta_\pi(K) = \{\Theta_\pi(\sigma) | \sigma \in K\} \subseteq \Sigma^{\pi^*},$$

where $2^{L(G)}$ denotes the set of sub-languages of $L(G)$.

(Mealy machine output) $\Theta_\pi : X \times \Sigma \longrightarrow \Sigma^\pi \cup \{\epsilon\}$,

$$\Theta_\pi(x,w) = \begin{cases} U_i^j & \text{if } x \in X_i,\ \delta(x,w) \in X_j, \text{and } \langle X_i, X_j \rangle_d, \\ V_i^j & \text{if } x \in X_i,\ \delta(x,w) \in X_j, \text{and } \langle X_i, X_j \rangle_u, \\ \epsilon & \text{otherwise} \end{cases} \tag{2.2.2}$$

Definition 2.6. ([39] for instance) **Mealy Machine**

A *Mealy machine* is a six-tuple $(X, \Sigma, \Delta, \delta, \Theta, q_0)$ where X,Σ and δ are as in the supervisory automaton (2.2.1), q_0 is the initial state, Δ is an output alphabet, and the map $\Theta : X \times \Sigma \longrightarrow \Delta$ gives an output $\Theta(x,w)$ associated with the transition from state x on input w. □

The map Θ_π in (2.2.2) can be combined with the low-level system G to form a Mealy machine, G_{Θ_π}, via

$$G_{\Theta_\pi} = (X, \Sigma, \Sigma^\pi, \delta, \Theta_\pi, Q_0). \tag{2.2.3}$$

The interpretation of the output function in (2.2.3) is that symbols from Σ^π vocalise the crossing of partition boundaries. Again, a similar caveat applies to the notation of a set of initial states Q_0; it is assumed that a start state q_0 is identified for each execution of the system.

Note that, as an observation map, Θ_π is both formally and operationally different from the standard observation *mask* or natural projection that is used most frequently in the literature on supervisory control under partial observation ([**47, 25**]). As an observer, the effect of Θ_π in the present setting is to restrict the estimate of the current state to an element of the partition π.

2.2. Mealy and Moore Representations

In the next section a comparison will be made between the *Trace-DC* condition and the *output control consistent* condition from [**103**]. The *output control consistent* is formally defined for Moore machines hence Moore machines will now be formally defined and their relationship with Mealy machines summarised.

Definition 2.7. ([39] for instance) **Moore Machine**

A *Moore machine* is a six-tuple $(X, \Sigma, \Delta, \delta, \tilde{\Theta}, q_0)$ where X,Σ and δ are as in the supervisory automaton (2.2.1), q_0 is the initial state, Δ is an output alphabet, and the map $\tilde{\Theta} : X \longrightarrow \Delta$ gives an output associated with each state. □

An equivalent Moore machine representation can be constructed from a Mealy representation, in general, by embedding the states X in a new state-set $\tilde{X} = X \times \Sigma^\pi$ (see [**39**] p. 44 for instance).

In the current case, G_{Θ_π} yields the equivalent Moore representation.,

$$\tilde{G}_{\Theta_\pi} = (X \times \Sigma^\pi, \Sigma, \Sigma^\pi, \tilde{\delta}, \tilde{\Theta}_\pi, [q_0, A_0]) \tag{2.2.4}$$

where,

$$\tilde{\delta}([x, A], a) = [\delta(x, a), \Theta_\pi(x, a)] \in X \times \Sigma^\pi,$$
$$\tilde{\Theta}_\pi([x, A]) = A,$$

for $x \in X$, $A \in \Sigma^\pi$, $a \in \Sigma$. A_0 in (2.2.4) is an arbitrary member of Σ^π.

Example 2.1. *An example of the Mealy to Moore construction is given in 2.2. In the figure, G_{Θ_π} is a partition machine in the Mealy format, i.e. where the observation map, $\Theta_\pi : X \times \Sigma \longrightarrow \Sigma^\pi$, reports a subset of the low-level event set. $\tilde{G}_{\Theta_\pi}$ shows an equivalent Moore representation where the observation map is now $\tilde{\Theta}_\pi : \tilde{X} \longrightarrow \Sigma^\pi$. Note that, as stated in the text, only states x and y need be split because these are in-set states which have incoming transitions from multiple blocks. The two representations are equivalent in the sense that all accepted strings give the same string of observations. Note that in Figure 2.2, the ticked arrows are used to represent controllable transitions. Unticked arrows represent uncontrollable transitions.* □

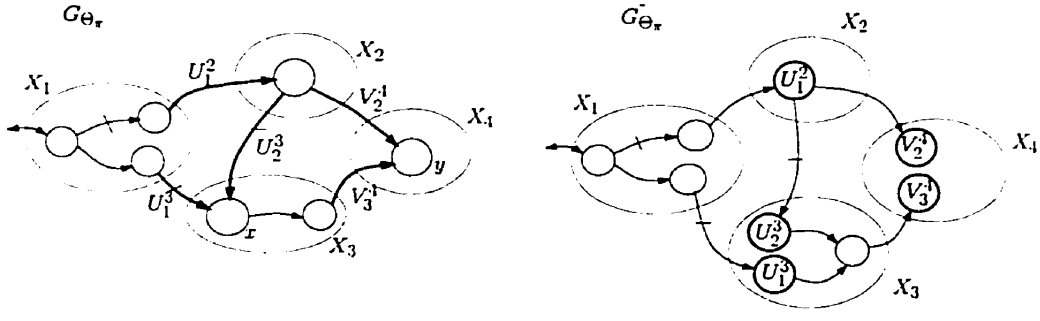


FIGURE 2.2. An example of the Mealy to Moore translation.

The interpretation of the Mealy to Moore translation is that the information about the current transition event is carried in the state of the Moore machine.

In practice, for the specific case of G_{Θ_π} created via (2.2.2) and (2.2.3), the construction of the equivalent Moore representation only requires extending the state space at specific states, i.e. those states which (a) are in-sets and (b) have arriving transitions from two or more different blocks $X_{i_1}, ..., X_{i_n}$. Such a state $x \in X_{i_0}$ is then replaced with the states $[x, i_0]...[x, i_n]$, and the these are mapped to the output symbols $U_{i_0}^{i_1}, ...U_{i_0}^{i_n}$ respectively. The remaining states need only be replaced by one state $[x', A_0]$, where A_0 may be chosen arbitrarily and these states are mapped to the null output symbol ϵ.

The intention in the construction of the Mealy and Moore machines in the context of supervisory control is that the events in Σ retain their controllability properties (i.e. that events in Σ_c can be inhibited). The issue in both settings is whether there is a meaningful setting for supervision of the events in Σ^π through the inhibition of events in Σ.

2.3. Comparison Between *Trace DC* and *Output Control Consistency*

A Moore automaton is said to be *output control consistent* ([**103**]) when the high-level alphabet can be divided unambiguously into controllable transitions (inhibitable) and uncontrollable transitions. A formal definition is as follows.

Definition 2.8. ([103]) Output Control Consistency

A Moore machine $G = (X, \Sigma = \Sigma_u \cup \Sigma_c, T = \{\tau_0\} \cup T_u \cup T_c, \delta, \Theta, x_0)$, where T, T_u, T_c are alphabets and τ_0 is an output symbol, is *output control consistent* if for every string $s \in L(G)$ of the form

$$\begin{aligned} &\text{(A)} \quad s = \sigma_1\sigma_2\cdots\sigma_k \qquad\qquad \text{or, respectively,} \\ &\text{(B)} \quad s = s'\sigma_1\sigma_2\cdots\sigma_k, \end{aligned}$$

where $s \in \Sigma^+$ and $\sigma_i \in \Sigma$, and,

$$\begin{aligned} \text{(A)} \quad &\Theta(\delta(x_0, \sigma_1\sigma_2\cdots\sigma_i)) = \tau_0, \text{ for } 1 \leq i \leq k-1, \text{ and} \\ &\Theta(\delta(x_0, s)) = \tau \neq \tau_0 \end{aligned}$$

or, respectively ,

$$\begin{aligned} \text{(B)} \quad &\Theta(\delta(x_0, s')) \neq \tau_0, \text{ and} \\ &\Theta(\delta(x_0, s'\sigma_1\sigma_2\cdots\sigma_i)), \text{ for } 1 \leq i \leq k-1, \text{ and} \\ &\Theta(\delta(x_0, s)) = \tau \neq \tau_0, \end{aligned}$$

it is the case that,

$$\text{if } \tau \in T_c, \text{ then for some } i, 1 \leq i \leq k, \sigma_i \in \Sigma_c, \text{ and}$$
$$\text{if } \tau \in T_u, \text{ then for all } i, 1 \leq i \leq k, \sigma_i \in \Sigma_u.$$

□

The conditions of *Trace-DC* for G^π and *output control consistency* for the associated $\tilde{G}_{\Theta_\pi}$ are incomparable. *Trace-DC* includes a universal quantification over the in-set of a given block which is not needed for *output control consistency*. *Output control consistency* requires that high-level uncontrollable transitions never be instantiated by controllable paths, which is allowable in the definition of $\langle\rangle_u$. Examples are shown in Figure 2.3 in which G_1 is *Trace-DC* but not *Output Control Consistent*, while G_2 is the opposite. The figures show the Moore representations, and the output alphabets are $T = \Sigma^\pi$ with $T_c = \Sigma_c^\pi$ and $T_u = \Sigma_u^\pi$.

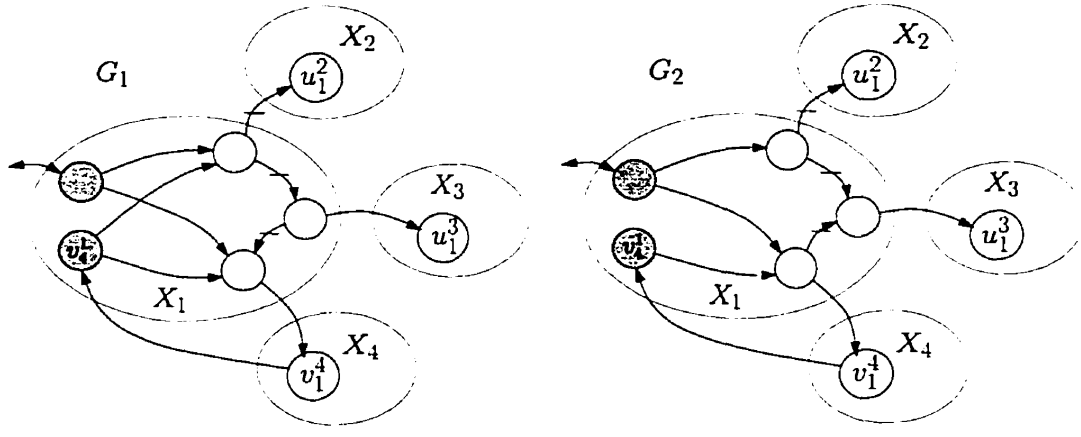


FIGURE 2.3. *Trace-DC* and *output control consistency* are incomparable (in-sets are shadowed).

The definition $\langle\rangle_u$ for uncontrollable π-level transitions can be strengthened to $\langle\rangle_{su}$ (strongly uncontrollable) with the additional requirement that there be no paths from the in-set to the next block which contain any controllable events. With this strengthening of $\langle\rangle_u$ to $\langle\rangle_{su}$, it may be shown that if G^π is *Trace-DC* then $\tilde{G}_{\Theta_\pi}$ is *output control consistent.*

It is significant that, as is shown in Section 4, the *Trace-DC* and *output control consistent* conditions still achieve the same property of preserving language controllability under mapping from low-level to high-level image.

3. Control of the Abstract Model (the π-Automaton)

Let a *supervisor* for the automaton G with start state q_0 in Q_0 be a function, $f \equiv f_{q_0} : \Sigma^* \longrightarrow 2^{\Sigma}$ specifying a set of enabled (i.e. not controller disabled) transitions which immediately follow any given event sequence σ. It is assumed henceforth that for all σ, $\Sigma_u \subseteq f(\sigma) \subseteq \Sigma$. After σ has occurred, the controlled automaton (denoted (G, f) may evolve with any transition w satisfying both $\delta(\delta(x_0, \sigma), w)!$ and $w \in f(\sigma)$.

At the π level, supervisors are defined in a similar fashion to that at the low level, i.e. $F^{\pi} : \Sigma^{\pi *} \longrightarrow 2^{\Sigma^{\pi}}$. Due to the fact that π-automata possess unique event labels (the labels U_i^j being indexed by the preceding and following blocks X_i and X_j), a starting state-set $|Q_0^{\pi}| \geq 1$, does not lead to nondeterministic behaviour at the π level.

Herein, only *state-feedback* supervision within each local block will be considered, i.e. at each block X_i, a supervisor $f_i : X_i \longrightarrow 2^{\Sigma}$ specifies the enabled transitions at the states within X_i.

Example 3.1. *Figure 2.4 illustrates two low-level automata and two partition automata. It can be checked that G_1^{π} is* Trace-DC. *There are four control inputs possible at X_1 in G_1^{π} and these can be enacted in G_1 by the low-level state-feedback controls $f_1 : x_3 \mapsto \Sigma_u, f_1 : x_5 \mapsto \Sigma_u$ which enables neither U_1^2 nor U_1^3 in G_1^{π} $f_2 : x_3 \mapsto \Sigma_u \cup \{u_1\}, f_2 : x_5 \mapsto \Sigma_u$ which enables U_1^2, $f_3 : x_3 \mapsto \Sigma_u, f_3 : x_5 \mapsto \Sigma_u \cup \{u_3\}$ which enables U_1^3, and $f_4 : x_3 \mapsto \Sigma_u \cup \{u_1, u_2\}, f_4 : x_5 \mapsto \Sigma_u \cup \{u_3\}$ which enables both U_1^2 and U_1^3 . (it is assumed $f_k(x_j) = \Sigma_u$ for $1 \leq k \leq 4, j = 1, 2, 4$).*

In the second example, on the right of Figure 2.4, it is also the case that G_2^{π} is Trace-DC, *but now there is no low-level control which disables only U_1^3 at X_1 in G_2^{π}.* □

A local automaton is now defined for each block X_i that is parameterised by a start state $x \in I(X_i, Q_0)$ and a set of goal states Q.

Definition 3.1. $G_{X_i^+}(x, Q)$

The *sub-automaton* $G_{X_i^+}(x, Q)$ of the automaton $G = (X, \Sigma, \delta, Q_0, Q_m)$ is defined as

$$G_{X_i^+}(x, Q) = (X_i^+, \Sigma, \delta^{X_i^+}, x, Q)$$

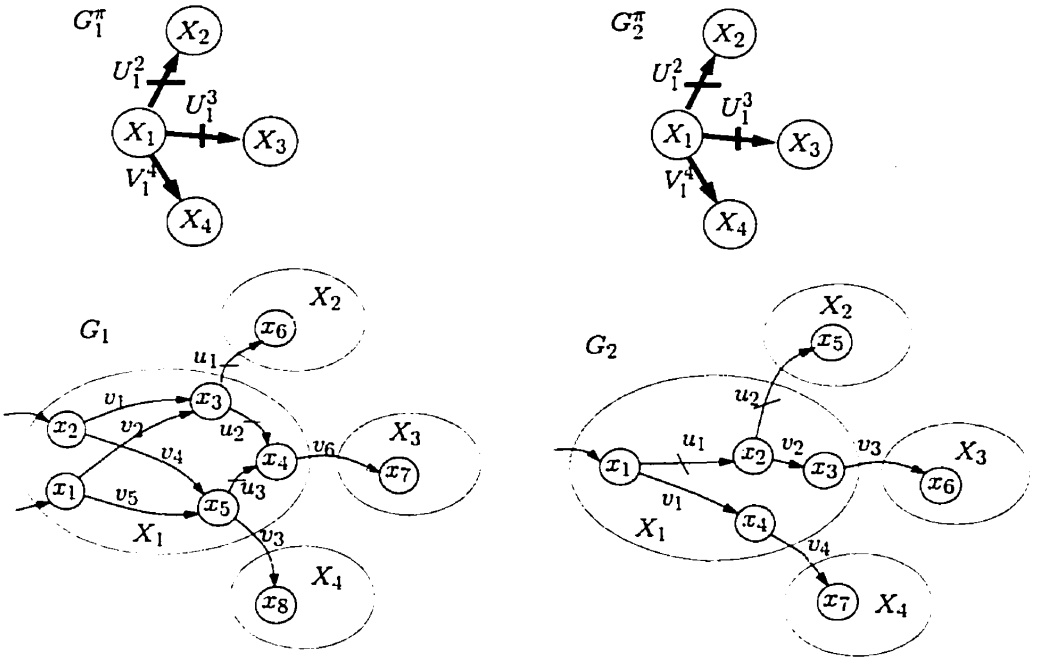


FIGURE 2.4. Examples of *Trace-DC* partition automata.

where $X_i^+ = X_i \cup \{x' \in X_i^c | \exists w \in \Sigma . \exists y \in X_i . \delta(y, w) = x'\}$, i.e. the block X_i and the states reachable from X_i in one transition. $\delta^{X_i^+}$ is δ with domain restricted to the set $X_i \times \Sigma$ and x and Q are parameters (a state and state-set, respectively). □

The controllability notions for state-feedback supervision and non-blocking accessibility (of target states) may be formulated with the following definition (adapted from [54], and appearing equivalently with the notion of predicates in [48]).

Definition 3.2. [54] (Non-Blocking) Controllable State-Sets

A set $R \subseteq X$ is *controllable* with respect to the automaton $G = (X, \Sigma_u \cup \Sigma_c, \delta, x_0, Q_m)$ if the following hold:

1. $\forall x \in R . \exists \sigma \in \Sigma^* .\ \delta(x_0, \sigma) = x \wedge \forall \sigma' \leq \sigma . \delta(x_0, \sigma') \in R$ (R-reachable)
2. $\forall x \in R .\ \nexists u \in \Sigma_u . \delta(x, u) \in R^C$ (closed with respect to Σ_u)

Additionally, R is *(non-blocking) controllable* with respect to the automaton G if the following condition also holds:

3. $\forall x \in R . \exists \sigma \in \Sigma^* .\ [\delta(x, \sigma) \in Q_m \wedge \forall \sigma' \leq \sigma . \delta(x, \sigma') \in R]$ (Non-Blocking)

□

For completeness, the analogous definition for languages is included.

Definition 3.3. ([77]) **Controllable and Non-blocking Languages**

A language $K \subseteq L(G)$ is *controllable* with respect to the automaton $G = (X, \Sigma_u \cup \Sigma_c, \delta, x_0, Q_m)$ if $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$. The language K is also *non-blocking* if $\overline{K} = \overline{\overline{K} \cap L_m(G)}$. □

In [**54**] it is shown that there exists a state-feedback supervisor that realizes each R-reachable controllable state-set R (in the sense that, under the set of enablements prescribed at each state by the supervisor, the reachable set is exactly R). Furthermore, it is clear that for each R-reachable controllable set R, there exists a unique maximally enabling state-feedback supervisor that realizes R, i.e. one that enables the largest possible set of events at each reachable state, while still restricting the reachable set to R.

Define the operation $\cup$ on supervisors by $f \cup g : \Sigma^* \longrightarrow 2^\Sigma$, $f \cup g(\sigma) \stackrel{def}{=} f(\sigma) \cup g(\sigma)$, i.e. $f \cup g$ inhibits a transition only if both f and g inhibit the transition. The maximally enabling state-feedback supervisor realizing R as the reachable set can be found by taking the union, $\bigcup f$, over all state-feedback supervisors f that realize R. This is a different issue to that of finding a maximal controllable sublanguage or sub-set of states for a given (possibly uncontrollable) specification. In the present discussion, the state-set R is assumed to be controllable yet there may be more than one state-feedback supervisor which realizes R.

3.1. Internal Requirements for Controllability

The control inputs at the π level are disablements of transitions in Σ_c^π but are to be enacted by local low-level state-feedback supervisors. This requires the existence of a low-level controllable state-set in the sub-automaton that allows reachability of neighbouring blocks as requested by the π-level supervision. The key point as far as achieving specifications at the π level is the existence of such a controllable state-set for *every* set of disablements at the π level. For example in G_2 in Example 3.1 there was a missing instantiation of the π-level disablement $f^\pi(X_1) = \{V_1^4, U_1^2\}$.

To ensure that there is such an instantiating state-set for each control in each block, further conditioning is required. This is given below for a block X_i and the collections of blocks $P_i^d = \{X_j : \langle X_i, X_j \rangle_d\}$, $P_i^u = \{X_j : \langle X_i, X_j \rangle_u\}$. This definition constitutes the supervisory control counterpart to the IBC definition in [**15**] for forced event systems. In particular, in the cases where all events are controllable ($\Sigma = \Sigma_c$)

or all events are uncontrollable ($\Sigma = \Sigma_u$) this definition and that of *ST-in-block-controllability* [15] differ only in the latter's requirement for mutual accessibility of out-sets (i.e. the set of states that are either goal states or from which transitions lead out of the block).

Definition 3.4. (Non-Blocking) In Block Controllable (IBC) Partition Automaton

A *Trace-DC* partition automaton G^π is *(non-blocking) IBC* if for all blocks $X_i \in \pi$, both of the following hold,

(i) $\forall X_j \in P_i^d . \forall x \in I(X_i, Q_0) . \exists R_{X_i,x}^{X_j} \subseteq X_i^+$ such that

1. $\forall k . [X_k \cap R_{X_i,x}^{X_j} \neq \emptyset] \Longleftrightarrow [k{=}i \ \vee \ k{=}j \ \vee \ X_k \in P_i^u]$, and
2. $R_{X_i,x}^{X_j}$ is (non-blocking) controllable w.r.t. the sub-automaton $G_{X_i^+}(x, P_i^u \cup [X_j \cap X_i^+])$.

(ii) $[X_i \cap Q_m \neq \emptyset] \Longrightarrow \forall x \in I(X_i, Q_0) . \exists R_{X_i,x}^{Q_m} \subseteq X_i$ such that

1. $Q_m \cap R_{X_i,x}^{Q_m} \neq \emptyset$, and
2. $R_{X_i,x}^{Q_m}$ is (non-blocking) controllable w.r.t. the sub-automaton $G_{X_i^+}(x, Q_m \cap X_i)$.

In this event, the partition π is also termed *(non-blocking) IBC.* □

Example 3.2. *Figure 2.5 shows possibilities for the four required controllable state-sets (shown as shaded regions) for condition (i) of the (non-blocking) IBC condition at block X_1 . The slanted inhibition lines are used to represent the status "inhibitable but not inhibited". Consider the the third state-set from the left (which would correspond to $R_{X_1,x_0}^{X_2}$ in Definition 3.4), in which the high-level transition U_1^3 is inhibited. Note that blocks X_2 and X_4 are still (non-blocking) accessible from x_0 because $R_{X_1,x_0}^{X_2}$ is (non-blocking) controllable with respect to the automaton $G_{X_1^+}(x_0, \{x_2, x_3\})$ which has goal states in the blocks X_2 and X_4.* □

Note that due to the definition of $\langle , \rangle_u$, any controllable subset $R \subseteq X_i^+$ must always contain states from each block in P_i^u, i.e. it is never possible, through low-level control, to block high-level uncontrollable transitions.

Condition (ii) of the (non-blocking) IBC condition assures that, within each π-level goal state, non-blocking goal state reachability is guaranteed to hold.

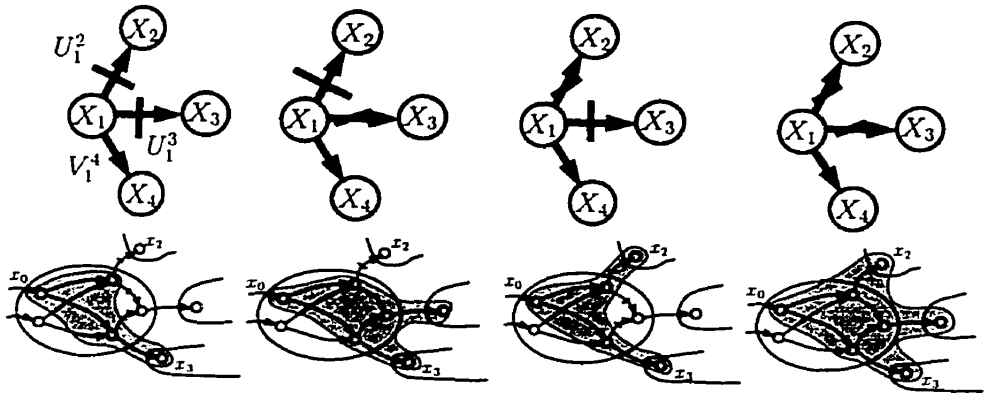


FIGURE 2.5. The required (non-blocking) controllable state-sets for (non-blocking) IBC condition (i) for the in-set state x_0.

Unfortunately, unlike the situation in [**19, 15, 17**], the (non-blocking) IBC condition is not preserved, in general, under either the greatest lower bound (intersection) or least upper bound (chain union) operations in the lattice of partitions. An example of the loss of the (non-blocking) IBC condition under the chain union operation is given in Figure 2.6, where two partitions (one marked with dashed lines, the other with solid lines and singletons assumed unless otherwise shown) are themselves (non-blocking) IBC, but their chain union is not (the blocks $\{x_3, x_4, x_5\}$, $\{x_2\}$ fail the *Trace DC* condition).

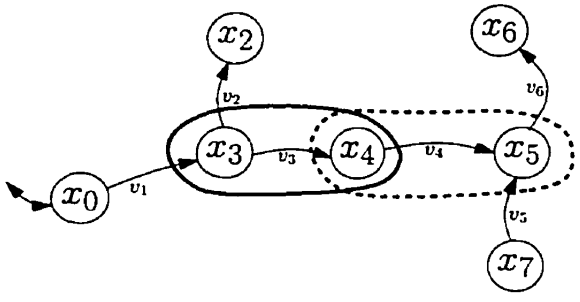


FIGURE 2.6. (non-blocking) IBC is not preserved under the chain union operation.

The IBC condition is transitive along the hierarchical layering. Let Π be a partition of π, i.e. $\Pi = \{\mathcal{X}_i, i = 1...|\Pi|\}$ such that $\bigcup \mathcal{X}_i = \pi$, $\forall i. \mathcal{X}_i \neq \emptyset$, and $\forall i \neq j \quad \mathcal{X}_i \cap \mathcal{X}_j = \emptyset$. If G^π is (non-blocking) IBC and G^Π is (non-blocking) IBC (with G^π as the low level machine), then $G^{\pi'}$ is (non-blocking) IBC, where $\pi' = \{X_i' | X_i' = \bigcup_{X_j \in \mathcal{X}_i} X_j\}$.

3.2. Synthesis of Supervisors through Sequential Refinement

Singleton blocks satisfy trivially the requirements in the definition of (non-blocking) IBC, hence the identity partition, $\pi^{id} = \{\{x\}|x \in X\}$ is (non-blocking) IBC. A *Trace-DC hierarchical control structure* is a chain, or sequence of (non-blocking) IBC partitions of increasing refinement. The key motivation is that specification and analysis may be performed at any level of granularity and the resulting supervisors may be translated down this chain of partitions, yielding subsequent levels of control refinement.

Definition 3.4 posits the existence of controllable state-sets. Hence for a given (non-blocking) IBC π-partition automaton, for each $\langle X_i, X_j \rangle_d$, $(1 \leq i, j \leq |\pi|)$ and $x \in I(X_i, Q_0)$, the maximal (non-blocking) controllable state-set, which will be labelled $R^{X_j}_{X_i,x}$, can be found by taking the union of all sets satisfying the (non-blocking) controllable condition. Similarly, for each block $X_i \in Q^{\pi}_m (1 \leq i \leq |\pi|)$ and state $x \in I(X_i, Q_0)$, the maximal (non-blocking) controllable state-set can be labelled by $R^{Q_m}_{X_i,x}$.

Furthermore, the maximally permissive state-feedback supervisors which realize $R^{X_j}_{X_i,x}$ and $R^{Q_m}_{X_i,x}$ are labelled by $f^{X_j}_{X_i,x} : X_i \longrightarrow 2^{\Sigma}$ and $f^{Q_m}_{X_i,x} : X_i \longrightarrow 2^{\Sigma}$ respectively.

For a π-level language specification $K^{\pi} \subseteq L(G^{\pi})$, the following scheme translates the control $H^{\pi} : L(G^{\pi}) \longrightarrow 2^{\Sigma^{\pi}}$, which synthesises K^{π} in G^{π}, to a low-level control $h_{low} : L(G) \longrightarrow 2^{\Sigma}$ (now possibly history-dependent by the dependence on H^{π} in its construction).

IBC Synthesis Algorithm $K_{low}^{IBC}()$

Input : H^{π}

[a] For $s \in L(G)$ such that $\Theta_{\pi}(s) = \epsilon$, let

$$h_{low}(s) = \bigcup_{W \in H^{\pi}(\epsilon)} f_{X_0,x_0}^{\delta^{\pi}(X_0,W)}(\delta(x_0,s)).$$

[b] For $s \in L(G)$ such that $\Theta_{\pi}(s) = S$, let

$$h_{low}(s) = \bigcup_{W \in H^{\pi}(S)} f_{X',x_0'}^{\delta^{\pi}(X',W)}(\delta(x_0,s)),$$

where $X' = \delta^{\pi}(X_0, S)$, $x_0' = \delta(x_0, s')$ and s' is a minimal string such that $\Theta_{\pi}(s') = S$ and $s' \leq s$ (i.e. for any other s'' satisfying these requirements, $s' \leq s'' \leq s$).

[c] Finally, for $s \in L(G)$ such that $\Theta_{\pi}(s) = S \in L_m(G^{\pi})$,

$$h_{low}(s) = f_{X',x_0'}^{Q_m}(\delta(x_0,s)),$$

where, again, $X' = \delta^{\pi}(X_0, S)$, $x_0' = \delta(x_0, s')$ and s' is a minimal string such that $\Theta_{\pi}(s') = \Theta_{\pi}(s)$ and $s' \leq s$.

Output: h_{low}

The scheme is illustrated in Figure 2.7 for the intermediate case [b]. Consider the low-level supervision after a low-level string s. $h_{low}(s)$ is to be calculated via the IBC Synthesis Algorithm. In this case, the high-level control action at $S = \Theta_{\pi}(s)$ is $H^{\pi}(S)$ which means U_3^6 is inhibited (this is the inhibitable arc between X_3 and X_6). Within the block X_3 this high-level control action is translated to the low level state-feedback as $f_{X_3,x'}^{X_4}$ which inhibits flow to X_6 as illustrated. Hence the control to be applied at s is $h_{low}(s) = f_{X_3,x'}^{X_4}(x)$ where $x = \delta(x_0, s)$.

Let the unique low-level language resulting from the control h_{low} found through the IBC synthesis algorithm be $K_{low}^{IBC}(K^{\pi})$. An illustration of this high-to-low synthesis for (non-blocking) IBC partitions is provided in the following chapter in the context of manufacturing systems.

4. Controllable Sub-Languages of the π-Automaton

The principal theoretical results of this chapter are now given. It will be shown first in Theorem 4.1 that the *Trace-DC* condition alone achieves the same consistency result as that of *output control consistency*; namely that the π-level image of

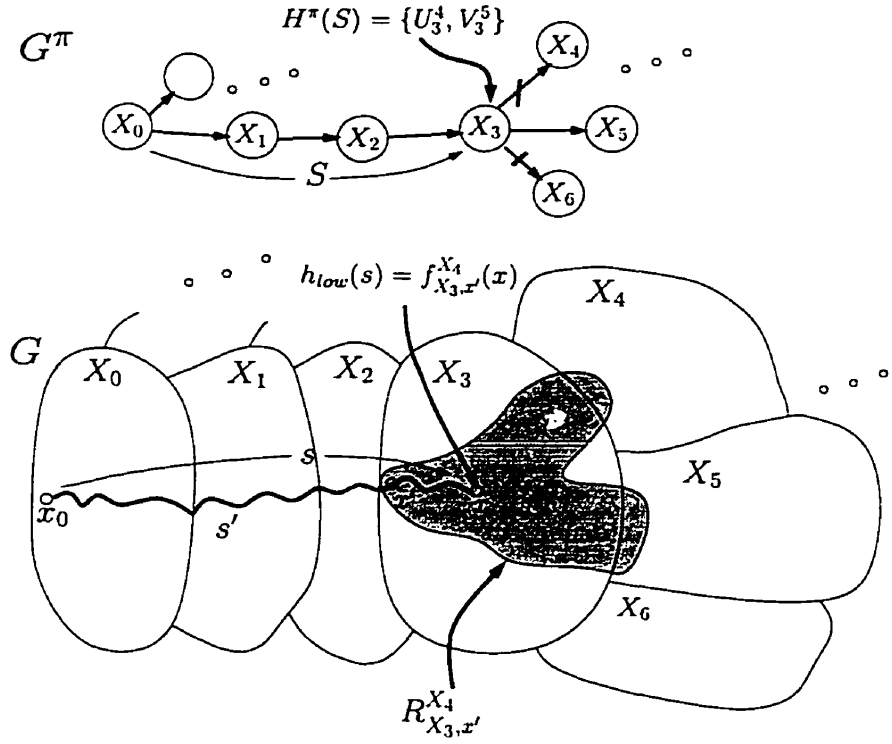


FIGURE 2.7. Translation of control from high to low levels.

a controllable language is controllable. It will then be shown in Theorem 4.2 that, with the added (non-blocking) IBC hypothesis, the inverse holds; namely that for any controllable π-level language, there exists a low-level controllable language with the π-level language as its image. These results are shown in the setting of languages and then as a corollary are obtained for state-sets.

Theorem 4.1. *Consider a Trace-DC partition π of X in the automaton*

$$G = (X, \Sigma_u \dot{\cup} \Sigma_c, \delta, Q_0, Q_m),$$

and any $x_0 \in Q_0$. If a language K is non-blocking and controllable w.r.t. G then $\Theta_\pi(K)$ is non-blocking and controllable w.r.t.

$$G^\pi = (\pi, \Sigma_u^\pi \dot{\cup} \Sigma_c^\pi, \delta^\pi, Q_0^\pi, Q_m^\pi),$$

where $Q_0^\pi \stackrel{\text{def}}{=} \{X_i \in \pi | X_i \cap Q_0 \neq \emptyset\}$ and $Q_m^\pi \stackrel{\text{def}}{=} \{X_i \in \pi | X_i \cap Q_m \neq \emptyset\}$.

Proof. Let $\sigma^\pi \in \overline{\Theta_\pi(K)}$ and $V \in \Sigma_u^\pi$ and let $\sigma^\pi V \in L(G^\pi)$. Also, let $\sigma \in \overline{K}$ be an instantiating string such that $\Theta_\pi(\sigma) = \sigma^\pi$. That $\Theta_\pi(K)$ is non-blocking and controllable can be shown (independently) as follows.

(controllable) V represents an uncontrollable DC link $\langle X_i, X_j \rangle_u$ for some $1 \leq i, j \leq |\pi|$. Let $\sigma' \leq \sigma$ be such that $\delta(x_0, \sigma') \in I(X_i, Q_0)$. Since $\langle X_i, X_j \rangle_u$, σ' can be continued by $s \in \Sigma_u^*$ such that $\delta(x_0, \sigma' s) \in X_j$. But K is controllable, so $\sigma' s \in \overline{K}$, meaning $\sigma^\pi V = \Theta_\pi(\sigma s) \in \overline{\Theta_\pi(K)}$. Hence $\Theta_\pi(K)$ is controllable.

(non-blocking) As K is non-blocking, σ can be continued by some $s \in \Sigma^*$ to the goal states. All low-level goal states are contained in π-level goal states so the image $\Theta_\pi(\sigma)$ can be continued to $\Theta_\pi(\sigma s)$ such that $\delta^\pi(X_0, \Theta_\pi(\sigma s)) \in Q_m^\pi$. ∎

In order to translate the results from languages to state-sets, the following observations are needed. First, note that for any non-blocking, controllable subset R, there exists a unique maximal non-blocking and controllable language L_R that has R as the reachable state-set. Second, note that for any non-blocking, controllable language K, the (unique) reachable state-set must also be non-blocking and controllable.

Corollary 4.1. *Consider a Trace-DC partition π of X in the automaton G and any $x_0 \in Q_0$. If $R \subseteq X$ is (non-blocking) controllable w.r.t. G then $\Theta_\pi(R)$ is (non-blocking) controllable w.r.t. G^π.*

Proof. It can be verified that the reachable set (of π-level states) associated with $\Theta_\pi(L_R)$ is $\Theta_\pi(R)$. The (non-blocking) controllability of $\Theta_\pi(R)$ follows from that of $\Theta_\pi(L_R)$ (which is non-blocking and controllable via Theorem 4.1). ∎

The construction in the previous section via the synthesis algorithm K_{low}^{IBC} allows the formation of a low-level language $K_{low}^{IBC}(K^\pi)$ from a high-level specification K^π. This construction is effective in the sense that it yields a non-blocking, controllable low-level language which has the correct image K^π.

Theorem 4.2. *Consider a (non-blocking) IBC partition π of X in the automaton*

$$G = (X, \Sigma_u \dot{\cup} \Sigma_c, \delta, Q_0, Q_m)$$

and any $x_0 \in Q_0$. If K^π is non-blocking and controllable w.r.t.

$$G^\pi = (\pi, \Sigma_u^\pi \dot{\cup} \Sigma_c^\pi, \delta^\pi, Q_0^\pi, Q_m^\pi)$$

then $K_{low}^{IBC}(K^\pi)$ is non-blocking and controllable w.r.t. G and $\Theta_\pi(K_{low}^{IBC}(K^\pi)) = K^\pi$.

Proof. (controllable) Let $\sigma \in \overline{K_{low}^{IBC}(K^\pi)}$ and let $v \in \Sigma_u$ be such that $\sigma v \in L(G)$. Let σ' be the shortest prefix of σ with the same image (i.e. $\sigma' \leq \sigma$ and $\Theta_\pi(\sigma') = \Theta_\pi(\sigma) = \sigma^\pi$ and for any other such σ'', $\sigma' \leq \sigma''$). Further, let $x' = \delta(x_0, \sigma')$ and $X' = \delta^\pi(X_0, \sigma^\pi)$.

The string σ is realizable under h^{IBC}_{low} (i.e. is an element of the language generated by G under the control h^{IBC}_{low}) since, by assumption, $\sigma \in \overline{K^{IBC}_{low}(K^\pi)}$. So for at least one $W \in H^\pi(\sigma^\pi)$ with $X'' = \delta^\pi(X', W)$, it is the case that $v \in f^{X''}_{X',x'}(\delta(x_0, \sigma))$ (or $f^{Q_m}_{X',x'}(\delta(x_0,\sigma))$ if X' is already in the goal set Q^π_m). Hence,

$$v \in h^{IBC}_{low}(\sigma) = \bigcup_{W \in H^\pi(\sigma^\pi)} f^{\delta^\pi(X_0, \sigma^\pi W)}_{X',x'}$$

and therefore $K^{IBC}_{low}(K^\pi)$ is controllable.

(non-blocking) Let $\sigma \in \overline{K^{IBC}_{low}(K^\pi)}$, and let $\sigma^\pi = \Theta_\pi(\sigma)$ be continued by $S = S_1 S_2 \cdots S_{|S|}$, where $S_i \in \Sigma^\pi, i = 1...|S|$, such that $\delta^\pi(X_0, \sigma^\pi S) \in Q^\pi_m$ (which is possible as K^π is non-blocking).

We may recursively construct an instantiating string $s = s_1 s_2 \cdots s_{|S|+1}$, $s_i \in \Sigma^*$, by finding, in succession:

[a] s_1 such that $\sigma s_1 \in K^{IBC}_{low}$ and

$$\delta(x_0, \sigma) \quad \overrightarrow{s_1} \quad I(\delta^\pi(X_0, \sigma^\pi S_1), Q_0),$$

[b] s_i, for $2 \le i \le |S|$, such that $\sigma s_1 \cdots s_i \in K^{IBC}_{low}(K^\pi)$ and

$$I(\delta^\pi(X_0, \sigma^\pi S_1 S_2 \cdots S_i), Q_0) \ \overrightarrow{s_i} \quad I(\delta^\pi(X_0, \sigma^\pi S_1 S_2 \cdots S_{i+1}, Q_0)),$$

[c] $s_{|S|+1}$ such that

$$I(\delta^\pi(X_0, \sigma^\pi S), Q_0) \ \overrightarrow{s_{|S|+1}} \ Q_m.$$

In cases [a] and [b], such a string exists because

$$h^{IBC}_{low}(\sigma) \supseteq f^{A(S,i+1)}_{A(S,i),B(s,i)}(\delta(x_0, \sigma)),$$

where $A(S,i) = \delta^\pi(X_0, \sigma^\pi S_1 S_2 \cdots S_i)$ and $B(s,i) = \delta(x_0, \sigma s_1 s_2 \cdots s_i)$. This is the case for all σ such that $\Theta_\pi(\sigma) = \sigma^\pi S_1 S_2 \cdots S_i$ since $S_{(i+1)} \in H^\pi(\sigma^\pi S_1 S_2 \cdots S_i)$, by assumption, and the application of each local function $f^{\cdot}_{\cdot,\cdot}$ results in a (non-blocking) controllable state-set. For the case $i = |S| + 1$, we can find a string to instantiate the final portion within the high-level goal state $\delta^\pi(X_0, \sigma^\pi S)$, because we use the local non-blocking control $f^{Q_m}_{\delta^\pi(X_0,\sigma^\pi S),\delta(x_0,\sigma s)}$.

Hence for any $\sigma \in \overline{K^{IBC}_{low}(K^\pi)}$, there exists s such that $\sigma s \in \overline{K^{IBC}_{low}(K^\pi)}$ and $\delta(x_0, \sigma s) \in Q_m$. Hence $K^{IBC}_{low}(K^\pi)$ is non-blocking.

($K^\pi \subseteq \Theta_\pi(K^{IBC}_{low}(K^\pi(K^\pi))))$ Let $S = S_1 S_2 \cdots S_{|S|} \in K^\pi$. S may be instantiated by $s \in K^{IBC}_{low}(K^\pi)$ such that $\Theta_\pi(s) = S$; this is by the same construction used for the

proof above of non-blocking starting from the empty string ϵ and leading to the block $\delta^{\pi}(X_0, S)$. Hence $S \in \Theta_{\pi}(K_{low}^{IBC}(K^{\pi}))$.

$(\Theta_{\pi}(K_{low}^{IBC}(K^{\pi})) \subseteq K^{\pi})$ Let $S = S_1 S_2 \cdots S_{|S|} \in \Theta^{\pi}(K_{low}^{IBC}(K^{\pi}))$ and $s = s_1 s_2 ... s_{|S|} \in K_{low}$ be such that for each i,
$\delta(x_0, s_1 s_2 ... s_i) \in I(\delta^{\pi}(X_0, S_1 S_2 \cdots S_i), Q_0)$, i.e. for each pair $S_i S_{i+1}^{\pi}$, s_i connects in-set to subsequent in-set.

Recall the construction of h_{low} in $\delta^{\pi}(X_0, S_1 S_2 \cdots S_i)$, and the fact that non-blocking controllability and reachability from a given start state are closed with respect to union. Hence it is the case that $\delta(x_0, s_1 s_2 \cdots s_{i+1})$ is reachable from $I(\delta^{\pi}(X_0, S_1 S_2 \cdots S_i), Q_0)$ if and only if $S_{i+1} \in H^{\pi}(S_1 S_2 \cdots S_i)$. But this means S is a realizable string under the application of H^{π}, i.e. $S \in K^{\pi}$. ∎

By the very specification of Θ_{π} and the definition of controllable state subsets in the π-partition automaton G^{π}, one obtains:

Corollary 4.2. *Consider a π-level (non-blocking) IBC partition of X in the automaton G and any $x_0 \in Q_0$. If $R^{\pi} \subseteq \pi$ is (non-blocking) controllable w.r.t. G^{π} then there exists a set $R \subseteq X$ which is (non-blocking) controllable w.r.t. G such that $\Theta_{\pi}(R) = R^{\pi}$.*

Proof. To R^{π} can be associated a unique maximal nonblocking controllable language $K_{R^{\pi}}^{\pi}$. From Theorem 4.2, $K_{low}^{IBC}(K_{R^{\pi}}^{\pi})$ is non-blocking, controllable and $\Theta_{\pi}(K_{low}^{IBC}(K_{R^{\pi}})) = K_{R^{\pi}}$. Hence the reachable state-set R associated with $K_{low}^{IBC}(K_{R^{\pi}})$ also has these properties. ∎

4.1. Hierarchical Consistency and (Non-blocking) IBC

To further the comparison with the work in [**103**] (in the light of Theorem 4.2) the definition of *hierarchical consistency* is now provided. Let $G_{lo} = (Z, \Sigma, T, \delta, \Theta, x_0)$ be a Moore automaton where $\Theta : Z \longrightarrow T$ is an output map from state to output symbols in T. In analogy with the map Θ_{π} defined in Section 2, the map Θ can be extended to $\Theta : 2^{L(G_{lo})} \longrightarrow 2^{T^*}$ which associates to each low-level language E_{lo} a high-level language E_{hi} composed of strings of symbols observed along the state trajectories of the strings in E_{lo}. Let G^{hi} be an automaton with alphabet T, i.e. $L(G^{hi}) \subseteq T^*$.

Definition 4.1. ([103]) **Hierarchical Consistency**
A pair (G_{lo}, G^{hi}) possesses *hierarchical consistency* if $L_m(G^{hi}) = \Theta(L_m(G_{lo}))$ and for every non-empty, closed, controllable language $E_{hi} \subseteq L_m(G_{hi})$,

$$\Theta(\Theta^{-1}(E_{hi})^{\uparrow}) = E_{hi}$$

where $()^{\uparrow}$ is the maximal controllable sub-language operator. □

The symbol $\tilde{G}_{\Theta_\pi}$ will continue to represent a Moore automaton as defined in Section 2 via the translation from Mealy to Moore automata. Note that a language is controllable with respect to G if and only if it is controllable with respect to $\tilde{G}_{\Theta_\pi}$.

Theorem 4.3. *Consider a (non-blocking) IBC partition π of X in the automaton G and any choice of x_0 in Q_0. Then the pair $(\tilde{G}_{\Theta_\pi}, G^{\pi})$ possesses hierarchical consistency.*

Proof. Consider $E^{hi} \subseteq L_m(G^{\pi})$, a non-empty, closed, controllable language, and let $E^{\uparrow}_{lo}$ be the maximal controllable language satisfying $\Theta(E^{\uparrow}_{lo}) \subseteq E^{\pi}$, i.e. $E^{\uparrow}_{lo} = (\Theta^{-1}(E_{hi}))^{\uparrow}$. For any choice of $x_0 \in Q_0$, the mapping $E_{hi} \longrightarrow K^{IBC}_{low}(E_{hi}) \stackrel{=}{\nabla} E^{IBC}_{low}$ of the IBC synthesis algorithm gives a low-level language E^{IBC}_{low} controllable with respect to G such that $\Theta(E^{IBC}_{low}) = E^{hi}$ (by Theorem 4.2). By assumption $E^{IBC}_{low} \subseteq E^{\uparrow}_{lo}$ so we have that $E_{hi} \supseteq \Theta(E^{\uparrow}_{lo}) \supseteq \Theta(E^{IBC}_{low}) = E^{hi}$, and hence $\Theta(\Theta^{-1}(E_{hi})^{\uparrow}) = \Theta(E^{\uparrow}_{lo}) = E^{\pi}$, as required. ■

Hierarchical Consistency of the pair $(\tilde{G}_{\Theta_\pi}, G^{\pi})$ does not, in general, imply that G^{π} is (non-blocking) IBC. A counter-example is as follows.

Example 4.1. *In Figure 2.8,* Hierarchical Consistency *of the pair $(\tilde{G}_{\Theta_\pi}, G^{\pi})$ can be verified by checking the four high-level controllable languages accepted by G^{π} and*

their associated inverse images, e.g.

$$
\begin{aligned}
\{\epsilon, U_1^2\}: \quad & \Theta^{-1}\{\epsilon, U_1^2\} = \{\epsilon, a, b, ac, be\} \\
& (\Theta^{-1}\{\epsilon, U_1^2\})^{\uparrow} = \{\epsilon, a, b, ac\} \\
& \Longrightarrow \Theta((\Theta^{-1}\{\epsilon, U_1^2\})^{\uparrow}) = \{\epsilon, U_1^2\} \\
\{\epsilon, U_1^2, U_1^2U_2^3\}: \quad & \Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^3\} = \{\epsilon, a, b, ac, be, acd, bef\} \\
& (\Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^3\})^{\uparrow} = \{\epsilon, a, b, ac, acd\} \\
& \Longrightarrow \Theta((\Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^3\})^{\uparrow}) = \{\epsilon, U_1^2, U_1^2U_2^3\} \\
\{\epsilon, U_1^2, U_1^2U_2^4\}: \quad & \Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^4\} = \{\epsilon, a, b, ac, be, acg, beh\} \\
& (\Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^4\})^{\uparrow} = \{\epsilon, a, b, ac, acg\} \\
& \Longrightarrow \Theta((\Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^4\})^{\uparrow}) = \{\epsilon, U_1^2, U_1^2U_2^4\} \\
\{\epsilon, U_1^2, U_1^2U_2^3, U_1^2U_2^4\}: \quad & \Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^3, U_1^2U_2^4\} = L(\tilde{G}_{\Theta_\pi}) \\
& \Longrightarrow \Theta((\Theta^{-1}\{\epsilon, U_1^2, U_1^2U_2^3, U_1^2U_2^4\})^{\uparrow}) = \{\epsilon, U_1^2, U_1^2U_2^3, U_1^2U_2^4\}.
\end{aligned}
$$

This partition is Trace-DC but is not (non-blocking) IBC due to the canonical lack of control of the flow after transition e. □

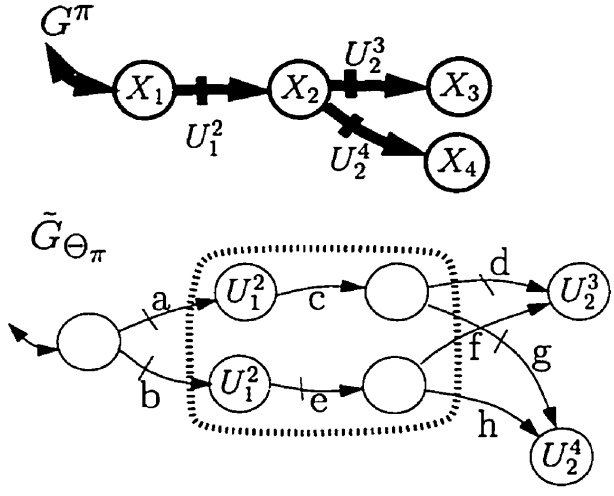


FIGURE 2.8. A Hierarchically Consistent pair where G^π is not IBC.

It is still possible to discuss the consistency criteria from the perspective of *existence* of the partitions, i.e. does *hierarchical consistency* imply the existence of *(non-blocking)* IBC partitions? And if not, under what conditions is this the case? These are discussed in the following section.

5. (Non-blocking) IBC Partitions and the *Vocalised Lifting* Algorithm

Given the appealing transparency of a hierarchy based on state aggregation and the requirement for *hierarchical consistency* between layers, an immediate issue that arises is whether all systems that exhibit *hierarchical consistency* will have IBC partitions. This is generally not the case as is shown in Example 5.1 below.

Example 5.1. *Figure 2.9 illustrates a pair* (G_{lo}, G^{hi}) *where* G_{lo} *is a Moore automaton and the automaton* G^{hi} *is such that* $L(G^{hi}) = \Theta(L(G_{lo}))$. (G_{lo}, G^{hi}) *possess* hierarchical consistency *yet there does not exist a partition* $\bar{\pi}$ *of the state space of* G_{lo} *such that the partition automaton* $G^{\bar{\pi}}$ *is isomorphic to* G^{hi}. *This is because* x_3 *cannot be placed in a block with any of the other states. For example, if* x_3 *is placed in a block with* x_2 *as illustrated on the diagram then* $< X_1, X_2 >_u$ *holds yet there is no high-level transition between* X_1 *and* X_2 *in* G^{hi} *as there would be in* $G^{\bar{\pi}}$. *Similarly, if* x_3 *is placed with* x_4 *then* $< X_3, X_4 >_u$ *holds yet, again, there is no high-level transition between* X_3 *and* X_4 *in* G^{hi} *as there would be in* $G^{\bar{\pi}}$. x_3 *cannot be placed as a singleton block as there are insufficient states in* G^{hi}. □

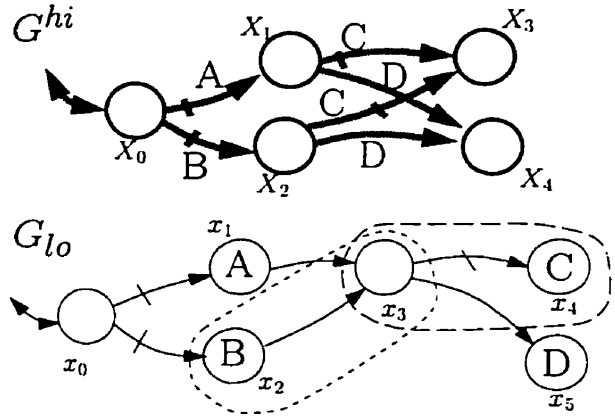


FIGURE 2.9. A hierarchically consistent pair for which there is no partition automaton isomorphic to G^{hi}.

The essential problem in Example 5.1 is that the output map for G_{lo}, when translated to a Mealy output, bears no representation in the form of Equation 2.2.2 which is the partition-based Mealy machine output (the output map for G_{lo} is based on the current state *and* the current event which can be accomplished by setting, for all $x \in X$ and $\sigma \in \Sigma$, $\Theta_{Mealy}(\sigma, x) = \Theta_{Moore}(x)$). To treat this issue, a so-called *Vocalised Lifting (VL)* is provided below that extends the state space in order to make the

system amenable to state-aggregation. The goal is to create, presumably at the design stage, a state-based hierarchical structure that maintains *hierarchical consistency.* The advantage is that the a state-based hierarchy is now available and the level of granularity can be chosen to suit the design considerations. The disadvantage is that the cardinality of the state-space is increased. Essentially, minimality is sacrificed for regularity.

Algorithm VL

Input: $G = (X, \Sigma, \delta, Q_0, Q_m)$, $X_{seed} \subseteq X$.

Output: $G_{VL} \stackrel{def}{=} (X^{VL}, \Sigma, \delta^{VL}, Q_0^{VL}, Q_m^{VL})$, π_{VL}.

1. (closure from Seeds)
   For each state $x \in X_{seed} \cup Q_0$, compute the forward closure,
   $$Y_x = \{x' \in X | \exists s \in \Sigma^*.[(\delta(x,s) = x') \wedge (\forall s', \epsilon > s' \leq s.\delta(x,s') \notin X_{seed})]\}.$$
   and define the cover $\mathcal{K} \stackrel{def}{=} \{Y_x \mid x \in X_{seed} \cup Q_0\}$ and number the blocks $Y_1, ...Y_{|\mathcal{K}|}$.
2. (definition of X^{VL}, π_{VL})
   For each state $x_k \in X$, if $x_k \in Y_{i_1} \cap Y_{i_2} \cap \cdots \cap Y_{i_n}$, (where $1 \leq n \leq |\mathcal{K}|$), define the states $x_{k,i_1}^{VL}, x_{k,i_2}^{VL}, ..., x_{k,i_n}^{VL}$ in a new state set X^{VL} and let,
   $$x_{k,i_1}^{VL} \in X_{i_1}^{VL,\pi}$$
   $$x_{k,i_2}^{VL} \in X_{i_2}^{VL,\pi}$$
   $$\cdots$$
   $$x_{k,i_n}^{VL} \in X_{i_n}^{VL,\pi}$$
   where $X_i^{VL,\pi}, 1 \leq i \leq |\mathcal{K}|$ are blocks in a partition π_{VL} of X^{VL}. The sets X^{VL} and $X_i^{VL,\pi}$ contain no other states other than those specified by this step.
3. (definition of δ_{VL})
   For $\sigma \in \Sigma$ and $x_k, x_l \in X$, the (partial) transition function δ_{VL} is defined such that
   $$\delta^{VL}(x_{k,i}^{VL}, \sigma) = x_{l,j} \quad \text{iff} \quad \begin{array}{l}(i)\ i = j \text{ and } \delta(x_k, \sigma) = x_l, or \\ (ii)\ x_l \in X_{seed} \cup Q_0 \text{ and } \delta(x_k, \sigma) = x_l.\end{array}$$

4. (definition of Q_0^{VL}, Q_m^{VL})
   Define the start states and goal states $Q_0^{VL} = \{x_i^{VL} | x \in Q_0\}$ be $Q_m^{VL} = \{x_i^{VL} | x \in Q_m\}$ respectively.

The VL algorithm creates a new supervisory automaton G_{VL} from G and a partition π_{VL} of the state space of G_{VL}. For each state in $X_{seed} \cup Q_0$, the effect of this algorithm is to create one state in X^{VL}. Each such new state is placed in a different partition element of π_{VL} (note that the cardinality π_{VL} is also $|X_{seed}| + |Q_0|$).

The effect of the algorithm for each state in $X\bar{(}X_{seed} \cup Q_0)$ is to create up to $|X_{seed}| + |X_0|$ states in the new set X^{VL}, and place them in appropriate blocks in π_{VL} such that they share a block with a seed state or initial state if and only if the state for which they were created is reachable from the original seed state. The transition function is then defined for G_{VL} such all cross-boundary transitions end in a seed state.

A similar algorithm to the VL algorithm was presented in [**103**]. It should be noted that the algorithms have different goals; the VL algorithm seeks to create an IBC partition in a structure that was already *hierarchically consistent* while the algorithm in [**103**] seeks to enhance the system structure so that it is *hierarchically consistent* when previously it was not. Unlike the VL algorithm this was done in [**103**] by enhancing the observation function rather than increasing the number of states.

A possible interpretation of the seed points $X_{seed} \subseteq X$ is that of states in a Moore automaton that are not mapped to ϵ by the output map. A growth bound on the state-set cardinality is $O(n^2)$. In the worst case, all non seed nodes need to be put in each seed node's cover and seed nodes make up 50% of the total nodes.

Formally, the π_{VL}-partition automaton can be defined as follows.

Definition 5.1. $G^{\pi_{VL}}$

The partition automaton $G^{\pi_{VL}}$ is defined (in the same manner as partition automata in Section 2) to be,

$$G^{\pi_{VL}} \stackrel{def}{=} (\pi_{VL}, \Sigma_u^{\pi} \dot{\cup} \Sigma_c^{\pi}, \delta^{\pi_{VL}}, Q_0^{\pi_{VL}}, Q_m^{\pi_{VL}})$$

where π_{VL} is now the state-set. When $\langle X_i^{VL}, X_j^{VL} \rangle_d$ holds between two blocks, we define a π-level disableable transition U_i^j. Similarly, when $\langle X_i^{VL}, X_j^{VL} \rangle_u$ holds we define an undisableable π-level transition V_i^j. The (partially-defined) transition function,

$\delta^{\pi}_{VL} : \pi_{VL} \times \Sigma^{\pi} \to \pi_{VL}$, is defined such that when a transition exists; it forms a directed edge between the associated states, i.e.$\langle X^{VL}_i, X^{VL}_j \rangle_d \implies \delta^{\pi_{VL}}(X^{VL}_i, U^j_i) = X^{VL}_j$, and $\langle Y^{VL}_i, X^{VL}_j \rangle_u \implies \delta^{\pi}_{VL}(X^{VL}_i, U^j_i) = X^{VL}_j$. The set of π_{VL}-level initial states is $Q^{\pi_{VL}}_0 \stackrel{\text{def}}{=} \{X^{VL}_i \in \pi | X^{VL}_i \cap Q^{VL}_0 \neq \emptyset\}$. The set of π-level goal states is $Q^{\pi_{VL}}_m \stackrel{\text{def}}{=} \{X^{VL}_i \in \pi_{VL} | X^{VL}_i \cap Q^{VL}_m \neq \emptyset\}$. □

Example 5.2. *The VL algorithm is applied to Example 5.1 with the seed set* $X_{seed} = \{x_1, x_2, x_4, x_5\}$. *The result is shown in the bottom right of Figure 2.10. The effect is to split the state* x_3 *into* x^A_3 *and* x^B_3 *and place these in separate partition elements. Note that* $G^{\pi_{VL}}$ *is then isomorphic to* G^{hi}.

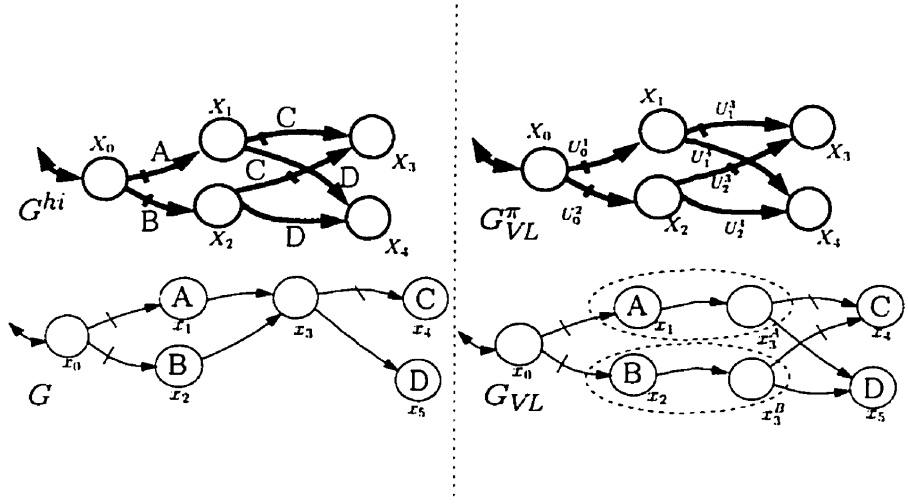


FIGURE 2.10. The result of the VL algorithm on Figure 2.9.

□

Example 5.3. *Another example of the application of the VL algorithm is given in Figure 2.11, where the states in* X_{seed} *are labelled A and B, and the states in* X^{VL} *are shown with subscripts tagging them to their respective seed nodes.* □

Theorem 5.1. G^{π}_{VL} *is* Trace-DC

Proof.

Note first that for every block $X^{VL}_i \in \pi_{VL}$: (i) $I(X^{VL}_i, Q_0) = \{y\}$ for some $y \in X_{seed} \cup Q_0$, i.e. there is only one in-set state per block; and (ii) by construction every state in the block X^{VL}_i is reachable from this in-set state y.

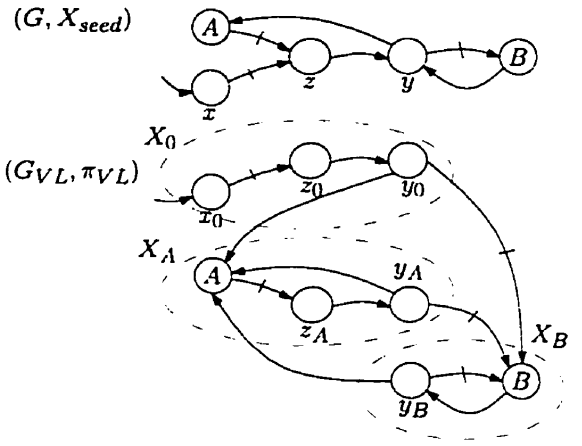


FIGURE 2.11. Illustration of the VL algorithm.

The Trace-DC condition can be restated as,

$\forall\, i \neq j, 1 \leq i, j \leq |\pi_{VL}|.$

$\{[\exists w \in \Sigma. \exists x' \in X_i^{VL}. \exists x'' \in X_j^{VL}.\ \delta(x', w) = x''] \Longrightarrow (\langle X_i^{VL}, X_j^{VL}\rangle_u \vee \langle X_i^{VL}, X_j^{VL}\rangle_d)\}$,

which is now proved.

Consider any pair of blocks $X_i^{VL}, X_j^{VL} \in \pi_{VL}$ and let there exist some $x' \in X_i^{VL}, x'' \in X_j^{VL}$ and $w \in \Sigma$ such that $\delta_{VL}(x', w) = x''$. The state x' is reachable from $I(X_i^{VL}, Q_0)$ and so there exists at least one path from $I(X_i^{VL}, Q_0)$ to X_j^{VL}. If all such paths contain at least one controllable transition, then, as there is only one in-set state in X_i^{VL}, it is the case that $\langle X_i^{VL}, X_j^{VL}\rangle_d$. Similarly, if there is at least one such path that is composed entirely of uncontrollable transitions, then $\langle X_i^{VL}, X_j^{VL}\rangle_u$.

■

The statement that minimality is sacrificed for regularity can now be clarified. Let X_{seed} be interpreted as a set of observable states of a Moore automaton (i.e. let the observation map $\Theta : X \longrightarrow T$, where T is a high-level alphabet map $X_{seed} \longrightarrow T/\{\epsilon\}$ and $X/X_{seed} \longrightarrow \epsilon$). Let (G, Θ) and (G_{VL}, Θ) denote the Moore machines formed via the observation Θ of the sets X and X_{VL}. The following remarks can be made regarding G , G_{VL}, (G, Θ) and (G_{VL}, Θ):

- $L(G) = L(G_{VL})$
- The string of observations $\tau_\sigma \in T^*$ along the state trajectory in G of a string $\sigma \in L(G)$ is the same as the string of observations along the state trajectory in G_{VL} of σ

- A language $K \subseteq L(G)$ is controllable w.r.t. G if and only if it is controllable w.r.t. G_{VL} since the (i) Σ_c and Σ_u retain their meaning in the alphabet of G_{VL} and (ii) after a string s, for all symbols $\sigma \in \Sigma$, $\delta(\delta(x_0, s), \sigma)! \iff \delta_{VL}(\delta_{VL}(x_0, s), \sigma' w)$, i.e. the set of symbols Λ_s that may follow a string s in G is equal to that which may follow s in G_{VL}.
- The Nerode equivalence classes for the mapping from Σ^* to T^* defined by the pair (G, Θ), i.e.

$$\sigma =_G \sigma' \iff \forall w.\ \Theta(\delta(x_0, \sigma w)) = \Theta(\delta(x_0, \sigma' w))$$

are identical to those for the mapping defined by the pair (G_{VL}, Θ), i.e.

$$\sigma =_{G_{VL}} \sigma' \iff \forall w.\ \Theta(\delta_{VL}(x_0, \sigma w)) = \Theta(\delta_{VL}(x_0, \sigma' w))$$

Hence, from an input-output perspective, the systems (G, Θ) and (G_{VL}, Θ) are equivalent (i.e. their respective maps Σ^* to T^* are identical). The state-set of G_{VL} is generally larger, hence the claim that the VL algorithm leads to loss of minimality (if indeed G was minimal).

5.1. The VL Algorithm and IBC partitions

Next it is shown that a non-trivial IBC partition of G_{VL} exists when (G, G^{hi}) are hierarchically consistent, G is unmarked (i.e. $Q_m = X$) and the map $\Theta : X_{seed} \longrightarrow T$ is injective.

Theorem 5.2. *Let (G, Θ) denote a Moore automaton with vocal nodes $X_{seed} \subseteq X$ and goal states $Q_m = X$ and let the automaton $G^{hi} = (X^{hi}, T, \delta^{hi}, Q_0^{hi}, Q_m^{hi})$ be such that $L(G^{hi}) = \Theta(L(G))$.*

If the following conditions hold for all $x_0 \in Q_0$:

(i) *the pair $((G, \Theta), G^{hi})$ is hierarchically consistent, and*
(ii) *$\Theta : X_{seed} \longrightarrow T$ is injective,*

then $G^{\pi_{VL}}$ is a (non-blocking) IBC partition automaton, i.e. the partition π_{VL} of X_{VL} is (non-blocking) IBC.

Proof. By Theorem 5.1, $G^{\pi_{VL}}$ is Trace-DC.

Since T is injective, there exists a many-to-one map $\Theta_\pi^{hi} : \Sigma^{\pi_{VL}} \longrightarrow T$ such that for all $E_{low} \subseteq L(G)$, it is the case that $\Theta(E_{low}) = \Theta_\pi^{hi}(\Theta_{\pi_{VL}}(E_{low}))$.

Let $X_i^{VL} \in \pi_{VL}$ and let $\gamma_i^j = \{V_i^k | \langle X_i^{VL}, X_k^{VL} \rangle_u\} \cup \{U_i^j\}$. It is shown that for all start states $q_0 \in Q_0$, there exists a controllable language $E_{hi} \subseteq L(G^{hi})$, such that for

all control policies $f^{\pi_{VL}}$ with $\Theta_\pi^{hi}(L(G^{\pi_{VL}}/f^{\pi_{VL}})) = E_{hi}$ it is the case that,
(1) $X_i^{VL} \in Rbl(G^{\pi_{VL}}/f^{\pi_{VL}})$ and (2) $f^{\pi_{VL}}(X_i^{VL}) = \{\gamma_i^j\}$.

This can be shown by construction for each γ_i^j of the appropriate language $E_{hi}(\gamma_i^j)$. Consider the state-feedback control for $G^{\pi_{VL}}$,

$$f^{\pi_{VL},\gamma_i^j} : \pi_{VL} \longrightarrow 2^{\Sigma^{\pi_{VL}}}, \qquad f^{\pi_{VL},\gamma_i^j}(X_k^{VL}) = \begin{cases} \gamma_i^j & k = i \\ 2^{\Sigma^{\pi_{VL}}} & elsewhere, \end{cases}$$

which inhibits only those transitions necessary in the application of the action γ_i^j at X_i^{VL}. Let $E_{hi}(\gamma_i^j) = \Theta_\pi^{hi}(L(G^{\pi_{VL}}/f^{\pi_{VL},\gamma_i^j}))$. This language is by construction controllable, and from the global assumption stated in Section 2 that all states in G are reachable from some start state x_0, it is the case that X_i^{VL} is reachable when the policy f^{hi,γ_i^j} is applied to $G^{\pi_{VL}}$ (satisfying (1)). Furthermore, event labels are unique in $G^{\pi_{VL}}$ (since labels are indexed by the preceding and following blocks) hence any other policy $f^{\pi_{VL}}$ such that $\Theta_\pi^{hi}(L(G^{\pi_{VL}}/f^{\pi_{VL}})) = E_{hi}(\gamma_i^j)$ must also apply γ_i^j at X_i^{VL} (satisfying (2)).

Since the pair $((G,\Theta),G^{hi})$ is hierarchically consistent, $E_{hi}(\gamma_i)$ necessarily has at least one controllable low-level counterpart E_{low} such that $\Theta(E_{low}) = E_{hi}(\gamma_i)$. Let f_{low} be the control policy which results in E_{low}. Let $f_{x_0,X_i^{VL}}^{X_j^{VL}}$ be f_{low} extended to the set X^{VL} and then restricted to the state set X_i^{VL+} (see Definition 3.1). The reachable sets within X_i^{VL+} for $f_{x_0,X_i^{VL}}^{X_j^{VL}}$ provide the necessary sets $R_{x_0,X_i^{VL}}^{X_j^{VL}}$ for the (Non-blocking) IBC definition part (i) (see Definition 3.4).

Part (ii) of the (non-blocking) IBC condition follows trivially as every state X^{VL} is a goal state, so within each goal block $X_m^{\pi_{VL}}$ on, the set X_i^{VL+} is the required set $R_{X_i^{VL},x'}^{Q_m}$, where x' is the unique in-set state in block X_i^{VL}. ∎

Figure 2.12 shows the relationship between the two consistency conditions under the assumptions of Theorem 5.2). The many-to-one map Θ_π^{hi} is shown to emphasise that the original high-level observations can be recaptured from the transition labels in the VL partition automaton.

Theorem 5.2 extends naturally to systems with three or more layers since hierarchical consistency and the (non-blocking) IBC property are transitive. The VL algorithm can be performed either top-down or bottom-up (i.e. apply the VL algorithm to the high-level, then to the low-level, or vice-versa) leading to the same result.

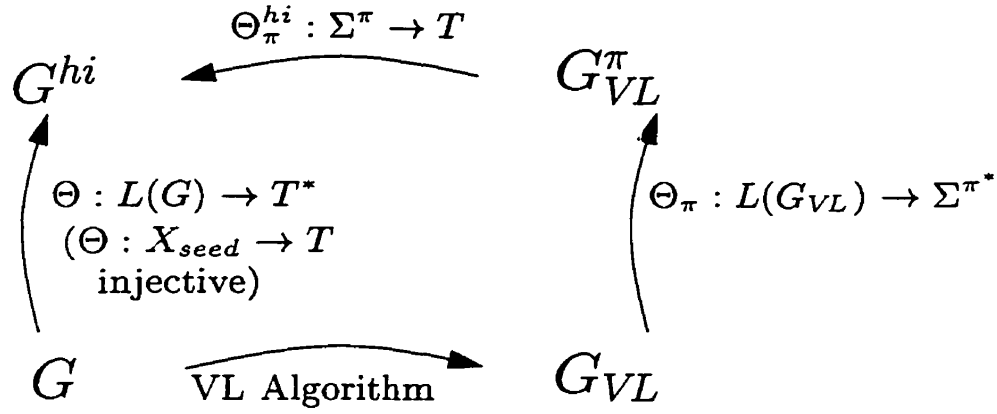


FIGURE 2.12. Hierarchical consistency on the left implies (non-blocking)-IBC on the right when $\overline{L_m(G)}$ is controllable (Theorem 5.2).

CHAPTER 3

Trace-DC Hierarchical Supervisory Control: Examples and Applications

---

1. Illustration of the Formation of IBC Partitions

In many applications, the overall system is composed of subsystems that act independently and are only synchronised or co-ordinated via the application of control. In the special case of non-interacting subsystem models, Non-blocking IBC partitions of the overall system space can be built-up from partitions in the individual components.

The lack of interaction between subsystems can be captured by the lack of overlap between the event alphabets of the components. Consider two automata G_1 and G_2 with alphabets Σ_1 and Σ_2 such that $\Sigma_1 \cap \Sigma_2 = \emptyset$. Figure 3.1 shows an example of such a case along with a recogniser for the shuffle product $L(G_1)||L(G_2)$ labelled $G_1||G_2$ (in which the event labels have been suppressed for convenience).

This example is the standard "cat and mouse" scenario taken from [**78**]. The problem context may be summarised as follows (the reader is referred to [**78**] for a full discussion):

- The states represent the location of agents (cat and mouse, perhaps) in one of 5 numbered rooms. The transitions represent doors between rooms that are all blockable (i.e. controllable) except for c_7.
- The control problem is to block appropriate doors in order to achieve the control objectives of:
  [1] not allowing the agents to occupy the same room ("cat eats mouse"),
  [2] always ensuring accessibility to the start state (rooms (2) and (4)), and

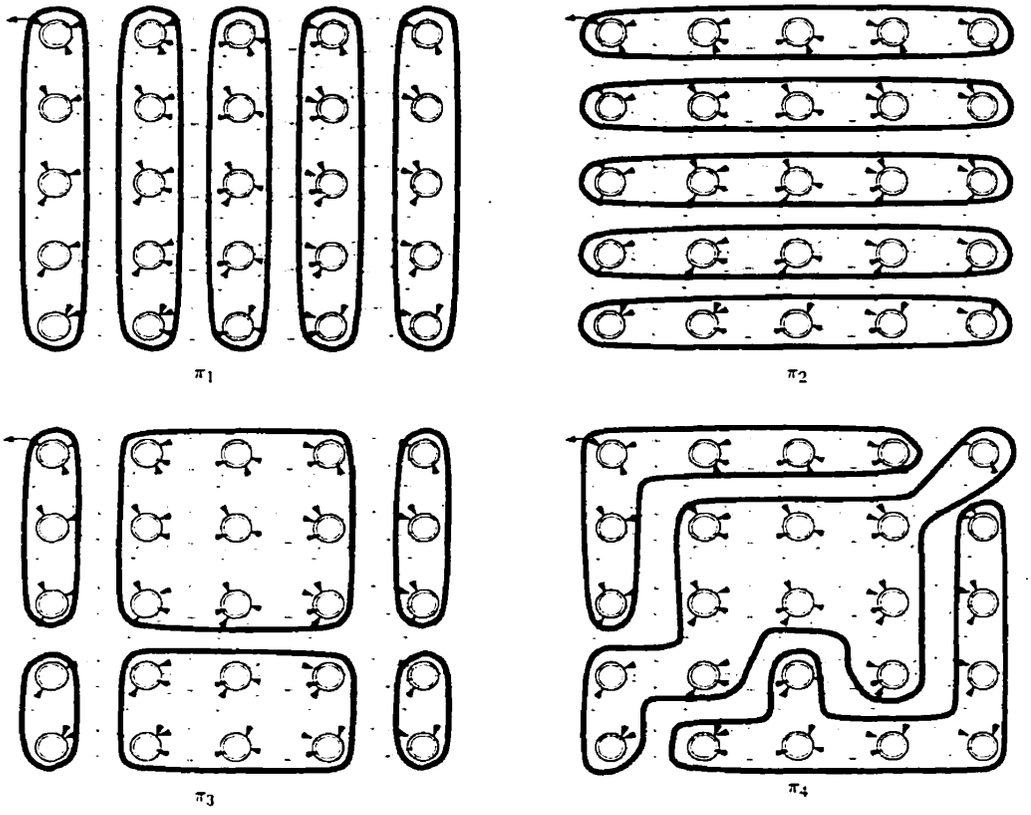


FIGURE 3.2. IBC partitions for $G_1||G_2$.

substitution property [37], namely: for any two states z and z' in $X_1 \times X_2$ and for any transition label $\sigma \in \Sigma$, $\quad (z =_{\pi_i} z') \implies (\delta(z,\sigma) =_{\pi_i} \delta(z',\sigma)), \quad i = 1,2.$

The partition π_3 in Figure 3.2 illustrates the more general fact that (non-blocking) IBC partitions π_{G_1} and π_{G_2} of the component state spaces X_1 and X_2 can be combined to form a (non-blocking) IBC partition of $X_1 \times X_2$ via the equivalence relation

$$(x,y) =_{\pi_{G_1 \times G_2}} (x',y') \quad \iff \quad \{(x =_{\pi_{G_1}} x') \wedge (y =_{\pi_{G_2}} y')\}.$$

in this manner, the partitions $\pi_{G_1} = \{\{(2)\},\{(1),(0),(3)\},\{(4)\}\}$ and $\pi_{G_2} = \{\{(4),(3),(0)\},\{(1),(2)\}\}$ yield the partition π_3. Note that π_1 and π_2 are also cases of this phenomenon with $\pi_{G_1} = \pi_{id}, \pi_{G_2} = \{X_2\}$ and $\pi_{G_1} = \{X_1\}, \pi_{G_2} = \pi_{id}$ respectively.

The partition π_4 is an illustration that not all (non-blocking) IBC partitions may be decomposed into component partitions. π_4 also highlights the conceptual reward of a hierarchical description. It can be checked that the top left block in π_4 is the maximal controllable state set which satisfies the original control objectives. The bottom right block represents a set of states that are "safe" (i.e. states which do not

allow access via uncontrollable events to a state in which both agents occupy the same room), but are disallowed because they are not co-accessible to the start state without travelling through unsafe states. Possible extensions are immediately apparent that would make the bottom right block safely co-accessible to the start state; for instance, put an attendant in room (4) to separate the agents. This would allow a safe trajectory between the two high level blocks and hence permit considerably more freedom to the agents. This argument illustrates the relative ease with which one can reason about complex systems when presented with a hierarchical decomposition.

For the case of systems that do interact, i.e. when either $\Sigma_1 \cap \Sigma_2 \neq \emptyset$ or a synchronisation constraint is used (see Chapter 4), the IBC condition is in general no longer preserved when component partitions are combined.

2. Manufacturing Systems

In this section, the hierarchical decomposition and control of manufacturing plants will be examined. Several examples will be presented and all are based on the models in Figure 3.3 for machines, buffers and testing units. The states I, W, E and F stand for "idle", "working", "empty" and "full", respectively. By assumption, the machines and testing units may be disabled from starting a task, but may not be disabled from finishing. These models form a set of primitives $\mathcal{G}_{primitives} = \{M, B, TU\}$.

The *plant layout* for a manufacturing plant is defined as follows.

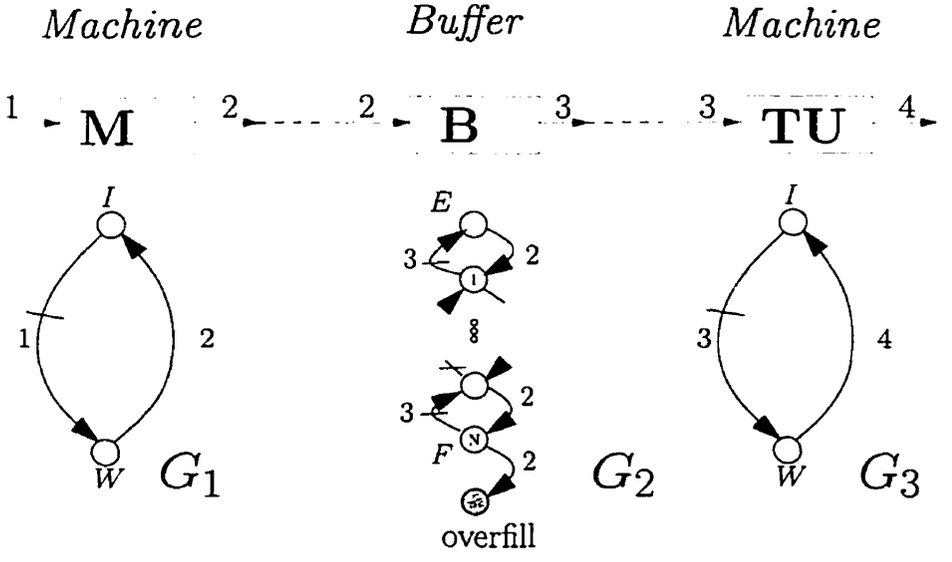


FIGURE 3.3. The machine, buffer and testing unit models.

Definition 2.1. Manufacturing Plant Layout

Let $\mathcal{G} = \{G_i, i = 1, ..., N\}$ be a finite set of automata with $G_i \in \mathcal{G}_{primitives}$ for each i. Let $\Sigma = \bigcup_i \Sigma_i$ be the union of all alphabets. A *manufacturing plant layout* is defined as a 3-tuple,

$$(\mathcal{G}, \tilde{\delta}_{layout}, \Sigma)$$

where $\tilde{\delta}_{layout} \subseteq \{\mathcal{G} \cup S\} \times \Sigma \times \{\mathcal{G} \cup S\}$ in which S is the set $\{S_\gamma, \gamma \in \Sigma\}$ where each S_γ is an automaton with alphabet $\{\gamma\}$ accepting the language γ^*. Each event in $\tilde{\delta}_{layout}$ is one of the following forms:

$$\begin{array}{lll} [G_i, \gamma, G_j], & \gamma \in [\Sigma_{ic} \cap \Sigma_{jc}] \bigcup [\Sigma_{iu} \cap \Sigma_{ju}], & \text{(an internal move)} \\ [S_\gamma, \gamma, G_i], & \gamma \in \Sigma_i, & \text{(introduction of a piece)} \\ [G_i, \gamma, S_\gamma], & \gamma \in \Sigma_i & \text{(removal of a piece).} \end{array}$$

□

The interpretation of the layout is that the events in $\tilde{\delta}_{layout}$ represent physical connections (pathways for pieces) between primitive units (in this case machines, buffers and testing units). The event label captures the synchronisation requirement for the removal of a piece from one model and the corresponding addition of a piece in another. It is assumed that these labels are either controllable in the alphabets of both systems involved, or uncontrollable in both systems. The introduction of pieces from outside the layout, and their removal, is captured by events to and from the automata in S respectively, with the convention that the automata S_γ are not explicitly illustrated on the layout.

The dashed lines in Figure 3.3 show an illustration of a plant layout. Hence the machine M and the buffer B share the event label "2" in their alphabets. An automaton capturing the complete dynamics can be constructed by taking the synchronous product (see Chapter 4, Equation 2.4.1 for the definition) of the primitive models, with event labels assigned appropriately from the layout. Note that $S_\gamma || G_i = G_i$ if $\gamma \in \Sigma_i$, so the synchronous product $S_1||G_1||G_2||G_3||S_4$ for the Machine-Buffer-Testing Unit sequence in Figure 3.3 reduces to $G_1||G_2||G_3$.

Several layouts will now examined. The *control objective* will be the same for each layout considered. This is to avoid overfilling the buffers while

A) maintaining reachability to the "empty" state, i.e. where all buffers are empty and all machines idle and
B) allowing maximally permissive use of the machines.

This objective can be re-stated as a non-blocking accessibility problem by creating a "dump" state to which all overfill events lead, and from which there are no exiting transitions, i.e. the empty state (the goal) is not accessible from the "dump" state. For illustrative purposes, all overfill states (those shaded in Figure 3.3) will often be condensed to a single "dump" state, as the dynamics, once an overfill event may have occurred, are considered inconsequential.

The intention is for the resulting controls to represent the first layer, designed for safety, of a control architecture. Additional control action may be applied, in combination with the control for safety, in order to optimise for throughput, minimise time, etc. The additional control action could be applied with a forced-event style control, in which case the inhibitions from the safety-supervisor would override the forced-event control. Alternatively, additional control could be applied as further inhibition of events that are permitted by the safety-supervisor.

The examples have been simulated, and data files can be found at [**1**] for software in the formats of [**5**] and [**6**].

2.1. An Illustrative Example: Transfer Line with Re-entrant Flow

Consider the layout for a transfer line with re-entrant flow shown in Figure 3.4 (an extension to an example in [**103, 29**]). Each workpiece must be processed by all the machines, and upon testing by the testing unit, may be accepted or rejected, the latter resulting in another pass through machines M2 and M3.

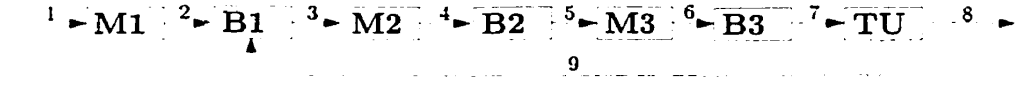


FIGURE 3.4. A material transfer line with re-entrant flow.

The state in which all buffers are empty and all machines and testing units are idle is identified as both the initial and goal state. This system has 129 states. A portion of the automaton for the complete dynamics is displayed in Figure 3.5. A natural partition based on the number of active pieces is also displayed for this portion of the automaton. It can be verified that this partition satisfies the *non-blocking* IBC condition in Definition 3.4. The π-automaton for this partition is illustrated in Figure 3.6. G^{π} is a *non-blocking* IBC partition automaton.

The main result of Section 4 states that the flow at the level of the π-automaton can be realized in the low-level system via the IBC Synthesis Algorithm (see Section 4). This can be illustrated for this example for the control within the block X_4.

Let the π-level supervisor be the shaded blocks in Figure 3.6. Now consider the action of this control at the block X_2.

$$H^{\pi}(X_4) = \{\Sigma_u^{\pi} \cup U_4^3\}$$

i.e. the control applied at the block X_4 is to enable only U_4^3 and hence force the state to X_3. In general, the π-level control may be history dependent, but for illustrative purposes, a state-feedback supervisor is considered.

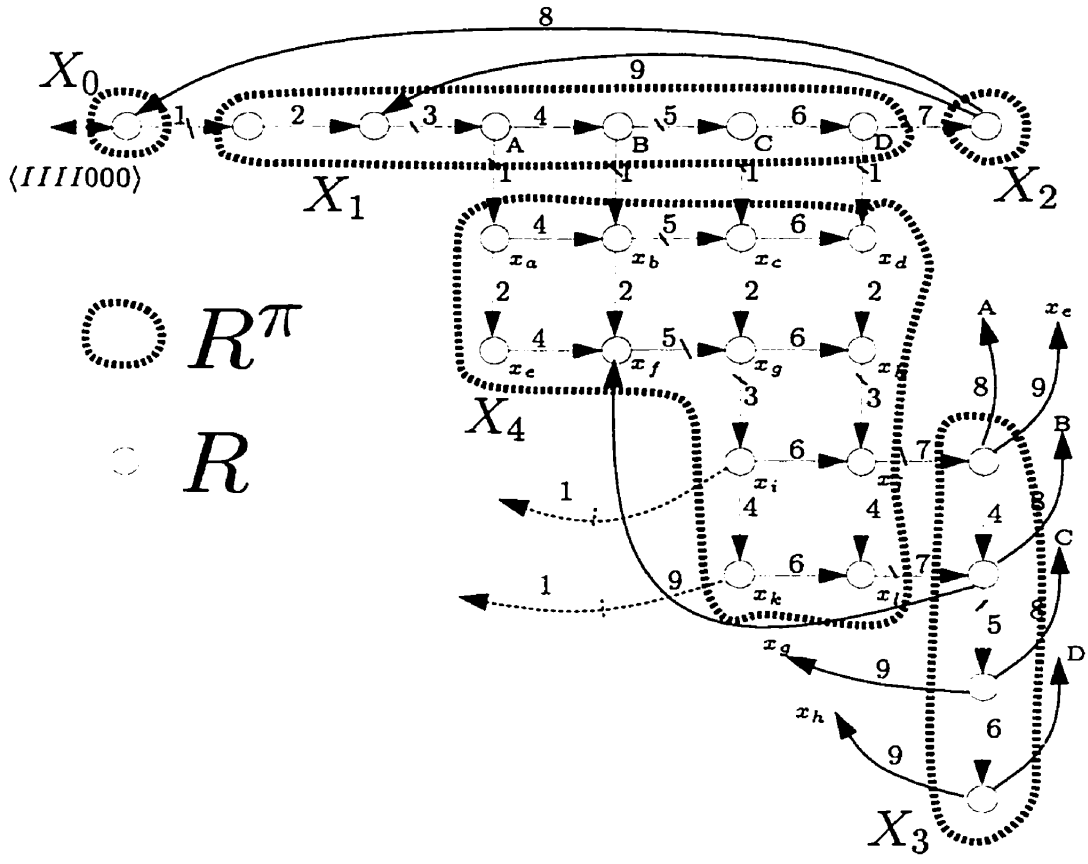


FIGURE 3.5. A controllable state set and partition with a typical inhibited undesirable event 1 (all others suppressed for clarity).

To achieve the necessary π-level control, the following low-level control, h_{low} would be be synthesised by the IBC synthesis algorithm (with uncontrollable events that are not defined at the given state suppressed for clarity).

$$\begin{aligned} h_{low}: \quad & x_a \longmapsto \{2,4\}, \quad x_b \longmapsto \{2,5\}, \quad x_c \longmapsto \{2,6\}, \quad x_d \longmapsto \{2\}, \\ & x_e \longmapsto \{4\}, \quad x_f \longmapsto \{5\}, \quad x_g \longmapsto \{3,6\}, \quad x_h \longmapsto \{3\}, \\ & x_i \longmapsto \{4,6\}, \quad x_j \longmapsto \{4,7\}, \quad x_k \longmapsto \{6\}, \quad x_l \longmapsto \{7\} \end{aligned}$$

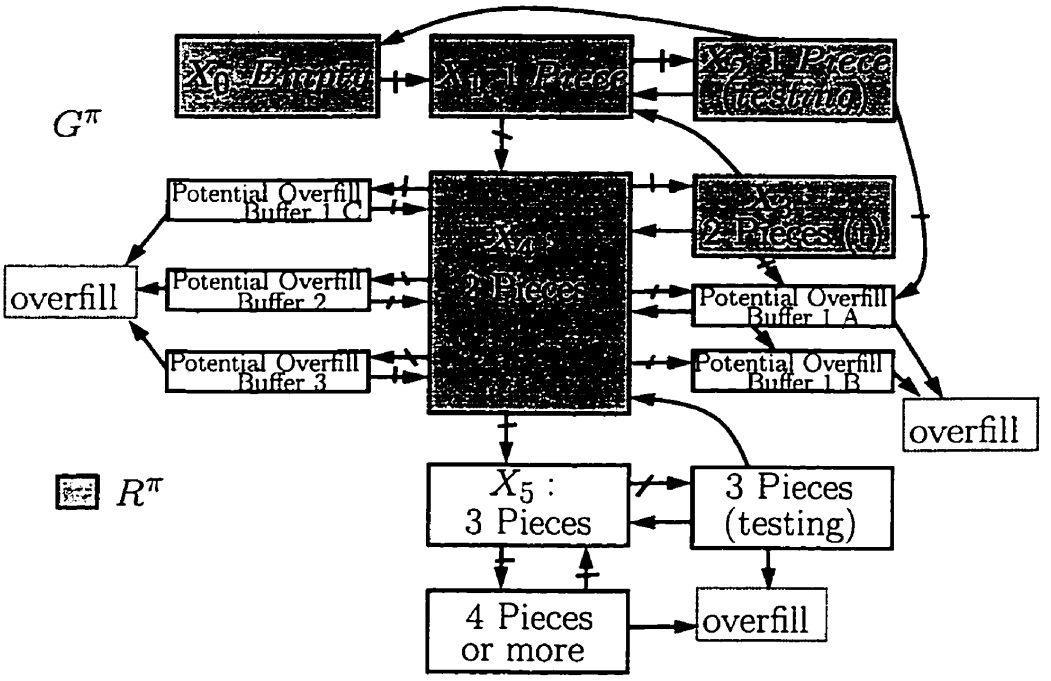


FIGURE 3.6. A π-level controllable state-set.

Notice, for instance, that the low-level control h_{low} inhibits the event 1 at x_i to prevent a transition to the block X_5 because U_4^5 is inhibited at the π-level.

All control actions of H^π can similarly be instantiated at the low-level illustrating Corollary 4.2 by providing a state set R with required image R^π. In this specific example, R^π is the maximal controllable state-set satisfying the objectives, though maximality is not required for Corollary 4.2 to hold and hence all controllable state-sets could be instantiated in this fashion.

2.2. A Double Queue

Consider the two-stage queue in Figure 3.7, with buffer size N for both buffers. As a preliminary step, the first machine-buffer-machine sequence M1-B1-M2 is analysed independently (see the top left of Figure 3.8). The state space ($4N$ states) for the M1-B1-M2 portion is shown at the right in Figure 3.8 with a (non-blocking) IBC partition. The corresponding partition automaton is shown on the left. The labelling refers to the state of the machines and buffers: "0" is used for "empty", "N" for "full" and "N+1" for "overfill" (e.g. in the state "*IkW*", M1 is idle, there are k pieces B2 and M2 is working). The overfill states and their images at the higher levels are shaded. Note that adjacent blocks in Figure 3.8 can be amalgamated to form larger (non-blocking) IBC blocks.

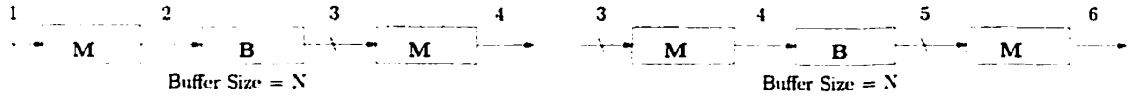


FIGURE 3.7. A two buffer queue ("double queue").

The automaton for the double queue is formed by taking the synchronous product of the low-level automaton in Figure 3.8 with its counterpart for the second portion of the queue. The shared events (3 and 4) force the second machine in the first portion to be in the same state as that of the first machine in the second portion, hence the total count of the reachable state space is the expected $4N \times 4N/2 = 8N^2$. A partition automaton, G_{π_1} for this system is displayed in Figure 3.9 for buffer size $N = 1$ and in Figure 3.10 in the general case.

Finally, a partition of the π_1-level state-set is also presented in Figure 3.10, which leads to a third level, G_{π_2} in the hierarchical layering. The partition π_2 has a natural description at the base level since each diagonal band has the same number of active pieces within a margin of 2.

The utility of the theory is exhibited by the fact that the control specification, i.e. that of not allowing reachability of the overfill states, can be stated and solved for through common sense reasoning at the aggregate level; for instance, in the case of $N = 1$ a reasonable evolution which is clearly safe is the language $K^{\pi} = U_1^2(U_2^3U_3^4U_4^2)^*U_2^4U_4^1$. The control can then be translated down a chain of increasingly refined partitions to the full system model in a straightforward and sound manner.

2.3. Join and Split Layouts

Analyses of other formats for the machine-buffer connections can be made in a similar manner. Figures 3.11 and 3.12 show a *join* and *split* respectively for the buffer size $N = 1$ case. These can be extended similarly to the N sized buffer case. A possible goal in this work is the emergence of primitives that would allow for the immediate description of control methodologies for arbitrary plant layouts.

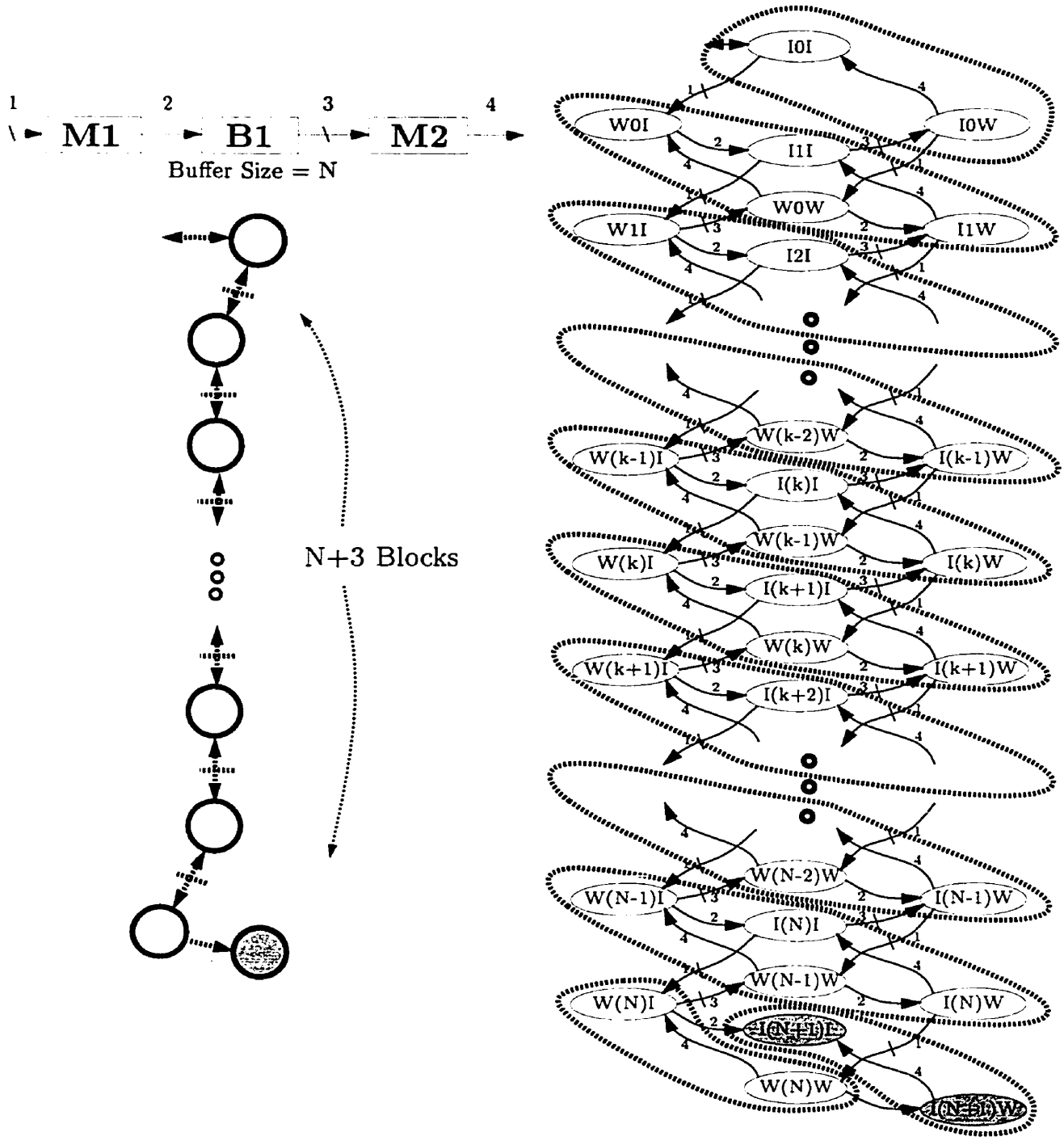


FIGURE 3.8. State space and partition for first portion of double queue.

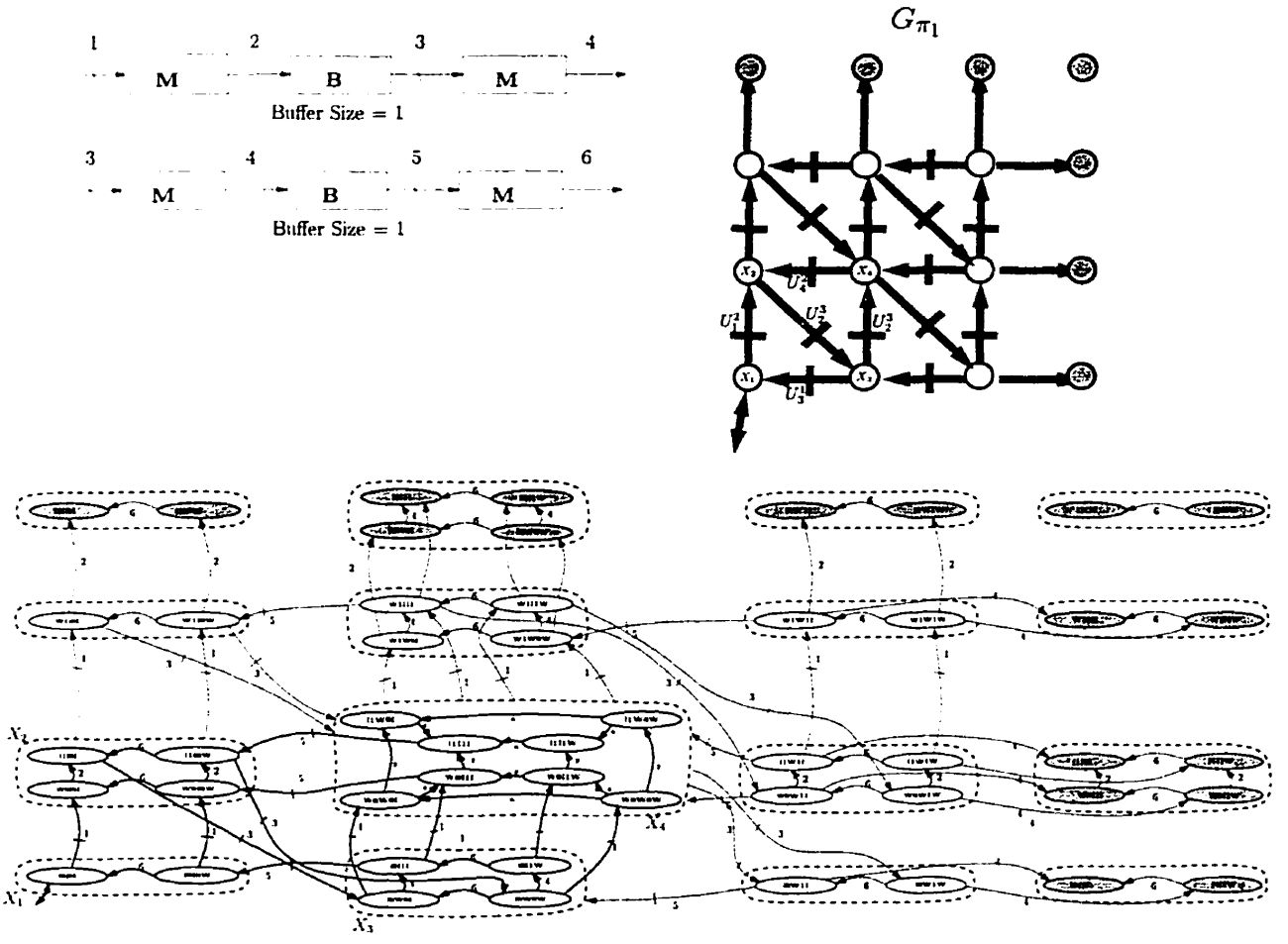


FIGURE 3.9. The double queue with buffer size $N = 1$.

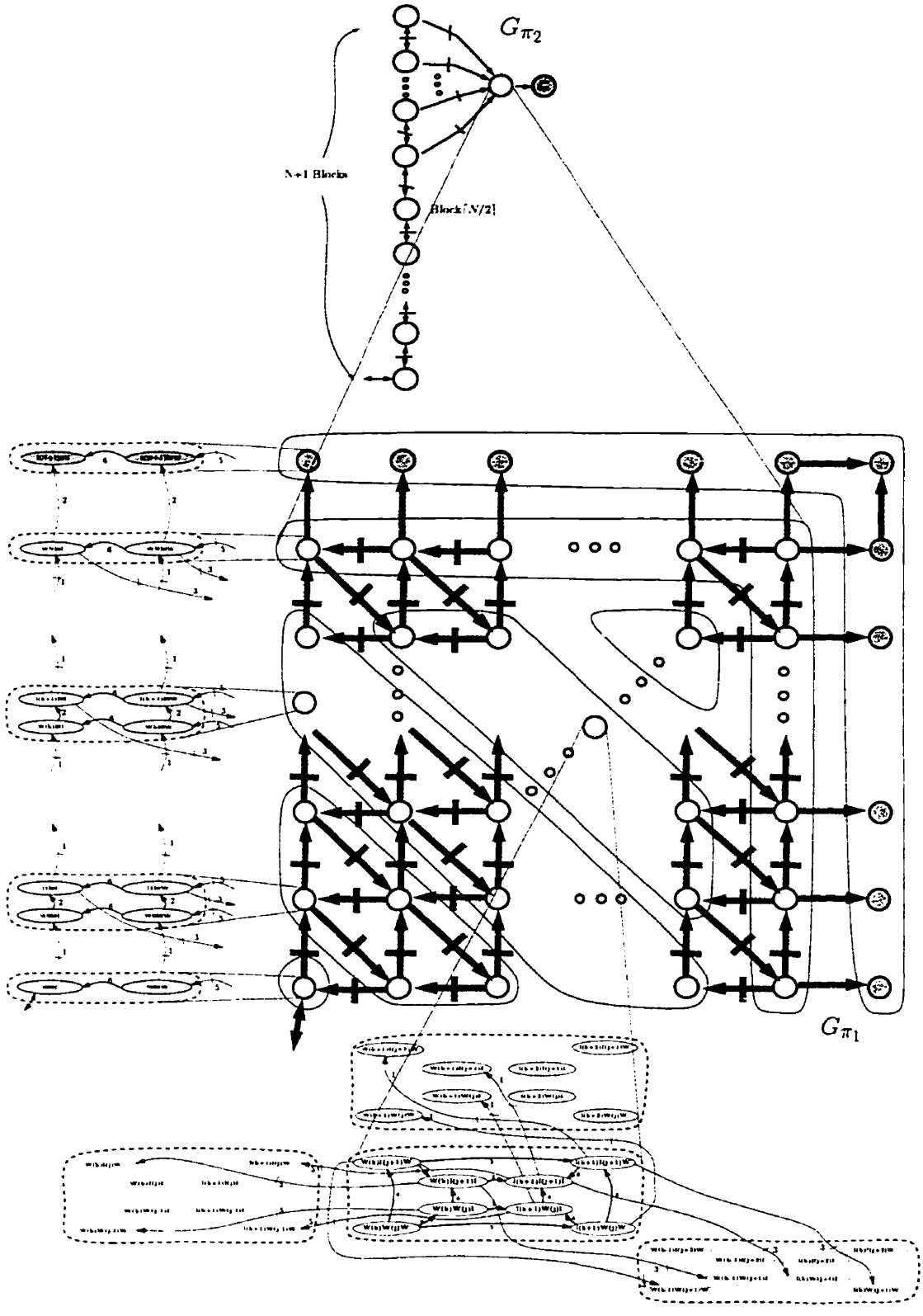


FIGURE 3.10. Three levels of hierarchy for the double queue.

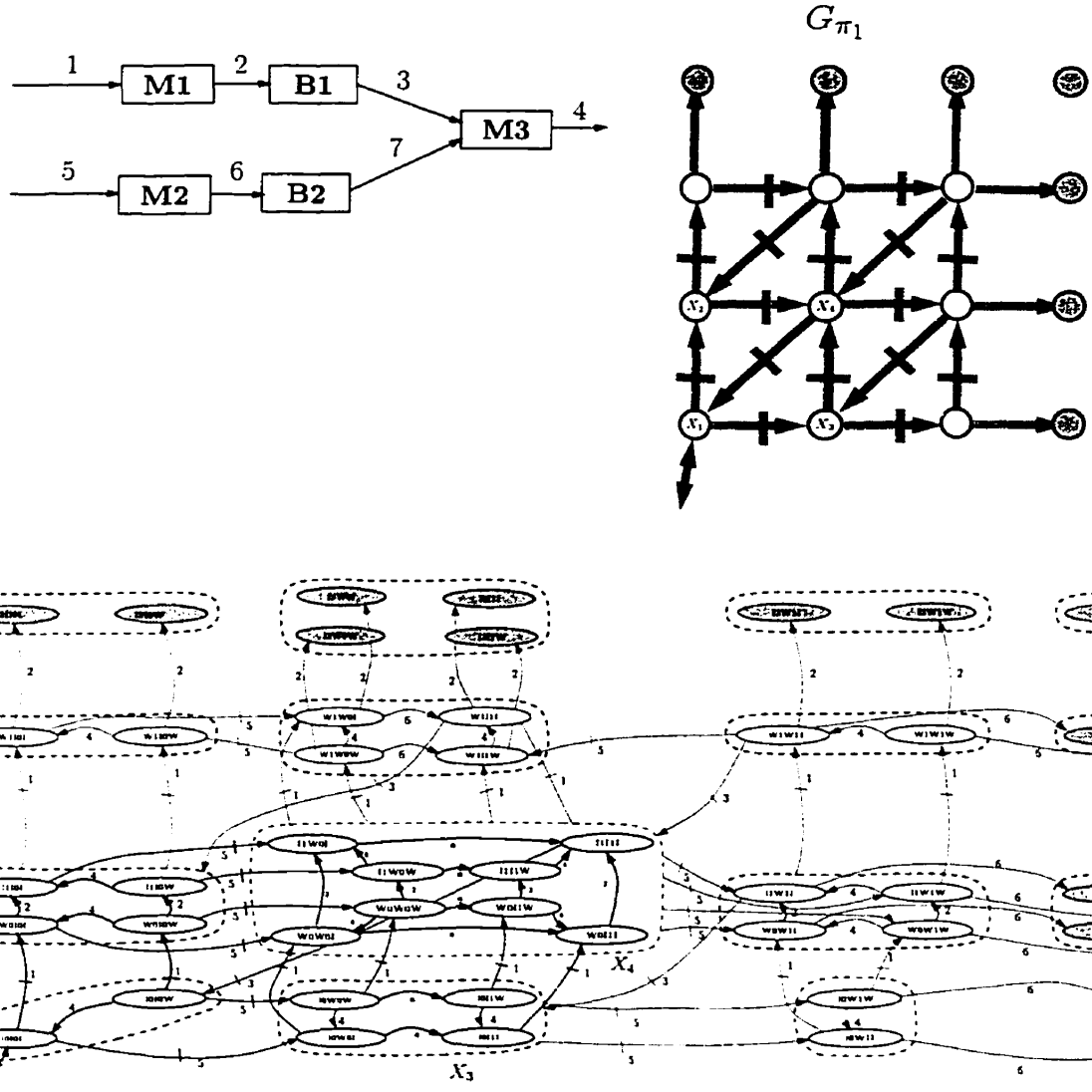


FIGURE 3.11. A join element with buffer size $N = 1$.

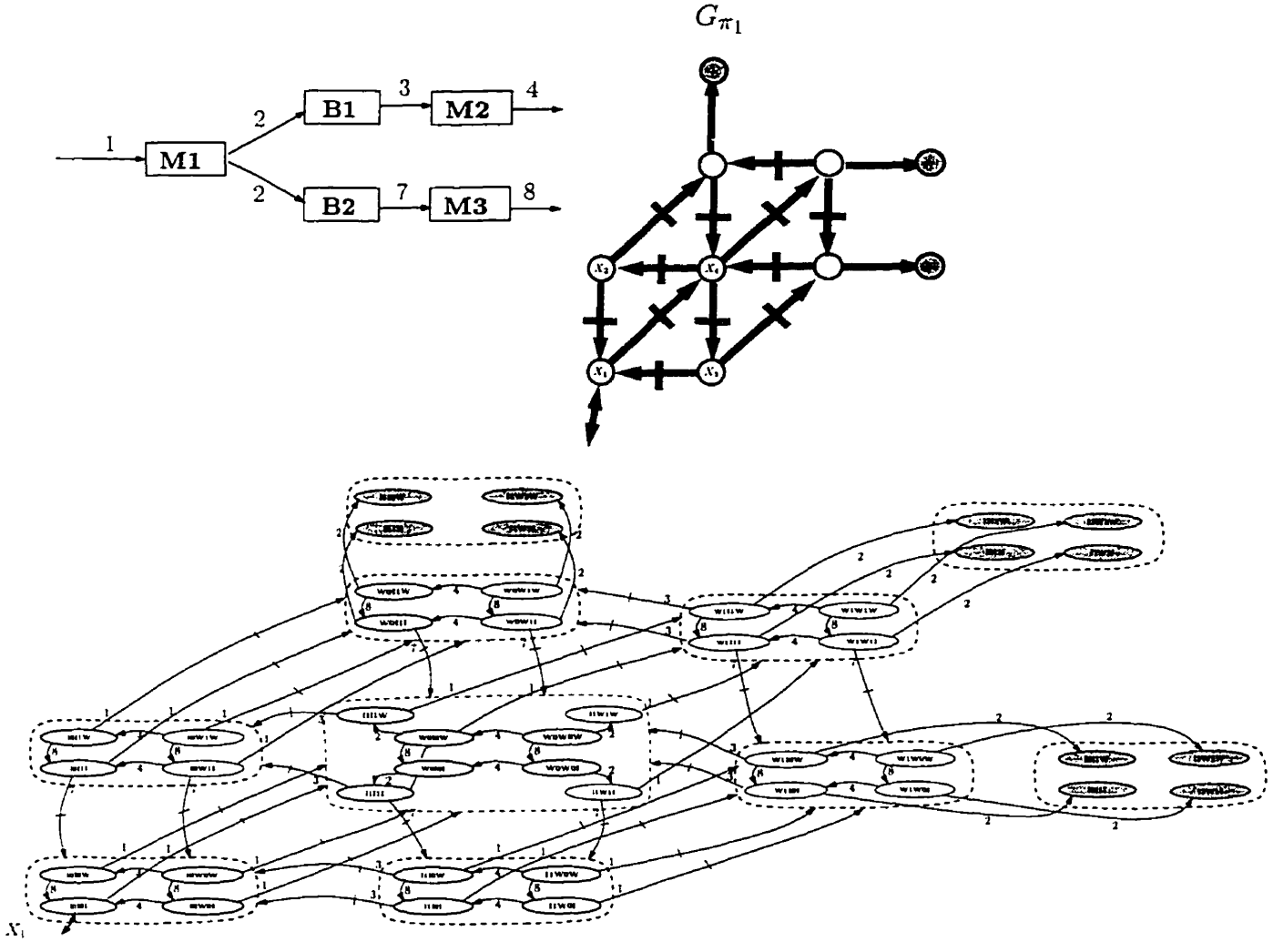


FIGURE 3.12. A split element with buffer size $N = 1$.

3. Embedded Control of Appliances at General Electric R&D

This section summarises a preliminary investigation of DES models for home appliances. This work was performed by the candidate as part of a joint industry-university collaboration with General Electric Corporate R & D in Schenectady, NY. The goal of the work and this section is to illustrate a possible application of DES control, rather than prototype an implementable appliance control scheme.

3.1. Background

Historically, most home appliance control applications are performed with electro-mechanical controls. The controls took the form of rotating disks (cams) to control wash and dry cycles and could be regarded as a form of open loop control. Dangerous failure conditions were and still are identified by performing a Failure Modes Effect Analysis (FMEA). Safety features are included in the appliance by the use of hard-wired electrical overrides or built in mechanical fail-safes.

With the addition of micro-processors to modern appliance control, there is now the possibility of extensive branching within the nominal cycle as information regarding the status of the appliance is fed back to the controlling device (see [**9**] for a discussion of home appliance "smarts"). In particular, measurements related to safety as well as measurements of performance are available and hence safety features, such as failure diagnosis, may be included within the framework of the nominal control.

There are several benefits to this. One is the wider range of actions available to correct for failures and other unplanned events when the nominal control and safety actions are designed together. Another is that methods may be developed to formally verify (within the context of the model of the appliance), that corrective actions do indeed lead to safe configurations.

3.2. DES Washing Machine Models and a Supervisory Control Problem

There is extensive literature on the continuous processes involved in washing and drying fabrics [**61, 87, 27**] but, as yet, there are no discrete event models for appliances as a whole. The intention in the use of a discrete event model is to subsume the continuous underlying process dynamics in favour of purely discrete dynamics. In some cases, such as a valve or pump, this is straightforward; the valve may open or close and the pump may turn off or turn on. In other cases, the abstraction is arbitrary. For instance, the continuous rise in water level might be modelled by a transition from a *tub-empty* state to a *tub-full* state.

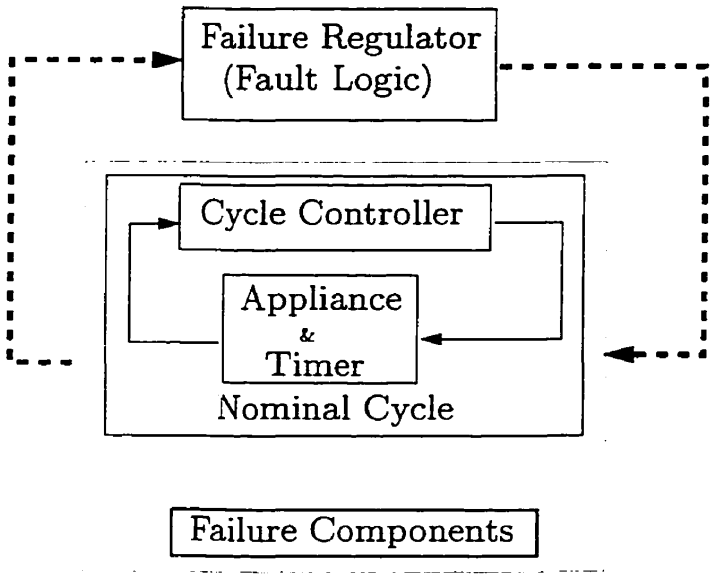


FIGURE 3.13. A coarse system architecture for fault regulation.

Figure 3.13 shows a coarse system-level architecture for appliance control. Conceptually, the nominal cycle can be thought of as a single path through the product space of the appliance and the cycle controller (see Figure 3.14 for an illustration). A failure manifests itself as an exogenous transition off the nominal cycle in an extended < nominal cycle, failure modes > state space. When a failure occurs (e.g. a valve is stuck open), the failure regulator must drive the system to a safe region. This may be a continuation of the nominal cycle if the fault is recoverable but will more often be a termination of the cycle, and, in this case, the washer is simply turned off. Note that both 3.14 and 3.15 are purely illustrative and refer to neither a running simulation nor a usable model for the appliance.

In Figure 3.13, the combination of failure modes and a nominal cycle result in a language of possible evolutions, each an interleaving of cycle events, failures events and recovery actions. The design of the failure regulator can be cast in the supervisory control framework. Direct control events, such as "turn on pump" are controllable. Exogenous events such as the transition to a *tub-full* state mentioned above and failure events are uncontrollable The control problem is then:

- Find a *supervisor* (Failure regulator), that allows maximal use of the appliance while not allowing access to states that exhibit dangerous failure conditions (as determined through the FMEA).

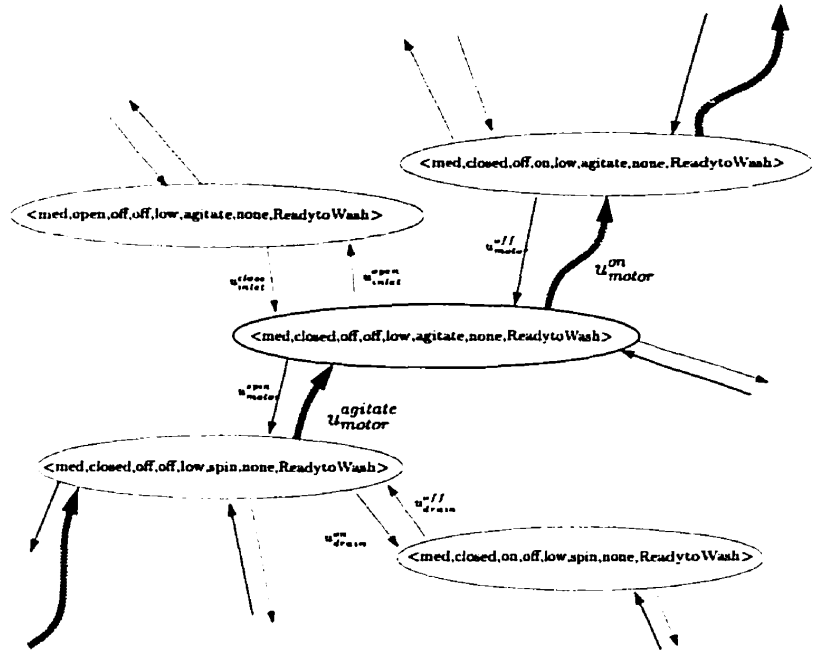


FIGURE 3.14. A path through the state space.

The illustration in Figure 3.15 shows an example of the avoidance of a "spin when full" state. When the "pump stuck" event occurs, the appliance must be inhibited from entering the spin cycle.

3.3. Simulation of the Nominal Wash Cycle

The Matlab package **Stateflow** [2] was used as a simulation testbed for a DES appliance model. It must be noted immediately that Matlab provides a deterministic single sample path evolution which is in direct contrast to supervisory models in which *multiple* sample paths (i.e. the strings in the behaviour language) are usually considered. Despite this, and as a first approach, a purely discrete nominal cycle with several failure modes was developed with the Stateflow software. This is illustrated in Figure 3.17, with the Simulink top level in Figure 3.16. A printout of the "*explore*" interface window with a list of states and events is shown in Figure 3.18. The state space for the appliance model is a 5 tuple ; the *Inlet_Valve*, the *Drain_Pump*, the *Clutch_Position*,the *Motor_Status* and the *Tub_level*. The possible states for each system component and the *Controller* are listed at the left in the "*explore*" window under the respective component name. The events are also listed at the right of the *explore* window.

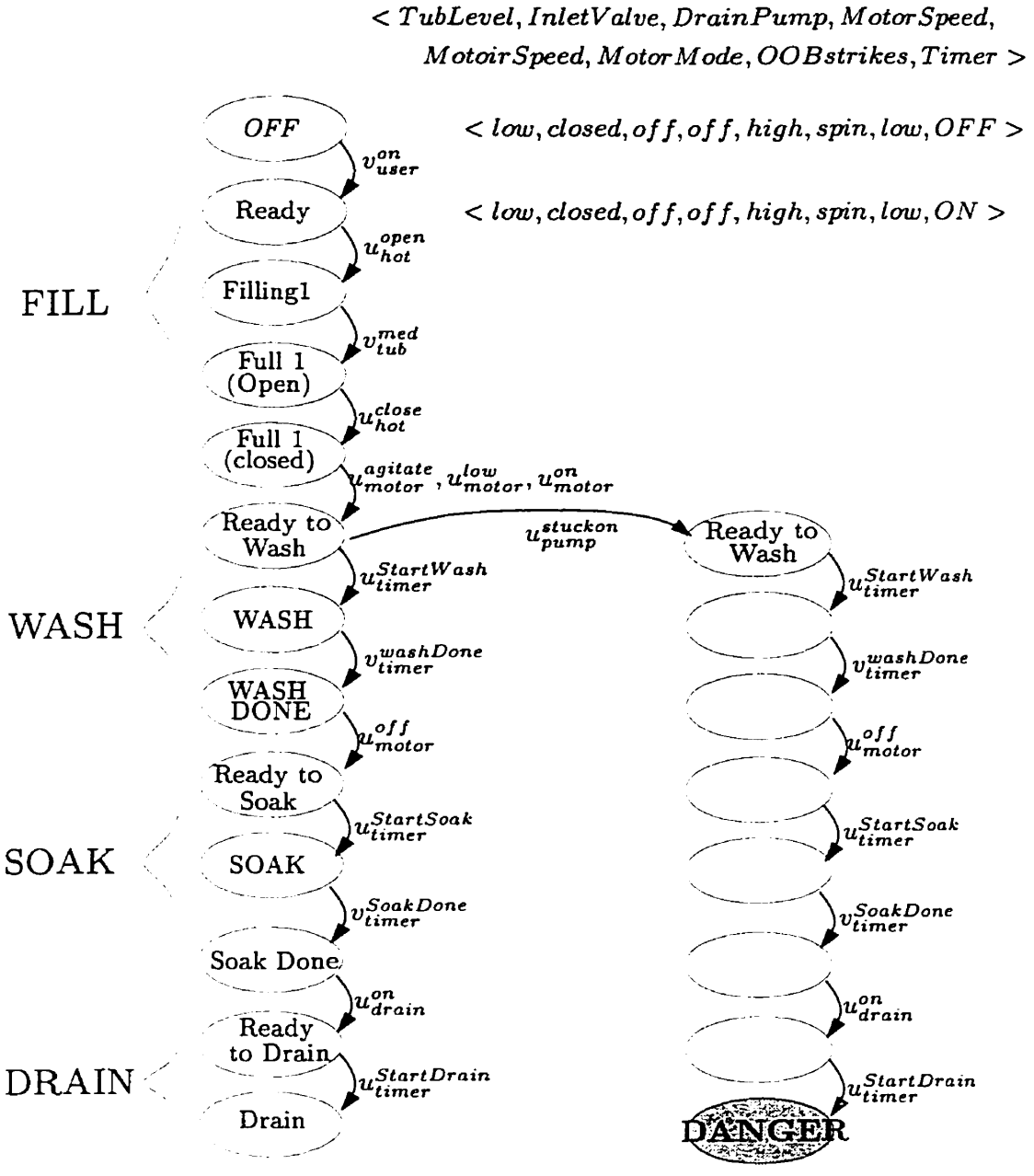


FIGURE 3.15. A dangerous blocked path from the nominal cycle.

The reader is referred to the Matlab Stateflow manual for a full description of the evolution of the charts, but a partial illustration of the event broadcast methodology follows. The majority of the events in the simulation reside at the level (or *scope* on the *Explore* window) of the *Washer* block. That is to say, when these events are broadcast, only blocks below (i.e. graphically inside) the *Washer* block will be

evaluated. The events that do not reside at this level are the input data events from the simulink environment which include a *time_base* event (indicating the passage of a non-zero finite time slice) and the failure events (set to occur at any user-selected time). The *Explore* window shows all the events and their scope.

The following is an example of the evolution of the chart. Within the *Drain_Pump* component, the initial state (indicated by a source-less arrow in the top left) is the *Off* state. When the event *turn_pump_on* is broadcast, the *Drain_Pump* component enacts the arc with label *turn_pump_on*. The effect is to move to the state *Will_turn_on* if *[pump_fail==0]* is true or back to *Off* if *[pump_fail==1]* is true. The *pump_fail* is an external value sampled by the block (see the Simulink Figure 3.16). The *turn_pump_on* event is also broadcast to all other components and these must be updated with respect to this event. In this case, the update has no effect because the event *turn_pump_on* either does not appear, or cannot be immediately enacted within these components. The Stateflow chart *Washer* is then in a stable and fully updated status, and requires exogenous input to update further. The exogenous input comes in the form of the event *time_base* which here represents the start-up time of the pump. Similarly to the *turn_pump_on* event, the *time_base* event is broadcast to each component in turn (the ordering in fact is with respect to position within the interface window, starting from the top left). The only component which is ready to act on the *time_base* event is the *Drain_Pump* which moves to the *On* state, and in so doing, broadcasts the event *pump_on* as part of the broadcast of the event *time_base*. It does this due to the presence of the command *exit:pump_on* within the *Will_turn_on* state.

As this short description illustrates, the evolution has a strong sequential character which, as noted, contrasts the supervisory control framework in which inhibition and evolution are often considered simultaneous. Sequencing within a single run is a difficult issue within the object hierarchy and event broadcast structure of Stateflow. In the final analysis, this sequence performed may be determined by the placement of blocks within the Stateflow visual user interface.

3.4. Verification

As mentioned above, it is the language of all possible sample paths (interleavings of cycle events, failure events and regulator events), that must be exhaustively checked for lack of dangerous conditions in order to verify the safety logic. There are various options for performing this search in Matlab. All of these require multiple simulations. One option is to incorporate noise sources in model the stochastic occurrence of failure events. Multiple runs can the be used to cover some significant portion (in

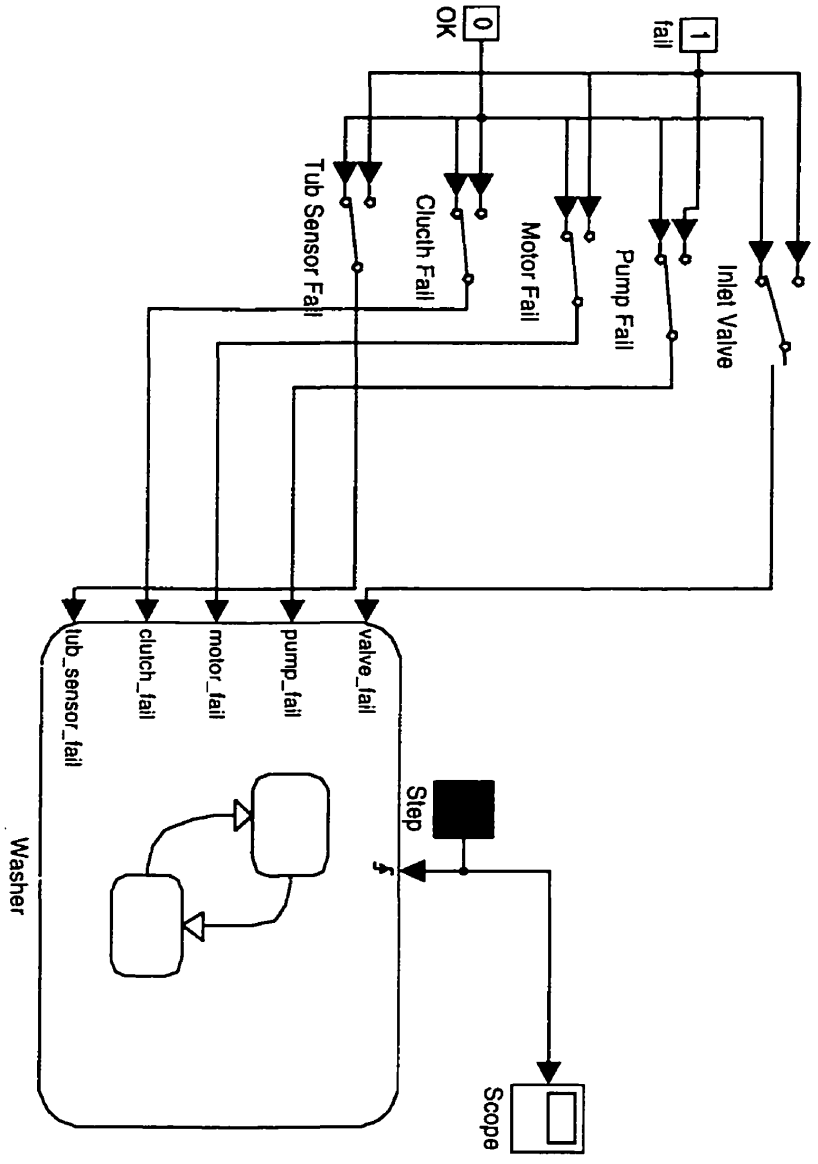


FIGURE 3.16. The Simulink block diagram for a nominal cycle with failures.

a probabilistic sense) of the sample paths. This can be extended to an exhaustive simulation of all failure sequences by sequentially testing all possible failure orderings with respect to the system's evolution.

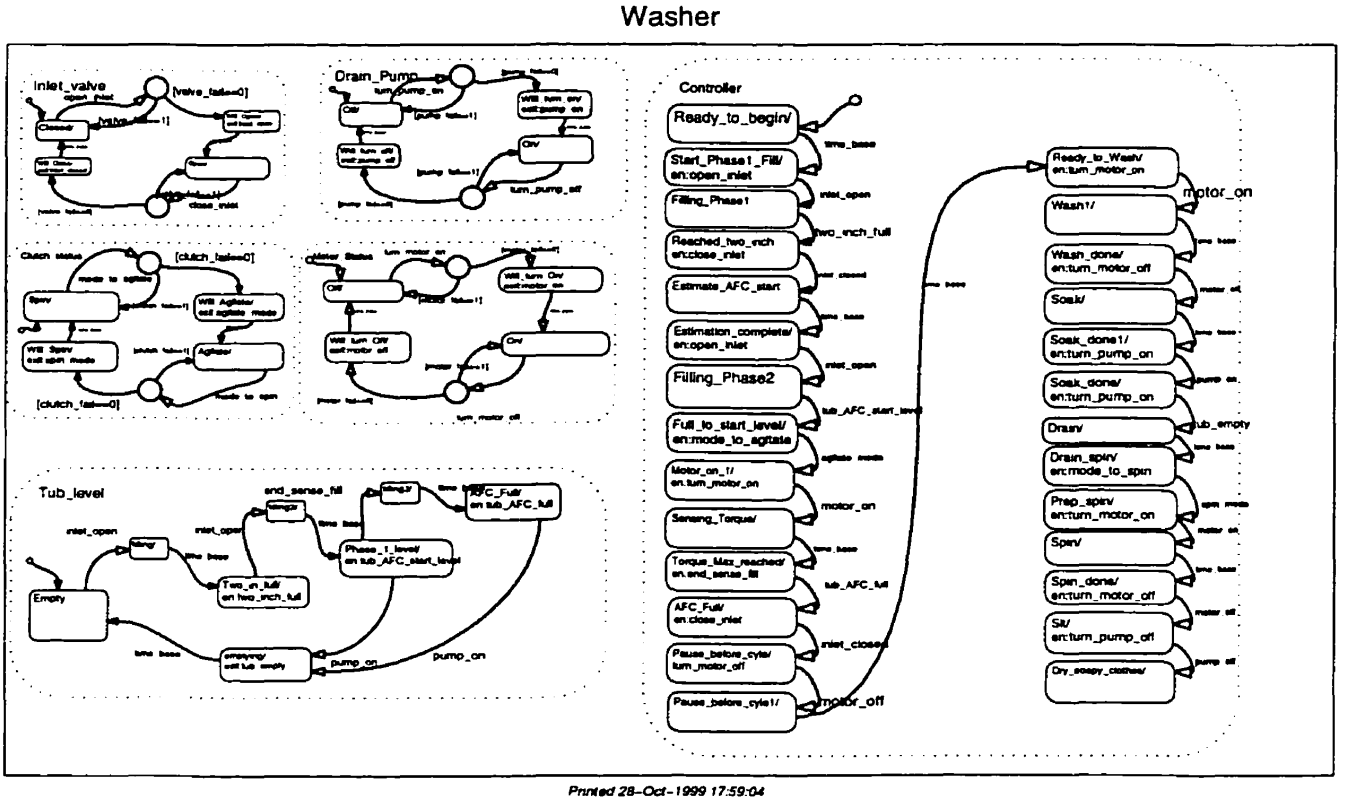
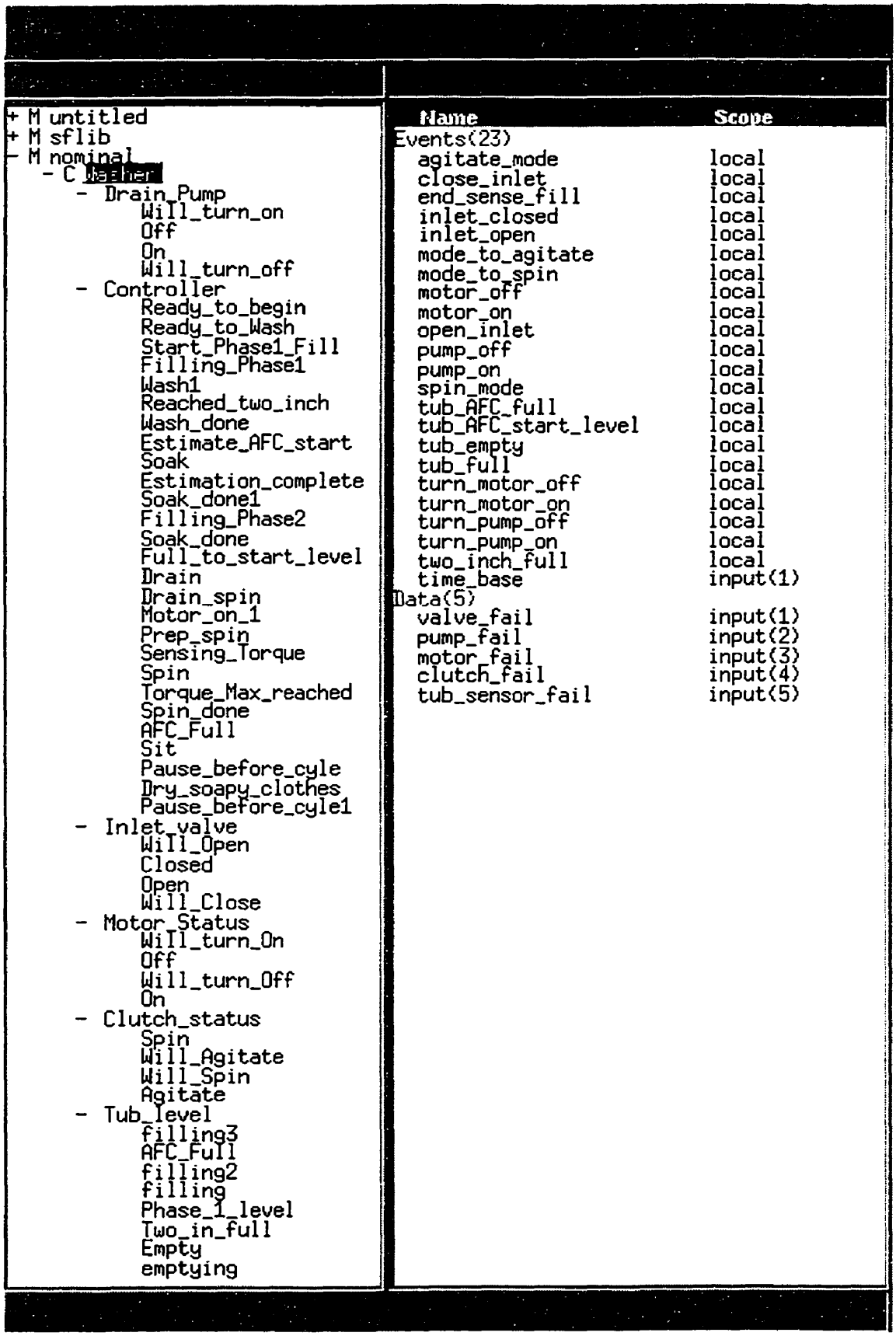


FIGURE 3.17. The StateChart for Stateflow block *Washer*.



58

FIGURE 3.18. The *Explore* window with component state spaces and events.

3.5. Conclusion

The design task is performed iteratively by testing the logic, adapting when it fails, then re-testing. It is this task that is perhaps better suited to the Discrete Event Systems setting and perhaps solved via a maximal controllable sublanguage calculation. It must be concluded that the Stateflow package is not well suited to this purpose and software at sources such as [**6, 5**], written specifically for the supervisory control community, represent better testbeds for this work.

Independent of the software package used, the verification task for a given control policy is computationally intensive and therefore impractical for fine models that have many state components. As a result, proposed future research efforts will concentrate on ways to avoid the verification task, specifically by tailoring the design methodology to synthesise controllers that are already known to satisfy the safety conditions. Thus, hierarchical decompositions such as those described in Chapters 2 and 4 of this thesis are proposed as tools to aid in the design and implementation of these controllers.

CHAPTER 4

Multi-Agent Systems and the Multi-Agent Product

1. Introduction

Systems in the areas of transportation, telecommunications and manufacturing are often represented by networks of interacting agents. Such multi-agent systems are distinguished from classical single agent systems in that both task specifications and cost functions may differ from agent to agent in the cooperative, as well as in the competitive, case. Due to the dynamical interactions between agents, and because of the inherent complexity of many physical networks, the analysis and control of multi-agent network systems often engenders problems of enormous complexity. The objective of this chapter, which closely follows two related papers [**42, 14**], is to introduce a formal system theoretic framework for multi-agent systems and to introduce tractable methods for their analysis, control and optimisation.

Here, we propose a *multi-agent (MA) product* to formulate the simultaneous evolution of two automata models, i.e. the system model G is a specific vector product $G_1||_{MA}G_2$ (see Figure 4.1). This represents a horizontal decomposition in contrast to Chapter 2 in which a vertical hierarchy was presented. A key motivation in this chapter, perhaps to be achieved in future work, is the modular design of a supervisor S when individual supervisors are based on respective system components.

More general settings of the interaction of finite automata exist (see [**44**] and [**37**]), and indeed pertinent and general discussions of agent interaction have appeared in such guises as information structures in epistemic and modal logic [**32**] or game playing models [**98**]. An attempt has been made in the present work to reach a

compromise between the generality of a broad definition of concurrent interaction and the specialisation to specific interaction rules. Several possible interaction rules (e.g. a simultaneous and synchronous vector products) are discussed in order to motivate and aid in the analysis of the multi-agent product.

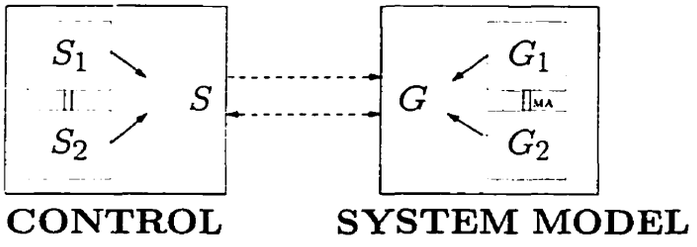


FIGURE 4.1. Modular control of a multi-agent system model.

In the supervisory control literature, the synchronous product is used both as a method to combine component systems and as a model for the interaction between the system and the supervisor (see [**78**], among others). The use of a so-called *independent product* appeared in the analysis of batch process applications in [**92**]. A key difference in the formulation presented here to that of the synchronous product is the notion of *vector event labels* (though work on other vector label systems has appeared in [**55, 56, 57**] discussed below) . In the synchronous product, event strings are interleaved with the interpretation that one component may make a transition while the other is inactive. This represents more than a notational difference. Consider two distinct events, u and v, occurring in P_1 and P_2, respectively. A sequential description, uv, as in the synchronous product, admits sample paths contain other symbols, e.g. w, interleaved with u and v as in uwv. In the simultaneous case, a vector event $\begin{bmatrix} u \\ v \end{bmatrix}$ admits no such interleaving with other events.

This leads to a philosophical stalemate for the issue of real-time feedback control. When simultaneity is modelled through the use of vector symbols, the argument can be made that any two distinct events must occur separated in time by some positive non-zero time δ. Therefore, there is room for a control implementation with response $0 < \epsilon < \delta$ to react between the events (and perhaps even, in a supervisory system, inhibit the second event). Interleaved symbols offer no better solution as the argument can also be made that for a reaction time δ of a control implementation, there can be two events which occur separated in time by only $\epsilon < \delta$. Hence no *feedback* control can take place to prevent their simultaneous evolution. These two views are also differentiated during the modelling phase, by the ordering of (i) the determination of

the sampling period and (ii) the generation of the system model. The view taken in the present work is that in many applications the sampling frequency is limited by a cycle time and so the use of vector symbols is meaningful.

Other usage of vector-event labels is found in [**55, 56**] in which both the system state and transition labels are formed as a vector of integer components, and the controllability and supervisor synthesis issues of supervisory control are addressed. In [**57**], concurrency is added to the model of [**55**] by allowing multiple events to occur simultaneously resulting the synthesis of non-deterministic supervisors.

Our primary motivating example for the MA product is that of multiple agents migrating through an underlying infrastructure or network where, at each time step, all agents must perform an action. "Stays" or "waits" can be defined artificially with self loops. Examples of this include product pieces moving through a flexible manufacturing system, messages through a communication system and individuals through a traffic management system.

The key components in this chapter are i) the development of an algebraic description of the language accepted by a multi-agent product system, and ii) an initial analysis of the supervisory control problem in the context of the multi-agent product. The former is presented in Sections 2 and 3 in which several products, including the *MA product*, are defined and properties such as associativity and distributivity between the various products, are discussed. Section 4 covers the second component and considers supervision of one agent by another, centralised supervision of multiple agents and notions of *non-simultaneous acceptance* of a string and *non-simultaneous controllability* of a vector language.

2. Products of Finite Automata and Regular Languages

Let $G_i = (X_i, \Sigma_i, \delta_i, Q_{0_i}, Q_{m_i})$, $i = 1, 2, .., N$ be N finite deterministic automata, where X_i are discrete state spaces, Σ_i are alphabets (possibly with $\Sigma_i \cap \Sigma_j \neq \emptyset$ for $i \neq j$), δ_i are transition functions and Q_{0_i} and Q_{m_i} are the start sets and goal sets, respectively. For any given run of the systems, the start states x_{0_i} belong to their respective start sets and are identified *a priori*. The generality of a start set is maintained in order to allow for analysis under partial information in future work.

Through much of this section, only the case of the concurrent evolution of *two systems* will be considered rather than an arbitrary finite number. This is done for illustrative purposes. Associativity of the MA product is shown in Section 2.4.

2.1. The MA Product

Definition 2.1. Multi-Agent (MA) Product

$$G_1||_{MA}G_2 \triangleq (X_1 \times X_2, \Sigma_1 \times \Sigma_2, \delta_{MA}, Q_{0_1} \times Q_{0_2}, Q_{m_1} \times Q_{m_2})$$

where,

$$\delta_{MA}(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix}) = \begin{cases} \begin{bmatrix} \delta_1(x,u) \\ \delta_2(y,v) \end{bmatrix} & \text{if } \delta_1(x,u)! \wedge \delta_2(y,v)! \wedge \\ & \quad [\,(u=v) \vee (\neg\delta_2(y,u)! \wedge \neg\delta_1(x,v)!)\,] \\ \text{undefined} & \text{otherwise} \end{cases}$$

□

In the MA product, G_1 and G_2 must both make a transition at every step. If one of the automata has no available transition from its current state, the product will also have no available transitions from its (vector) state. Similarly labelled events are necessarily synchronised when these are defined for both components models. In particular, this rules out a transition $\begin{bmatrix} u \\ v \end{bmatrix}$ at a state $\begin{bmatrix} x \\ y \end{bmatrix}$ if $\delta(x,u)!$ and $\delta(y,u)!$.

For the prefix closed languages $L_1 = L(G_1)$ and $L_2 = L(G_2)$, e.g. when $Q_{m_1} = X_1$ and $Q_{m_2} = X_2$, the language $L(G_1||_{MA}G_2)$ accepted by the multi-agent product system can be defined recursively. More will be said regarding decomposition of the vector language in Section 3.

Definition 2.2. $L_1||_{MA}L_2$

Let L_1 and L_2 be prefix-closed languages over the alphabets $\Sigma_1 \subseteq \Sigma$ and $\Sigma_2 \subseteq \Sigma$ respectively. Define the (vector) product language $L_1||_{MA}L_2$ recursively as follows:

$$\begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix} \in L_1||_{MA}L_2$$

and for $a \in \Sigma_1, b \in \Sigma_2, w \in \Sigma_1^*, v \in \Sigma_2^*$,

$$\begin{bmatrix} w \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \in L_1||_{MA}L_2 \quad \text{if } \begin{bmatrix} w \\ v \end{bmatrix} \in L_1||_{MA}L_2 \text{ and } C^{MA}_{L_1,L_2}(w,v,a,b),$$

where the boolean-valued multi-agent coincidence condition is,

$$\begin{aligned} C^{MA}_{L_1,L_2}(w,v,a,b) = \; & (wa \in L_1 \wedge vb \in L_2) \wedge \\ & \{(va \notin L_2 \wedge wb \notin L_1) \vee a = b\} \end{aligned}$$

□

There are two aspects of this product which differ from a standard notion of a synchronous product: first, simultaneity, i.e. the vector form of the event labels which necessarily preserve an equal count on symbols from each system; and second, a synchronisation constraint that applies only in cases where a given symbol may be accepted by both systems *at the current state*, rather than whenever the symbol is in $\Sigma_1 \cap \Sigma_2$.

A vector-state system with simultaneous transitions was defined in [**57**]. The MA product differs from this in at least two ways. First, vector symbols in [**57**] may have an arbitrary number of components, i.e. the symbols are subsets of Σ, whereas in the MA product, symbols always have the same number of components (equal to the number of component systems). Second, strict simultaneity, as defined here in the simultaneous and MA products, may cause blocking in one system (and hence the product system) when there is simply no symbol available in the other system.

2.2. The Simultaneous and Vector Synchronous Products

In order to aid the characterisation of the MA product, which is the main goal of this section, several other products will be considered. It will be shown that the MA product is a combination of the simultaneous and (state-dependent) vector synchronous product.

Definition 2.3. Simultaneous Product

$$G_1||_{sim}G_2 \triangleq (X_1 \times X_2, \Sigma_1 \times \Sigma_2, \delta_{sim}, Q_{0_1} \times Q_{0_2}, Q_{m_1} \times Q_{m_2})$$

where,

$$\delta_{sim}(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix}) = \begin{cases} \begin{bmatrix} \delta_1(x,u) \\ \delta_2(y,v) \end{bmatrix} & \text{if } \delta_1(x,u)! \text{ and } \delta_2(y,v)! \\ \text{undefined} & \text{otherwise} \end{cases}$$

□

In the simultaneous product, G_1 and G_2 must make a transition at every step. There is no other restriction.

It is not necessary that each component change state at each step; there may be self-loops in the original component models. The interpretation of loops within one of the component machines is that of a "stay" or "wait" denoted γ_i for a machine G_i (see below). Note that if both machines have self loops at every state in both machines, the state transition structure of the simultaneous product is richer than

the shuffle product; i.e. all single-component state transitions can be made as well as all simultaneous transitions.

Again, the the language accepted by $G_1||_{sim}G_2$ can be determined directly from the component languages, i.e.

$$L_1||_{sim}L_2 = \{ \begin{bmatrix} u \\ v \end{bmatrix} : u \in L_1, v \in L_2, |u| = |v| \}$$

where $|\sigma|$ is the length, or number of symbols, in σ.

The *natural projection* and *component-wise projection*

$$P_i : \Sigma^* \longrightarrow \Sigma_i^* \text{ and } I\!P_i : (\Sigma \times \Sigma)^* \longrightarrow \Sigma^*$$

can both be defined recursively as follows,

$$(i) P_i(\epsilon) = \epsilon$$
$$(ii) P_i(\sigma) = \sigma \text{ if } \sigma \in \Sigma_i, \epsilon \text{ if } \sigma \in \Sigma/\Sigma_i$$
$$(iii) P_i(s\sigma) = P_i(s)P_i(\sigma) \text{ for } s \in \Sigma^*, \sigma \in \Sigma.$$

and,

$$(i) I\!P_i(\vec{\epsilon}) = \epsilon$$
$$(ii) I\!P_i(\vec{u}) = u_i$$
$$(iii) I\!P_i(\vec{s}\vec{u}) = I\!P_i(\vec{s})I\!P_i(\vec{u}) \text{ for } \vec{s} \in \vec{\Sigma}^*, \vec{u} \in \vec{\Sigma},$$

where $\vec{\epsilon} = \begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix}$ and $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$.

In the *scalar synchronous product* (see [**102**] or [**47**] for instance),

$$L_1||_s L_2 \stackrel{def}{=} P_1^{-1}L_1 \bigcap P_2^{-1}L_2, \tag{2.4.1}$$

two generators G_1 and G_2 for L_1 and L_2 respectively would interact by synchronizing the events that have labels in $\Sigma_1 \cup \Sigma_2$.

A synchronous product can also be formed in the vector format with the addition of explicit "wait" events as follows.

Definition 2.4. Vector Synchronous Product

$$G_1||_{vs}G_2 = (X_1 \times X_2, (\{\gamma_1\} \cup \Sigma_1) \times (\{\gamma_2\} \cup \Sigma_2), \delta_{vs}, Q_{0_1} \times Q_{0_2}, Q_{m_1} \times Q_{m_2})$$

where,

$$\delta_{vs}(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix}) = \begin{cases} \begin{bmatrix} \delta_1(x,u) \\ \delta_2(y,v) \end{bmatrix} & \text{if} \quad ((v=\gamma_2) \wedge (u \in \Sigma/\Sigma_2) \wedge \delta_1(x,u)!) \\ & \quad \vee ((u=\gamma_1) \wedge (v \in \Sigma/\Sigma_1) \wedge \delta_2(y,v)!) \\ & \quad \vee (\delta_1(x,u)! \wedge \delta_2(y,v)! \wedge \\ & \qquad ((u \in \Sigma/\Sigma_2 \wedge v \in \Sigma/\Sigma_1) \vee (u=v))) \\ \text{undefined} & \text{otherwise,} \end{cases}$$

with the convention that the symbols γ_1 and γ_1 are not in $\Sigma_1 \cup \Sigma_2$, and $\delta_1(x,\gamma_1) = x$ for all $x \in X_1$ and $\delta_2(y,\gamma_2) = y$ for all $y \in X_2$. □

In the vector synchronous product, G_1 and G_2 generate $L_1||_{vs}L_2$ by synchronising those events with labels $\sigma \in \Sigma_1 \cap \Sigma_2$, but may each change state independently on unshared symbols via the vector symbols $\begin{bmatrix} \gamma_1 \\ b \end{bmatrix}, \begin{bmatrix} a \\ \gamma_2 \end{bmatrix}$ or $\begin{bmatrix} a \\ b \end{bmatrix}$.

With projection $P_{\backslash\{\gamma\}}$ defined as the natural projection $\Sigma^* \longrightarrow \Sigma^*/\{\gamma_1,\gamma_2\}$, the components of the vector synchronous product match the natural projections of the scalar product, i.e.

$$P_{\backslash\{\gamma\}}[I\!P_i(L_1||_{vs}L_2)] = P_i(L_1||_sL_2), \quad i=1,2,$$

but in general,

$$P_{\backslash\{\gamma\}}[I\!P_i(L_1||_{vs}L_2)] \subseteq L_i. \tag{2.4.2}$$

An issue of interest is when the projected components of a product of two systems actually equal the original systems prior to forming the product (i.e. when equation 2.4.2 yields equality). It will be shown that this requires that component systems stem from consistent masked observations of a single universal generating process (i.e. the language $L_1||_sL_2$). Equivalently, this could be viewed as the absence of blocking in each component due to the other component not accepting a symbol even though this symbol is observed through the observation mask.

Definition 2.5. Compatible languages

The languages L_1 and L_2 are *compatible* if and only if

$$P_{\backslash\{\gamma\}}[I\!P_1(L_1||_{vs}L_2)] = L_1 \text{ and } P_{\backslash\{\gamma\}}[I\!P_2(L_1||_{vs}L_2)] = L_2,$$

or equivalently, iff $P_1(L_1||_sL_2) = L_1$ and $P_2(L_1||_sL_2) = L_2$. □

Theorem 2.1. *Let L_1 and L_2 be prefix-closed languages. If L_1 and L_2 are compatible then $P_1(L_2) = P_2(L_1)$.*

Proof. The projections P_1 and P_2 commute, so $P_1P_2(K) = P_2P_1(K)$ for all K. In particular, with $K = L_1||_S L_2$,

$$P_2P_1(L_1||_sL_2) = P_1P_2(L_1||_sL_2) \quad \Longrightarrow \quad P_2(L_1) = P_1(L_2)$$

since $P_i(L_1||_sL_2) = L_i$ by assumption. ∎

A pertinent condition in this context appeared in [**80**] termed *decomposable*. The definition is included here for comparison and completeness.

Definition 2.6. [80] Decomposable Languages

A language $K \subseteq L(G)$ is *decomposable* (w.r.t. G and projections P_1 and P_2) if

$$K = L(G) \bigcap P_1^{-1}(P_1(K)) \bigcap P_2^{-1}(P_2(K)).$$

□

Proposition 2.1. *When L_1 and L_2 are compatible, $L_1||_sL_2$ is decomposable with respect to G accepting Σ^*.* □

Proof. With $K = L_1||_sL_2$ and $L(G) = \Sigma^*$ it can be checked that,

$$\begin{aligned} \Sigma^* \bigcap P_1^{-1}(P_1(L_1||_sL_2)) \bigcap P_2^{-1}(P_2(L_1||_sL_2)) &= \Sigma^* \bigcap P_1^{-1}(L_1) \bigcap P_2^{-1}(L_2) \\ &= \Sigma^* \bigcap L_1||_sL_2 \\ &= L_1||_sL_2. \end{aligned}$$

∎

Note that the contrary does not hold. A counterexample is Figure 4.2 where $L_1||_sL_2 = \{a, \epsilon\}$ is decomposable with respect to a^*, but L_1 and L_2 are not compatible.

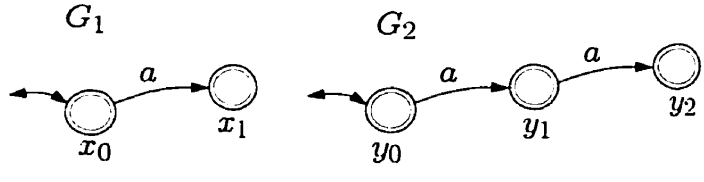


FIGURE 4.2. L_1 and L_2 are not compatible.

A second property that has appeared in the literature, [**102**], is,

Definition 2.7. [102]Non-conflicting

Two languages, L_1 and L_2, are *non-conflicting* if

$$\overline{L_1} \bigcap \overline{L_2} = \overline{L_1 \bigcap L_2}.$$ □

Compatibility is incomparable to *non-conflicting*. Figure 4.2 shows non-conflicting but also not compatible languages, and the case $L_1 = \{ab\}, L_2 = \{ac\}$ with $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a, c\}$ corresponds to compatible and not non-conflicting languages.

2.3. The MA Product as a Combination of Simultaneous and Synchronous Products

The opportunity for blocking alluded to above (i.e. where one component is blocked due to the other not accepting a recognised symbol) does not appear in the MA product in 2.1. Hence, the comparison of the MA product with a synchronous product requires another product system, i.e. a more liberal (state-dependent) synchronous product is defined.

Definition 2.8. Vector (State-dependent) Synchronous Product

$$G_1 ||_{vsd} G_2 = (X_1 \times X_2, (\{\gamma_1\} \cup \Sigma_1) \times (\{\gamma_2\} \cup \Sigma_2), \delta_{vsd}, Q_{0_1} \times Q_{0_2}, Q_{m_1} \times Q_{m_2})$$

where ,

$$\delta_{vsd}(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix}) = \begin{cases} \begin{bmatrix} \delta_1(x,u) \\ \delta_2(y,v) \end{bmatrix} & \text{if } (u = v \wedge \delta_1(x,u)! \wedge \delta_2(y,v)!) \\ & \vee ((v = \gamma_2) \wedge \delta_1(x,u)! \wedge \neg\delta_2(y,u)!) \\ & \vee ((u = \gamma_1) \wedge \neg\delta_1(x,v)! \wedge \delta_2(y,v)!) \\ & \vee (\delta_1(x,u)! \wedge \neg\delta_2(y,u)! \\ & \quad \wedge \neg\delta_1(x,v)! \wedge \delta_2(y,v)!) \\ \text{undefined} & \text{otherwise} \end{cases}$$

with the convention that the symbols γ_1 and γ_1 are not in $\Sigma_1 \cup \Sigma_2$, and $\delta_1(x, \gamma_1) = x$ for all $x \in X_1$ and $\delta_2(y, \gamma_2) = y$ for all $y \in X_2$. □

In the (state-dependent) synchronous product, G_1 and G_2 synchronise similarly labelled transitions if and only if that event is available *at the current state* in both machines. Otherwise they may act independently.

Contrary to the vector synchronous product, it is always the case that

$$P_{\backslash\{\gamma\}}[I\!P_i(L(G_1 ||_{vsd} G_2))] = L_i,$$

because $\delta_1(x,u)! \longrightarrow \delta(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} u \\ \gamma \end{bmatrix})!$.

In general $I\!P_i(L(G_1||_{vsd}G_2)) \supseteq I\!P_i(L_1||_{sv}L_2)$ because the evolution condition in the vector synchronous product is more stringent than in the state-dependent vector synchronous. Equality, $I\!P_i(L(G_1||_{vsd}G_2)) = I\!P_i(L_1||_{vs}L_2)$, occurs if and only if L_1 and L_2 are compatible languages because all instances of blocking due to the absence of a symbol in one of the alphabets is removed. In both products then, blocked symbols requre the same conditions; that an alternative symbol is available both agents.

Similarly, it is always the case that $P_2(I\!P_1(L_1||_{vs}L_2)) = P_1(I\!P_2(L_1||_{vs}L_2))$, but with the additional constraint that L_1 and L_2 are compatible, we also get this commutative property for the (state-dependent) synchronous product, i.e.,

$$P_2(I\!P_1(L(G_1||_{vsd}G_2))) = P_1(I\!P_2(L(G_1||_{vsd}G_2))). \tag{2.4.3}$$

Again, Figure 4.2 provides a counter example for (2.4.3) when L_1 and L_2 are not compatible. Here

$$L(G_1||_{vsd}G_2) = \{\begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}^*, \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}^* \begin{bmatrix} a \\ a \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}^*, \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}^* \begin{bmatrix} a \\ a \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}^* \begin{bmatrix} a \\ \gamma_2 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix}^*\},$$

so $P_2(I\!P_1(L(G_1||_{vsd}G_2))) = \{\epsilon, a, aa\}$ while $P_1(I\!P_2(L(G_1||_{vsd}G_2))) = \{\epsilon, a\}$.

Two examples illustrating the various products for two simple systems can be found in Figures 4.5 and 4.3 and a third example of the MA product is shown in 4.4.

To each transition function $\delta : Z \times \Sigma \to Z$ we associate a transition relation $\tilde{\delta} \subseteq Z \times \Sigma \times Z$ by $(z,\sigma,z') \in \tilde{\delta} \iff \delta(z,\sigma) = z'$. The MA product can then be categorised by the following theorem which highlights the way in which the MA product from the intersection of simultaneous and vector state-dependent synchronous products.

Theorem 2.2. *Consider two deterministic automata G_1 and G_2 with transition functions δ_1 and δ_2. Let $\tilde{\delta}_{MA}, \tilde{\delta}_{sim}$ and $\tilde{\delta}_{vsd}$ be the transition relations associated with the products $G_1||_{MA}G_2$, $G_1||_{sim}G_2$ and $G_1||_{vsd}G_2$ extended (as undefined) to the domain $(\{\gamma_1\} \cup \Sigma_1 \times (\{\gamma_2\} \cup \Sigma_2)$. Then it is the case that,*

$$\tilde{\delta}_{MA} = \tilde{\delta}_{sim} \bigcap \tilde{\delta}_{vsd},$$

□

Proof. Consider a state $\begin{bmatrix} x \\ y \end{bmatrix}$ and event $\begin{bmatrix} a \\ b \end{bmatrix}$.

$$\left(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} \delta_1(x,a) \\ \delta_2(y,b) \end{bmatrix}\right) \in \tilde{\delta}_{sim} \bigcap \tilde{\delta}_{vsd}$$

$$\iff \delta_{sim}\left(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} a \\ b \end{bmatrix}\right) = \begin{bmatrix} \delta_1(x,a) \\ \delta_2(y,b) \end{bmatrix} \quad \wedge \quad \delta_{vsd}\left(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} a \\ b \end{bmatrix}\right) = \begin{bmatrix} \delta_1(x,a) \\ \delta_2(y,b) \end{bmatrix}$$

$$\iff [\delta_1(x,a)! \wedge \delta_2(y,b)! \wedge a \in \Sigma_1 \wedge b \in \Sigma_2] \quad \wedge \quad [((a=b) \wedge \delta_1(x,a)! \wedge \delta_2(y,b)!)$$
$$\vee ((b=\gamma_2) \wedge \delta_1(x,a)! \wedge \neg\delta_2(y,a)!)$$
$$\vee ((a=\gamma_1) \wedge \neg\delta_1(x,b)! \wedge \delta_2(y,b)!)$$
$$\vee (\delta_1(x,a)! \wedge \neg\delta_2(y,a)!$$
$$\wedge\neg\delta_1(x,b)! \wedge \delta_2(y,b)!)]$$

$$\iff [\delta_1(x,a)! \wedge \delta_2(y,b)! \wedge a \in \Sigma_1 \wedge b \in \Sigma_2] \quad \wedge \quad [((a=b) \wedge \delta_1(x,a)! \wedge \delta_2(y,b)!)$$
$$\vee (\delta_1(x,a)! \wedge \neg\delta_2(y,a)!$$
$$\wedge\neg\delta_1(x,b)! \wedge \delta_2(y,b)!)]$$

$$\iff [a \in \Sigma_1 \wedge b \in \Sigma_2] \wedge [\delta_1(x,a)! \wedge \delta_2(y,b)! \wedge [(a=b) \vee (\neg\delta_2(y,a)! \wedge \neg\delta_1(x,b)!)]$$

$$\iff \delta_{MA}\left(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} a \\ b \end{bmatrix}\right) = \begin{bmatrix} \delta_1(x,a) \\ \delta_2(y,b) \end{bmatrix}$$

$$\iff \left(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} \delta_1(x,a) \\ \delta_2(y,b) \end{bmatrix}\right) \in \tilde{\delta}_{MA}$$

■

In the case that there are self-loops at each node labelled distinctly in both automata from all non self-loop transitions this reduces to

$$\tilde{\delta}_{MA} = \tilde{\delta}_{vsd}.$$

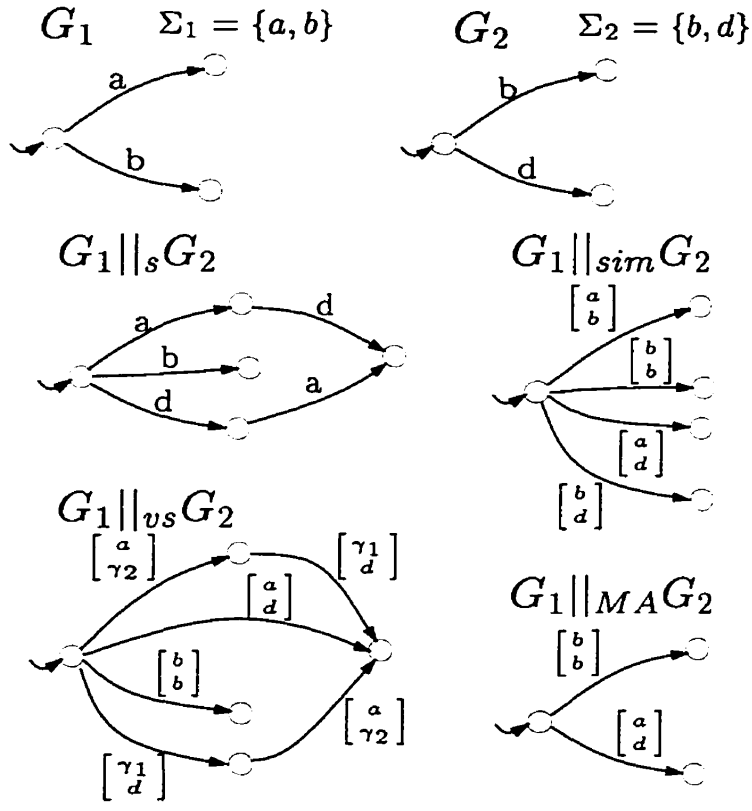


FIGURE 4.3. Example 2 of the various products.

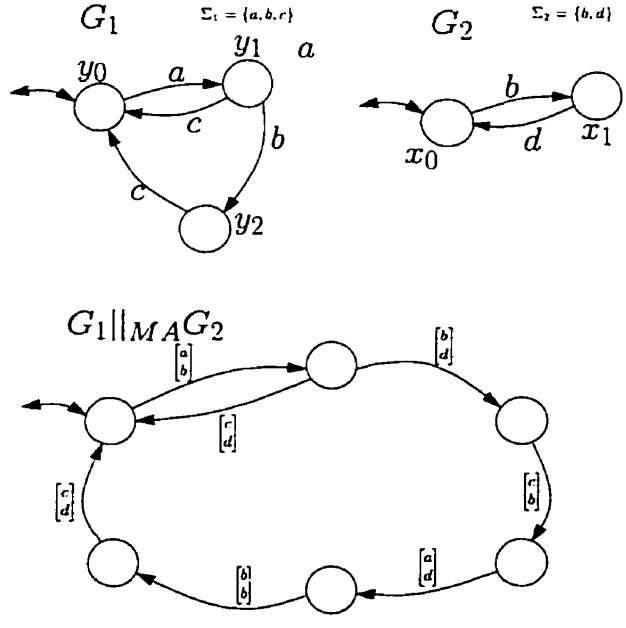


FIGURE 4.4. Example of the MA product.

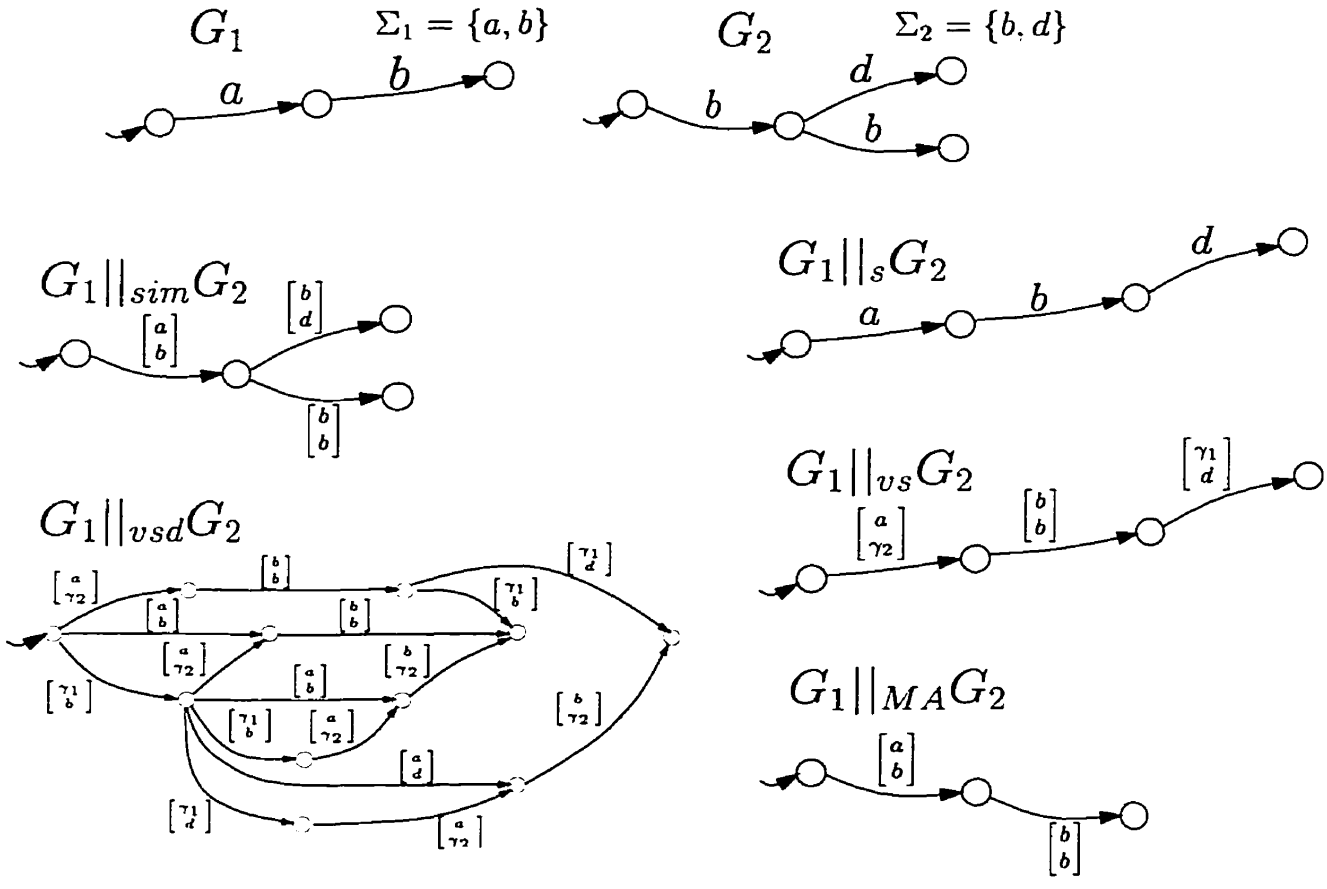


FIGURE 4.5. Example 1 of the various products.

2.4. Commutativity and Associativity of the MA Product

The MA product is not commutative as the components are ordered. On the other hand, there is a symmetry between the components, i.e.

$$I\!P_1(L_1||_{MA}L_2) = I\!P_2(L_2||_{MA}L_1).$$

In order to show associativity, consider the MA product of N systems.

Definition 2.9. $G_1||_{MA}\cdots||_{MA}G_N$

The MA product of N systems

$$G_i = (X_i, \Sigma_i, \delta_i, Q_{0_i}, Q_{m_i}), \quad i = 1, ..., N$$

is defined to be

$$G_{MA} = (X_{MA}, \Sigma_{MA}, \delta_{MA}, Q_{0_{MA}}, Q_{m_{MA}})$$

where,

$$\begin{aligned} X_{MA} &= X_1 \times X_2 \times \cdots \times X_N \\ \Sigma_{MA} &= \Sigma_1 \times \Sigma_2 \times \cdots \times \Sigma_N \\ Q_{0_{MA}} &= Q_{0_1} \times Q_{0_2} \times \cdots \times Q_{0_N} \\ Q_{m_{MA}} &= Q_{m_1} \times Q_{m_2} \times \cdots \times Q_{m_N} \end{aligned}$$

and

$$\delta_{MA}(\vec{x},\vec{u}) = \begin{cases} \begin{bmatrix} \delta_1(x_1,u_1) \\ \delta_2(x_2,u_2) \\ \cdot \\ \delta_N(x_N,u_N) \end{bmatrix} & \text{if } [\forall i.1 \le i \le N.\ \delta_i(x_i,u_i)!] \\ & \quad \wedge [\forall i,j,1 \le i,j \le N.\ ((u_i = u_j) \\ & \qquad \vee (\neg\delta_i(x_i,u_j)! \wedge \neg\delta_j(x_j,u_i)!))\ ] \\ \text{undefined} & \text{otherwise} \end{cases}$$

□

Now, consider the MA product of two MA systems.

Definition 2.10. $(G_1||_{MA}\cdots||_{MA}G_M)||_{MA}(G_{M+1}||_{MA}\cdots||_{MA}G_N)$

Let $(G_1||_{MA}\cdots||_{MA}G_M)$ and $(G_{M+1}||_{MA}\cdots||_{MA}G_N)$ be two MA systems with $1 \le M < N$. Let their transition functions be $\delta_{1,M}$ and $\delta_{M+1,N}$ respectively. The transition function $\delta_{1,M,N}$ for their product is,

$$\delta_{1,M,N}(\vec{x},\vec{u}) = \begin{cases} \begin{bmatrix} \delta_{1,M}(\vec{x}_1^M,\vec{u}_1^M) \\ \delta_{M+1,N}(\vec{x}_{M+1}^N,\vec{u}_{M+1}^N) \end{bmatrix} & \text{if } (\delta_{1,M}(\vec{x}_1^M,\vec{u}_1^M)! \wedge \delta_{M+1,N}(\vec{x}_{M+1}^N,\vec{u}_{M+1}^N!)) \\ & \quad \wedge [\forall i, 1 \le i \le M.\ \forall j, M+1 \le j \le N. \\ & \qquad ((u_i = u_j) \vee (\neg\delta_i(x_i,u_j)! \wedge \neg\delta_j(x_j,u_i)!))\ ] \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $\vec{x} = [x_1,\cdots,x_N]^T$ and $\vec{x}_i^j = [x_i,\cdots,x_j]^T$ and similarly for $\vec{u}$ and $\vec{u}_i^j$. □

Theorem 2.3.

The transition function $\delta_{1,M,N}$ in Definition 2.10 is independent of M. Furthermore it is equal to the transition function δ_{MA} of Definition 2.9.

Proof. We proceed by induction on N. The case $N = 2$ is trivial as $M = 1$ and $\delta_{1,1,2}(\vec{x},\vec{u}) = \delta_{MA}(\vec{x},\vec{u})$ of Definition 2.9.

We assume the result holds up to N as the induction hypothesis. For the case $N+1$, note that, when defined,

$$\begin{aligned}\delta_{1,M,N+1}(\vec{x},\vec{u}) &= \begin{bmatrix} \delta_{1,M}(\vec{x}_1^M,\vec{u}_1^M) \\ \delta_{M+1,N+1}(\vec{x}_{M+1}^{N+1},\vec{u}_{M+1}^{N+1}) \end{bmatrix} \\ &= \begin{bmatrix} \delta_1(x_1,u_1) \\ \delta_2(x_2,u_2) \\ \cdot \\ \delta_{N+1}(x_{N+1},u_{N+1}) \end{bmatrix}.\end{aligned}$$

By the induction hypothesis $\delta_{1,M}(\vec{x}_1^M,\vec{u}_1^M)$ and $\delta_{M+1,N+1}(\vec{x}_{M+1}^{N+1},\vec{u}_{M+1}^{N+1})$, which have N or less components, can be formulated as in Definition 2.9. Note that, when defined, $\delta_{1,M,N+1}(\vec{x},\vec{u})$ is independent of M and equal to $\delta_{MA}(\vec{x},\vec{u})$ in Definition 2.9. Hence only the condition that the transition *is defined* needs to be checked. By the induction hypothesis, the condition for $\delta_{1,M}(\vec{x}_1^M,\vec{u}_1^M)!$ and $\delta_{M+1,N+1}(\vec{x}_{M+1}^{N+1},\vec{u}_{M+1}^{N+1})!$ is formulated as in Definition 2.9. Therefore the condition $\delta_{1,M,N+1}(\vec{x},\vec{u})!$ is,

$$\begin{aligned}&\delta_{1,M}(\vec{x}_1^M,\vec{u}_1^M)! \ \wedge\ \delta_{M+1,N+1}(\vec{x}_{M+1}^{N+1},\vec{u}_{M+1}^{N+1})! \ \wedge\ [\forall i, 1\le i\le M.\forall j, M+1\le j\le N+1. \\ &\qquad ((u_i=u_j)\vee(\neg\delta_i(x_i,u_j)!\wedge\neg\delta_j(x_j,u_i)!))\ ] \\ \iff\ &[\forall i, 1\le i\le M.\ \delta_i(x_i,u_i)!] \ \wedge\ [\forall i, M+1\le i\le N+1.\ \delta_i(x_i,u_i)!] \\ &\wedge[\forall i,j, 1\le i,j\le M.\ ((u_i=u_j) \ \wedge[\forall i,j, M+1\le i,j\le N+1.\ ((u_i=u_j) \\ &\quad \vee(\neg\delta_i(x_i,u_j)!\wedge\neg\delta_j(x_j,u_i)!))\ ] \quad \vee(\neg\delta_i(x_i,u_j)!\wedge\neg\delta_j(x_j,u_i)!))\ ] \\ &\qquad \wedge\ [\forall i, 1\le i,j\le M.\forall j, M+1\le j\le N+1. \\ &\qquad\quad ((u_i=u_j)\vee(\neg\delta_i(x_i,u_j)!\wedge\neg\delta_j(x_j,u_i)!))\ ] \\ \iff\ &[\forall i, 1\le i\le N+1.\ \delta_i(x_i,u_i)!] \\ &\wedge[\forall i,j, 1\le i,j\le N+1.\ ((u_i=u_j) \\ &\quad \vee(\neg\delta_i(x_i,u_j)!\wedge\neg\delta_j(x_j,u_i)!))\ ]\end{aligned}$$

which is equal to the condition that $\delta_{MA}(\vec{x},\vec{u})!$ for $N+1$ systems as defined in Definition 2.9. ∎

The result of the theorem is that the products of N ordered systems,

$$G_1||_{MA}G_2||_{MA}\cdots||_{MA}G_N,$$

is unique and independent of the order in which the products are performed.

The interpretation of the product is that all the automata necessarily make a transition in unison at every step and the synchronisation constraint is that if a transition, labelled μ for instance, is to be made by some automaton then any automata for which a similarly labelled transition is available must make that transition.

It is interesting that there are circumstances in which all components have at least one transition available at a given state, but the unique product has none at the vector state. This is similar to the loss of the *deadlock-free* property for languages over intersection (see [**53**]).

The definition of the MA product for languages can similarly be extended to the case of N systems

Definition 2.11. $L_1||_{MA}L_2||_{MA}\cdots||_{MA}L_N$

Let $L_1, L_2, ..., L_N$ be prefix-closed languages. The unique language resulting from their MA product is defined recursively as follows.

$$\vec{\epsilon} \in L_1||_{MA}L_2||_{MA}\cdots||_{MA}L_N$$
$$\vec{u}\vec{a} \in L_1||_{MA}L_2||_{MA}\cdots||_{MA}L_N$$
$$\text{if } \vec{u} \in L_1||_{MA}L_2||_{MA}\cdots||_{MA}L_N \text{ and}$$
$$[(\forall i, 1 \leq N.u_i a_i \in L_i) \quad \text{and } (\forall i, j, i \neq j.(u_i a_j \notin L_i \vee a_i = a_j))]$$

□

3. MA Product and Vector Languages

In this section an algebra is developed for describing the vector languages. The following definition is made to show, below, how the MA product can be formed through taking the intersection of the permitted behaviour with respect to each component.

Definition 3.1. Embedded Constraint $\left[\begin{smallmatrix} L \\ \cdot \end{smallmatrix}\right]$

Let L be a prefix-closed language over the alphabet $\Sigma_1 \subseteq \Sigma$. Define the *embedded constraint* of L recursively as follows:

$$\begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix} \in \begin{bmatrix} L \\ \cdot \end{bmatrix},$$

and for $a \in \Sigma_1, b \in \Sigma, w \in \Sigma_1^*, v \in \Sigma^*$,

$$\begin{bmatrix} w \\ v \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \in \begin{bmatrix} L \\ * \end{bmatrix} \text{ if } \begin{bmatrix} w \\ v \end{bmatrix} \in \begin{bmatrix} L \\ * \end{bmatrix} \text{ and } C_L(w, a, b).$$

where $C_L(w, a, b)$ is a boolean valued *coincidence condition*,

$$C_L(w, a, b) = (wa \in L) \wedge (wb \notin L \text{ or } a = b).$$

$\begin{bmatrix} * \\ L \end{bmatrix}$ is defined similarly with co-ordinates interchanged. □

The relationship with union and intersection is as follows.

Proposition 3.1.

$$1.\ \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} L_2 \\ * \end{bmatrix} \subseteq \begin{bmatrix} L_1 \bigcap L_2 \\ * \end{bmatrix}$$

with equality if and only if $L_1 \cap L_2 = \emptyset$ *or* $L_1 = L_2$.

$$2.\ \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcup \begin{bmatrix} L_2 \\ * \end{bmatrix} \supseteq \begin{bmatrix} L_1 \bigcup L_2 \\ * \end{bmatrix}$$

Proof.

(1) Proceed by induction on length of strings. For the base case, it is true that $\vec{\epsilon} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} L_2 \\ * \end{bmatrix} \Longrightarrow \vec{\epsilon} \in \begin{bmatrix} L_1 \bigcap L_2 \\ * \end{bmatrix}$ trivially.
Assume, for vector strings of length n, that, $\vec{u} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} L_2 \\ * \end{bmatrix} \Longrightarrow \vec{u} \in \begin{bmatrix} L_1 \bigcap L_2 \\ * \end{bmatrix}$.
For strings of length $n+1$ (with $\vec{u} = \begin{bmatrix} u \\ v \end{bmatrix}$) consider,

$$\begin{aligned}
&\vec{u} \begin{bmatrix} a \\ b \end{bmatrix} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} L_2 \\ * \end{bmatrix} \\
&\iff \vec{u} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} L_2 \\ * \end{bmatrix} \wedge (ua \in L_1 \wedge [ub \notin L_1 \vee a = b)] \wedge (ua \in L_2 \wedge [ub \notin L_2 \vee a = b)] \\
&\iff \vec{u} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} L_2 \\ * \end{bmatrix} \wedge (ua \in L_1 \wedge ua \in L_2) \wedge [(ub \notin L_1 \wedge ub \notin L_2) \vee (a = b)] \\
&\iff \vec{u} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} L_2 \\ * \end{bmatrix} \wedge (ua \in L_1 \cap L_2) \wedge [(ub \notin L_1 \cup L_2) \vee (a = b)] \qquad (3.4.1) \\
&\Longrightarrow \vec{u} \in \begin{bmatrix} L_1 \bigcap L_2 \\ * \end{bmatrix} \wedge (ua \in L_1 \cap L_2) \wedge [(ub \notin L_1 \cap L_2) \vee (a = b)] \qquad (3.4.2) \\
&\iff \vec{u} \begin{bmatrix} a \\ b \end{bmatrix} \in \begin{bmatrix} L_1 \bigcap L_2 \\ * \end{bmatrix}
\end{aligned}$$

which completes the inductive step. Equality holds if $(L_1 \cup L_2)/(L_1 \cap L_2) = \emptyset$ (compare 3.4.1 and 3.4.2).

(2) Follows a similar induction to (1).

Base case: $\vec{\epsilon} \in \left[L_1 \bigcup_* L_2\right] \Longrightarrow \vec{\epsilon} \in \left[L_1\right] \bigcup \left[L_2\right]$.

Inductive hypothesis: $\vec{u} \in \left[L_1 \bigcup_* L_2\right] \Longrightarrow \vec{u} \in \left[L_1\right] \bigcup \left[L_2\right]$.

Inductive step (with $\vec{u} = \begin{bmatrix} u \\ v \end{bmatrix}$):

$$
\begin{aligned}
&\vec{u}\begin{bmatrix} a \\ b \end{bmatrix} \in \left[L_1 \bigcup_* L_2\right] \\
&\quad \Longleftrightarrow \quad \vec{u} \in \left[L_1 \bigcup_* L_2\right] \wedge (ua \in L_1 \cup L_2) \wedge [(ub \notin L_1 \cup L_2) \vee (a = b)] \\
&\quad \Longleftrightarrow \quad \vec{u} \in \left[L_1 \bigcup_* L_2\right] \wedge (ua \in L_1 \vee ua \in L_2) \wedge [(ub \notin L_1 \wedge ub \notin L_2) \vee (a = b)] \\
&\quad \Longleftrightarrow \quad \vec{u} \in \left[L_1 \bigcup_* L_2\right] \wedge (ua \in L_1 \vee ua \in L_2) \wedge [(ub \notin L_1 \vee a = b) \wedge (ub \notin L_2 \vee a = b)] \\
&\quad \Longleftrightarrow \quad \vec{u} \in \left[L_1 \bigcup_* L_2\right] \wedge \{((ua \in L_1) \wedge [ub \notin L_1 \vee a = b] \wedge [ub \notin L_2 \vee a = b]) \\
&\qquad\qquad\qquad\qquad \vee ((ua \in L_2) \wedge [ub \notin L_1 \vee a = b] \wedge [ub \notin L_2 \vee a = b])\} \\
&\quad \Longrightarrow \quad \vec{u} \in \left[L_1\right] \bigcup \left[L_2\right] \wedge \{(ua \in L_1 \wedge [ub \notin L_1 \vee a = b)] \vee (ua \in L_2 \wedge [ub \notin L_2 \vee a = b)]\} \\
&\quad \Longleftrightarrow \quad \vec{u}\begin{bmatrix} a \\ b \end{bmatrix} \in \left[L_1\right] \bigcup \left[L_2\right]
\end{aligned}
$$

which completes the proof. ∎

The sense in which the embedded constraint can be used to form a component-wise construction of the MA product can now be made explicit. This is illustrated in Figure 4.6 and an example is provided in 4.7

Proposition 3.2.

$$L_1 ||_{MA} L_2 = \left[L_1\right] \bigcap \left[L_2\right]$$

Proof. Note first that

$$C^{MA}_{L_1,L_2}(w,u,a,b) \Longleftrightarrow C_{L_1}(w,a,b) \text{ and } C_{L_2}(u,a,b).$$

Proceed by induction on lengths of strings $\begin{bmatrix} w \\ v \end{bmatrix}$. For the base case it is clear that,

$$\begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} * \\ L_2 \end{bmatrix}, \quad \begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix} \in L_1 || _{MA} L_2.$$

Assume $\begin{bmatrix} w \\ v \end{bmatrix} \in L_1 || _{MA} L_2 \iff \begin{bmatrix} w \\ v \end{bmatrix} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} * \\ L_2 \end{bmatrix}$. Then,

- $\begin{bmatrix} w \\ v \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \in L_1 || _{MA} L_2 \iff \begin{bmatrix} w \\ v \end{bmatrix} \in L_1 || _{MA} L_2$ and $C^{MA}_{L_1,L_2}(w, v, a, b)$
  $\iff$ $(w \in L_1$ and $v \in L_2)$ and
  $C_{L_1}(w, a, b)$ and $C_{L_2}(v, a, b)$
  $\iff$ $\begin{bmatrix} w \\ v \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \in \begin{bmatrix} L_1 \\ * \end{bmatrix} \bigcap \begin{bmatrix} * \\ L_2 \end{bmatrix}$

■

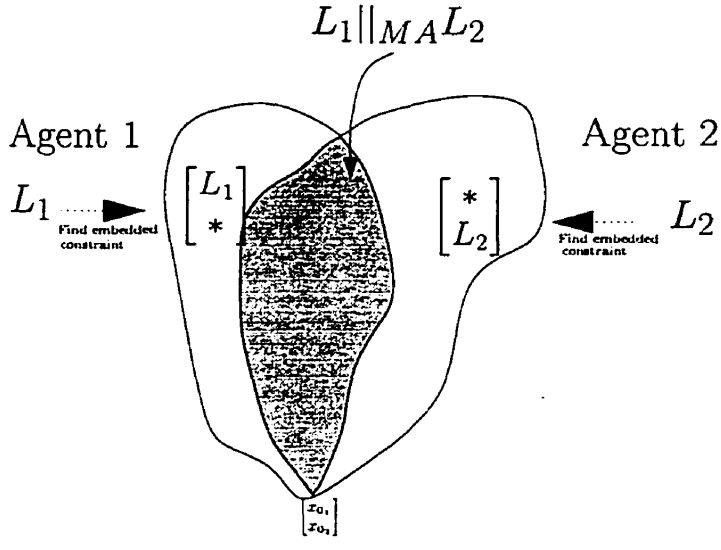


FIGURE 4.6. Illustration of the embedded constraint.

3.1. Non-Simultaneous Accepted Languages for Vector Systems

The standard sense of (marked) acceptance is that a string is accepted if its associated state trajectory ends at a goal state. The standard label will be used for the language of (marked) accepted strings, i.e. $L_m(G)$, which applies both to scalar and vector systems.

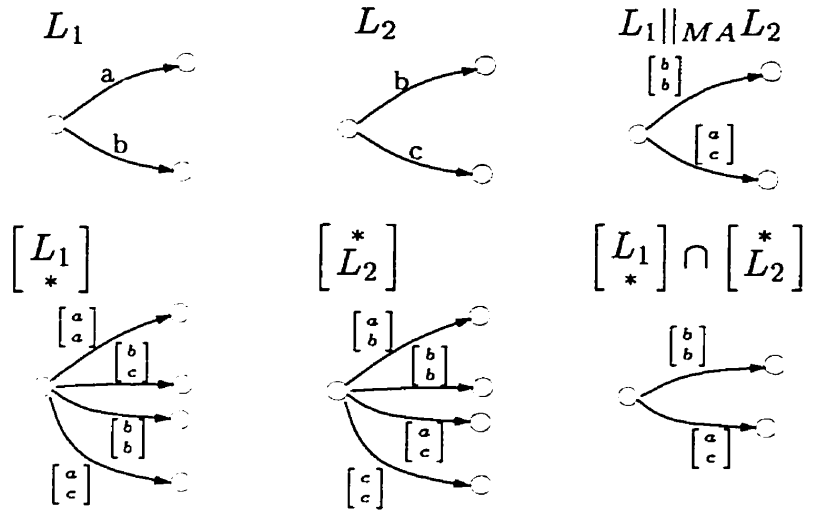


FIGURE 4.7. Example of the embedded constraint formation of MA product.

Simultaneity clearly restricts the accepted language as accepted strings (of vector symbols) must be composed of equal length strings from the language L_1 and L_2 (or $L_m(G_1)$ or $L_m(G_2)$ for marked acceptance).

For example, in Figure 4.8, $L_m(G_1||_{MA}G_2) = \{\begin{bmatrix} a \\ a \end{bmatrix}\}$, and $L_m(G_2||_{MA}G_3) = \{\begin{bmatrix} a \\ a \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix}\}$. If the goal set in G_2 was changed from $\{y_1, y_2\}$ to $\{y_1\}$, the result would be $L_m(G_2||_{MA}G_3) = \emptyset$.

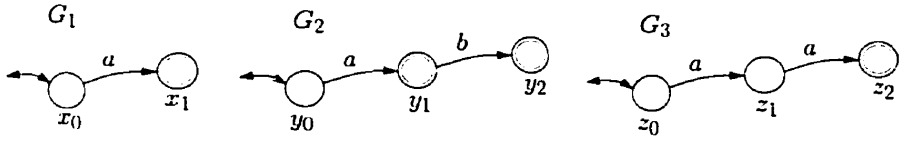


FIGURE 4.8. MA products and simultaneity: accepted strings must be of equal length.

Because of the restriction imposed by simultaneity a more general notion of language acceptance is now introduced (provided here for the case of two automata, but extendible to arbitrary finite N).

Definition 3.2. Non-Simultaneous Acceptance

For a given (x_{01}, y_{0i}), when self loops (labelled γ_i) are introduced at each goal state in Q_{m_i} in the automaton G_i, if the string $\vec{\sigma}$ is such that for each i, $1 \leq i \leq 2$,

$$\delta_i(x_{0i}, I\!P_i(\vec{\sigma})) \in Q_{m_i},$$

then $P_{\setminus\{\gamma\}}(\vec{\sigma})$ is *non-simultaneously accepted* by the product system $G_1||_{MA}G_2$. A vector language $\vec{L} \in (\Sigma \times \Sigma_2)*$ is *non-simultaneously accepted* if all strings in $\vec{L}$ are *non-simultaneously accepted.* □

Let $L_{non-sim}(G_1||_{MA}G_2)$ be the set of all strings non-simultaneously accepted by $G_1||_{MA}G_2$. For an index set $\mathcal{A} \subseteq \{1, ..., 2\}$, the following is defined.

Definition 3.3. ($\mathcal{A}$-partial) Non-Simultaneous Acceptance

For a given (x_{01}, y_{01}), when self loops (labelled γ_i) are introduced at each goal state in Q_{m_i} in the automaton G_i, if the string $\vec{\sigma}$ is such that for each $i \in \mathcal{A}$,

$$\delta_i(x_{0i}, I\!P_i(\vec{\sigma})) \in Q_{m_i},$$

then the language $P_{\setminus\{\gamma\}}(\vec{\sigma})$ is *($\mathcal{A}$-partially) non-simultaneously accepted* by the product system $G_1||_{MA}G_2$. □

In Figure 4.8, with G_2 goal states changed to $\{y_1\}$, the vector string $\begin{bmatrix} a \\ a \end{bmatrix}\begin{bmatrix} \epsilon \\ b \end{bmatrix}$ is *non-simultaneously accepted.*

4. MA Product and Supervisory Control

4.1. Agent as Supervisor: Using the MA Product Instead of the Synchronous Product

The standard interaction for the supervisor-system pair is that of the synchronous product (see [**78**] or [**47**]). An automaton $S = (Y, \Sigma, \delta_S, y_0, Y_m)$ representing the supervisor operates with the plant, an automaton $G = (X, \Sigma = \Sigma_c \cup \Sigma_u, \delta, x_0, X_m)$, and the resulting language is the scalar synchronous product $L(S)||_sL(G)$ (see 2.4.1).

An alternative is to consider control of a system G with a supervisor S acting in unison, as an agent, leading to the combined evolution $L(S)||_{MA}L(G)$. In what follows it is assumed that all languages are prefix-closed, hence both the terms $L(S||_{MA}G)$ and $L(S)||_{MA}L(G)$ can be used equivalently (note the latter is only defined for prefix closed languages). This assumption extends to specification languages (e.g. K below).

It is also assumed that the goal states X_m and Y_m are the entire states X and Y. This has the effect of simplifying the algebraic derivations by alleviating the need for a *non-marking* condition for the supervisor (as in [47], p. 66) and isolating the controllability criteria.

The results regarding controllability of a language and the synthesis of synchronous product based supervisors apply almost directly when the MA product is used in lieu. Consider the following lemma.

LEMMA 4.1. *Let K and L be prefix-closed languages. Then,*

$$K \subseteq L \implies K||_{MA}L = K||_{MA}K = \{\begin{bmatrix} u \\ u \end{bmatrix} | u \in K\}.$$

Proof. It will be shown that $\begin{bmatrix} w \\ v \end{bmatrix} \in K||_{MA}L \implies (w = v)$.

Assume $(a \neq b)$ for $a, b, \in \Sigma$. Then,

$$\begin{bmatrix} a \\ b \end{bmatrix} \in K||_{MA}L$$
$$\iff [(a \in K \wedge b \in L) \wedge \{(a \notin L \wedge b \notin K) \vee (a = b)\}]$$
$$\iff [a \in K \wedge a \notin L \wedge b \in L \wedge b \notin K]$$

contradicting the assumption that $K \subseteq L$. Similarly, the assumption that $\begin{bmatrix} u \\ u \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} \in K||_{MA}L$ with $(a \neq b)$ also contradicts the assumption that $K \subseteq L$.

From definition 2.1, $\begin{bmatrix} u \\ u \end{bmatrix} \in K||_{MA}L$ is logically equal to $(u \in K) \wedge (u \in L)$, completing the proof . ■

The following definition is required.

Definition 4.1. [47] Σ_u-enabling

A supervisor S is *Σ_u-enabling* if the following condition holds,

$$\forall s \in \Sigma^*, \sigma \in \Sigma_u : [s \in L(S)||_s L(G) \wedge s\sigma \in L(G)] \implies s\sigma \in L(S)||_s L(G).$$

□

Theorem 4.1.

Let G be the plant and $K \subseteq L(G)$ be a prefix-closed specification. There exists a Σ_u-enabling supervisor S_{MA} such that $I\!P_2(L(S)||_{MA}L(G)) = K$ iff K is controllable w.r.t. G (i.e. $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$).

Proof. This result is established in [47] (Theorem 3.1, p. 66) for the synchronous product case, i.e. there exists a Σ_u-enabling supervisor such that $L(S)||_s L(G) = K$ if and only if K is controllable, and further, it is shown that any automaton S with $L(S) = K$ can be used as a supervisor.

To establish the result in the context of the MA-product, one may choose the same supervisor $S_{MA} = S$, and note, from Lemma 4.1, that this gives the result $I\!P_2(L(S)||_{MA}L(G)) = K$. ∎

To complete the link with [47] (Theorem 3.1, p. 66), it should also be noted that Definition 4.1 is equivalent to an MA product version with condition,

$$\forall s \in \Sigma^*, \sigma \in \Sigma_u : [s \in L(S)||_{MA}L(G) \wedge s\sigma \in L(G)] \Longrightarrow s\sigma \in L(S)||_{MA}L(G).$$

The result is that MA-product based supervisors synthesising a language K are interchangeable with synchronous product based supervisors synthesising K. Note that this holds in general only for the full observation case where the alphabet of the supervisor is equivalent to that of the plant. The partial observation case is described in the future work of Chapter 5.

4.2. Supervision of Multi-Agent System

The results in the previous subsection assumed the interaction between system and supervisor were to be based on the MA product. In this subsection the supervision is assumed to be via the standard synchronous product, but the system is now formed from the MA product of multiple systems.

Let the plant model consist of the MA product of *supervisory* automata, i.e. the model from Figure 4.1 and let the components be,

$$G_i = (X_i, \Sigma_{i_c} \bigcup \Sigma_{i_u}, \delta_i, Q_{0_i}, Q_{m_i}), i = 1, 2..., N$$

where Σ_{i_c} are the disableable events and Σ_{i_u} are the undisableable events. It is assumed that (for the case $N = 2$),

$$\Sigma_{1_c} \bigcap \Sigma_{2_u} = \emptyset, \ \Sigma_{1_u} \bigcap \Sigma_{2_c} = \emptyset,$$

which forces synchronised events to be uncontrollable in both components.

Definition 4.2. Multi-Agent Product (Supervisory Case)

$$G_1||_{MA}G_2 = (X_1 \times X_2, \Sigma_C \bigcup \Sigma_U, \delta_{MA}, Q_{0_1} \times Q_{0_2}, Q_{m_1} \times Q_{m_2})$$

where,

$$\Sigma_U = \{ \begin{bmatrix} a \\ b \end{bmatrix} \mid a \in \Sigma_{1_u} \text{ and } b \in \Sigma_{2_u} \} \tag{4.4.1}$$

$$\Sigma_C = \{ \begin{bmatrix} a \\ b \end{bmatrix} \mid a \in \Sigma_{1_c} \text{ or } b \in \Sigma_{2_c} \} \tag{4.4.2}$$

$$\delta_{MA}(\begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} u \\ v \end{bmatrix}) = \begin{cases} \begin{bmatrix} \delta_1(x,u) \\ \delta_2(y,v) \end{bmatrix} & \text{if } \delta_1(x,u)! \wedge \delta_2(y,v)! \wedge \\ & \quad [\,(u=v) \vee (\neg\delta_2(y,u)! \wedge \neg\delta_1(x,v)!)\,] \\ \text{undefined} & \text{otherwise} \end{cases}$$

□

It is emphasised that the MA product is to be interpreted in the automata sense in that (non-disabled) legal moves occur in $G_1||_{MA}G_2$ in order to generate a language $L(G_1||_{MA}G_2)$.

Let $L_1 = L(G_1)$ and $L_2 = L(G_2)$ and let S_1, S_2 be prefix closed and such that $S_1 \subseteq L_1$ and $S_2 \subseteq L_2$. Two questions will be considered,

A When is $S_1||_{MA}S_2$ controllable w.r.t. $L_1||_{MA}L_2$?
B When is $S_1||_{MA}S_2 \subseteq L_1||_{MA}L_2$?

Consider the following as a preliminary answer to question A.

Theorem 4.2. *Let L_1, L_2, S_1, S_2 be prefix closed with $S_1 \subseteq L_1$ and $S_2 \subseteq L_2$. If S_1 is controllable w.r.t. L_1 and S_2 is controllable with respect to L_2, then, $S_1||_{MA}S_2$ is controllable w.r.t. $L_1||_{MA}L_2$.*

Proof. It needs to be shown that

$$\overline{S_1||_{MA}S_2}\Sigma_U \bigcap L_1||_{MA}L_2 \subseteq \overline{S_1||_{MA}S_2}.$$

Let $\begin{bmatrix} a \\ b \end{bmatrix}$ be in Σ_U (i.e. a is in Σ_{1_u} and b is in Σ_{2_u}) and let $\begin{bmatrix} u \\ v \end{bmatrix}$ satisfy

$$\begin{bmatrix} u \\ v \end{bmatrix} \in S_1||_{MA}S_2, \quad \begin{bmatrix} u \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \in L_1||_{MA}L_2.$$

This means $u \in S_1$, $ua \in L_1$ and so $ua \in (\overline{S_1}\Sigma_{1_u} \cap L_1)$. Therefore, by the controllability of S_1, $ua \in \overline{S_1}$. Similarly, $vb \in \overline{S_2}$.
(Case $a = b$) From the definition of $||_{MA}$,

$$[\begin{bmatrix} u \\ v \end{bmatrix} \in S_1||_{MA}S_2 \wedge (ua \in S_1 \wedge vb \in S_2) \wedge (a = b)] \Longrightarrow \begin{bmatrix} u \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \in S_1||_{MA}S_2.$$

(Case $a \neq b$) Consider the case where $va \in S_2$. This implies $va \in L_2$, which implies $\begin{bmatrix} u \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \notin L_1||_{MA}L_2$, contradicting the assumption to the contrary. Similarly, $ub \in S_1$ contradicts the assumptions. The remaining case is $va \notin S_2$ and $ub \notin S_1$ and in this case,

$$[\begin{bmatrix} u \\ v \end{bmatrix} \in S_1||_{MA}S_2 \wedge (ua \in S_1 \wedge vb \in S_2) \wedge (va \notin S_2 \wedge ub \notin S_1)] \Longrightarrow \begin{bmatrix} u \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \in S_1||_{MA}S_2.$$

■

Note that if the assumption regarding the overlap of alphabets (see Equations 4.4.1 and 4.4.2) is modified so that both components must be controllable in order for a vector event to be controllable, then the converse of Theorem 4.2 holds, i.e. the controllability of the product language ensures the controllability of the image in the components.

Example 4.1. *In Figure 4.9, L_i represent the system models, S_i the specifications and E_i the maximal controllable sublanguages of S_i w.r.t. L_i. The language K is controllable w.r.t. $L_1||_{MA}L_2$ as expected from Theorem 4.2, yet $\mathbb{P}_2(K)$ is not controllable w.r.t. L_2. The calculation of controllable sublanguages of $S_1||_{MA}S_2$ is a topic of current research and discussed further in Chapter 5.* □

The assumptions in Theorem 4.2 are insufficient for a positive answer to question B. Consider the case $L_1 = S_1 = \{\epsilon, a, b\}, S_2 = \{\epsilon, c\} \subseteq L_2 = \{\epsilon, b, c\}$ where

$$S_1||_{MA}S_2 = \{\begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix}, \begin{bmatrix} a \\ c \end{bmatrix}, \begin{bmatrix} b \\ c \end{bmatrix}\} \not\subseteq L_1||_{MA}L_2 = \{\begin{bmatrix} \epsilon \\ \epsilon \end{bmatrix}, \begin{bmatrix} a \\ c \end{bmatrix}, \begin{bmatrix} b \\ b \end{bmatrix}\}.$$

A case by case analysis of how strings in the language $S_1||_{MA}S_2$ can escape the language $L_1||_{MA}L_2$ follows. Let $\begin{bmatrix} u \\ v \end{bmatrix}$ be in $S_1||_{MA}S_2 \bigcap L_1||_{MA}L_2$ and consider when $\begin{bmatrix} u \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \in S_1||_{MA}S_2$ yet $\begin{bmatrix} u \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \notin L_1||_{MA}L_2$.
(**case** $a = b$) Never. $ua \in S_1, vb \in S_2$ so $(ua \in L_1 \wedge vb \in L_2)$ which together with

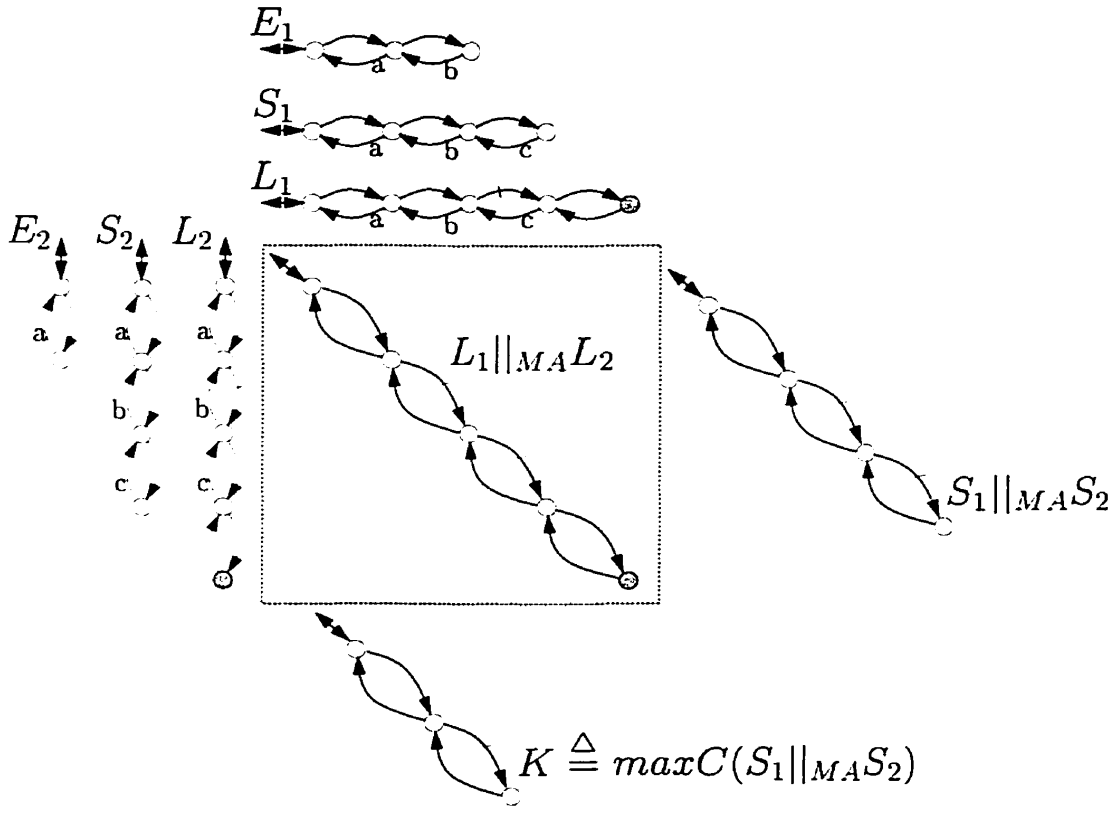


FIGURE 4.9. Controllable product does not imply controllable components.

$(a = b)$ means $\begin{bmatrix} u \\ v \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} \in L_1||_{MA}L_2$

(case $a \neq b$) $(ua \in S_1, vb \in S_2)$ implies $(ua \in L_1 \wedge vb \in L_2)$ so it must be the *coincidence condition* that succeeds for $S_1||_{MA}S_2$ but fails for $L_1||_{MA}L_2$, i.e.

$$(ub \notin S_1 \wedge va \notin S_2) \wedge (ub \in L_1 \vee va \in L_2).$$

The result is that $\begin{bmatrix} a \\ b \end{bmatrix}$ escapes $L_1||_{MA}L_2$ whenever

$$(ua \in S_1 \wedge vb \in S_2) \wedge \tag{4.4.3}$$
$$[(ub \notin S_1 \wedge va \notin S_2 \wedge ub \in L_1) \vee (ub \notin S_1 \wedge va \notin S_2 \wedge va \in L_2)]$$

These can be rewritten as the conjunction of the following two formulas,

$$(ua \notin S_1 \wedge ua \in L_1 \wedge va \in L_2) \Longrightarrow (va \notin S_2), \tag{4.4.4}$$
$$(vb \notin S_2 \wedge vb \in L_2 \wedge ub \in L_1) \Longrightarrow (va \notin S_1), \tag{4.4.5}$$

which can be interpreted as next-event rules:
(Equation 4.4.4): If Agent 1 has blocked a, a playable move, and a can also be played by Agent 2, then agent 2 must must also block a.
(Equation 4.4.5): If Agent 2 has blocked b, a playable move, and b can also be played by Agent 1, then agent 1 must must also block b.

The conclusion, for the purposes of supervisory control, is that at a state (x, y), the uninhibited next events Γ_1 and Γ_2 must be equal over the set $\Gamma^{\cap} = \{a|\delta(x,a)! \wedge \delta(x,a)!\}$. Given Γ_1 and Γ_2, there is a unique pair Γ_1' and Γ_2' of maximally enabling subsets of Γ_1 and Γ_2 satisfying this condition. These are

$$\Gamma_1' = \Gamma_1/(\Gamma_2 \cap \Gamma^{\cap})$$
$$\Gamma_2' = \Gamma_2/(\Gamma_1 \cap \Gamma^{\cap}).$$

When it is not the case that $S_1||_{MA}S_2 \subseteq L_1||_{MA}L_2$, there are unique maximal sublanguages of $S_1' \subseteq S_1$ and $S_2' \subseteq S_2$ such that

$$S_1'||_{MA}S_2' = S_1||_{MA}S_2 \bigcap L_1||_{MA}L_2.$$

These in turn may not be controllable. An iteration is suggested in Chapter 5 to find sublanguages S_1' and S_2' so that $S_1'||_{MA}S_2' \subseteq L_1||_{MA}L_2$ and at the same time $S_1'||_{MA}S_2'$ is controllable w.r.t. $L_1||_{MA}L_2$.

4.3. Centralised Control

Consider a single supervisor G_1 with language L_1 supervising a plant $G_1||_{MA}G_2$ as illustrated Figure 4.10. This yields a horizontal decomposition of the system but with centralised control. This is to be contrasted with Figure 4.1 of the introduction, in which the supervisor is also decomposed.

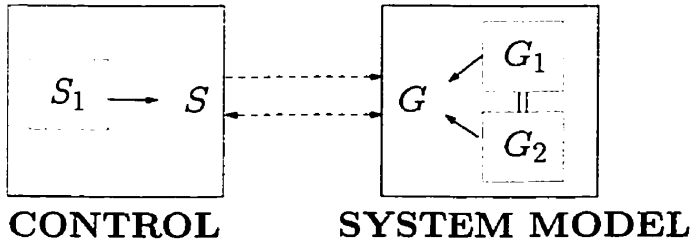


FIGURE 4.10. Centralised control of a multi-agent system model.

There is some ambiguity in the implementation of this scheme. This stems from competing interpretations of the formation of a vector language from the scalar language L_1. Two interpretations are

$$(L_1||_{MA}L_1)||_s(L_2||_{MA}L_3) \tag{4.4.6}$$

and,

$$\begin{bmatrix} L_1 \\ L_1 \end{bmatrix} ||_s(L_2||_{MA}L_3),$$

where $\begin{bmatrix} L_1 \\ L_1 \end{bmatrix} \stackrel{def}{=} \{\begin{bmatrix} u \\ w \end{bmatrix} | u, w \in L_1, |u| = |w|\}$.

Note that the synchronous product retains its standard scalar meaning in Equation 4.4.6 in the environment of vector symbols, i.e. for sets of vector symbols $\vec{\Sigma}_1, \vec{\Sigma}_2 \subseteq (\Sigma \times \Sigma)$, and languages $\vec{L}_1 \subseteq \vec{\Sigma}_1^*$ and $\vec{L}_2 \subseteq \vec{\Sigma}_2^*$,

$$\vec{L}_1||\vec{L}_2 = P_1^{-1}\vec{L}_1 \cap P_2^{-1}\vec{L}_2,$$

where $P_1 : (\Sigma \times \Sigma)^* \longrightarrow (\vec{\Sigma}_1)^*$ is defined analogously to the scalar natural projection $P_1 : \Sigma^* \longrightarrow \Sigma_1^*$.

Equation 4.4.6 reduces to the intersection (as in Lemma 4.2) when the alphabets associated with $G_1||_{MA}G_1$ and $G_2||_{MA}G_3$ ($\vec{\Sigma}_1$ and $\vec{\Sigma}_2$ respectively) are equal. This would be the case if $\Sigma_1 = \Sigma_2 = \Sigma$. Therefore consider the following lemma.

LEMMA 4.2.

$$(L_1||_{MA}L_1) \cap (L_2||_{MA}L_3) = (L_2||_{MA}L_1) \cap (L_1||_{MA}L_3).$$

Proof.

$$\begin{aligned} L_1||_{MA}L_1 \cap L_2||_{MA}L_3 &= (\begin{bmatrix} L_1 \\ * \end{bmatrix} \cap \begin{bmatrix} * \\ L_1 \end{bmatrix}) \bigcap (\begin{bmatrix} L_2 \\ * \end{bmatrix} \cap \begin{bmatrix} * \\ L_3 \end{bmatrix}) \\ &= (\begin{bmatrix} L_2 \\ * \end{bmatrix} \cap \begin{bmatrix} * \\ L_1 \end{bmatrix}) \bigcap (\begin{bmatrix} L_1 \\ * \end{bmatrix} \cap \begin{bmatrix} * \\ L_3 \end{bmatrix}) \\ &= L_2||_{MA}L_1 \cap L_1||_{MA}L_3 \end{aligned}$$

■

In the partial observation case when $\Sigma_1 \subseteq \Sigma_2 \cup \Sigma$, the vector alphabets $\vec{\Sigma}_1$ and $\vec{\Sigma}_2$ need further definition. $\vec{\Sigma}_1$ can reasonably be set equal to $(\Sigma_1 \times \Sigma_1)$ or, instead, set equal to $(\Sigma_1 \times \Sigma) \cup (\Sigma \times \Sigma_1)$. In both cases, straightforward algebraic results relating 4.4.6 to either $(L_2||_{MA}L_1)||(L_1||_{MA}L_3)$ or $(L_1||L_2)||_{MA}(L_1||L_3)$ have high complexity

(see Future works section) and it is at present unclear how these form a coherent modular supervision system.

5. Non-Simultaneous Controllability

In a more general setting of marked languages, non-simultaneously accepted languages (see Definition 3.2) must be considered. Let $E_1 \subseteq L_m(G_1)$ and $E_2 \subseteq L_m(G_2)$ be controllable languages (i.e. $\overline{E_i}\Sigma_{u_i} \cap L(G_i) \subseteq \overline{E_i}, i = 1, 2$ where the $\overline{E}$ is the prefix closure of E). Also, let

$$f_1 : \Sigma_1^* \longrightarrow 2^{\Sigma_1}$$
$$f_2 : \Sigma_2^* \longrightarrow 2^{\Sigma_2},$$

where $f_i(\sigma)$ are the uninhibited next transitions after the string σ, be such that $L_m(f_1/G_1) = E_1$ and $L_m(f_2/G_2) = E_2$, where $L_m(f/G)$ is the scalar (marked) language created by the application of the control f to the automaton G.

Definition 5.1. Non-Simultaneous Controllability

For a given (x_{01}, y_{01}), when uncontrollable self loops (labelled γ_i) are introduced at each goal state in Q_{m_i} in the the automaton G_i (labelled G_i^γ), if the vector language $\vec{L} \subseteq ((\Sigma_1 \cup \{\gamma_1\}) \times (\Sigma_2 \cup \{\gamma_1\}))^*$ is such that

(A) for all $\vec{\sigma} \in \vec{L}$, $\vec{\sigma}$ is *(non-simultaneously) accepted*

and

(B) L is controllable w.r.t. $G_1^\gamma || _{MA} G_2^\gamma$, i.e. $\overline{\overline{L}\Sigma_U \cap L(G_1^\gamma || _{MA} G_2^\gamma)} \subseteq \overline{L}$

then the language $P_{\setminus\{\gamma\}}(L)$ is *(non-simultaneously) controllable* with respect to the system $G_1 || _{MA} G_2$. □

Note that if a language is controllable in the standard sense, then it is *(non-simultaneously) controllable.*

The motivation for this notion of *non-simultaneous* acceptance and controllability is that of ensuring component-wise task completion, i.e. if $\vec{L}$ is *non-simultaneously* controllable, then the projections $I\!P_i(\vec{L})$, $1 \leq i \leq N$ contain marked strings each or which is independently accepted by the associated system G_i.

In general, it is not true that, if E_1 and E_2 are controllable w.r.t. G_1 and G_2 respectively, then the vector language (E_1, E_2) (with γ_1 and γ_2 added to ensure equal symbol counts) is *non-simultaneously controllable* w.r.t. $G_1||_{MA}G_2$. It is also not necessarily true that, when $\vec{L}$ is *non-simultaneously controllable*, the projections $I\!P_i(\vec{L})$ will themselves be controllable w.r.t. the component automata G_i.

Without further conditions, the problem of determining the controllability of languages (or finding maximal controllable sublanguages subject to given specifications) for MA product systems requires the solution of a standard supervisory problem on the vector state space.

Two possible approaches to tackling this problem are proposed. The first is through hierarchical decomposition discussed in the next section. The second is through the definition of additional conditions on component systems and their specifications such that the vector language is *non-simultaneously controllable* w.r.t. $G_1||_{MA}G_2$. The following problem can be stated:

Problem 1: Find sufficient conditions on G_1, G_2, E_1 and E_2 and vector control policy $\vec{f}$ based on f_1 and f_2 such that $I\!P_i[L(\vec{f}/G_1||_{MA}G_2] = E_i, \ \ i = 1, 2.$ □

Note these notions are similar to the notions of observability and controllability in the decentralised setting discussed in [**80, 69**].

5.1. Aggregated Hierarchical Control

The multi-agent systems defined via the MA product are subject to the explosion in state cardinality that appears in any product system. Hence there is a need for a hierarchical theory. The hierarchical theory based on state aggregation presented in Chapter 2 applies directly to the supervisory control of the MA product system, but requires the computation of the partitions with the necessary conditions, either directly or through the combination of partitions in the component systems. The two theorems on consistency from Chapter 2 will then apply and hence allow high-level specifications to be achieved via hierarchical control.

Consider the situation where the partitions π_1 and π_2 are *Trace-DC* and *(non-blocking) IBC* with respect to G_1 and G_2 respectively. It is observed that the *Trace-DC* property is not preserved, in general, for the partition $\pi_1 \times \pi_2$ of the MA product system. The same is true for the simultaneous product system and the synchronous product system (except for the case where $\Sigma_1 \cap \Sigma_2 = \emptyset$).

Problem 2: Find a sufficient condition of on G_1, G_2, π_1 and π_2 in order to preserve the properties of *Trace-DC* and *non-blocking IBC*, for the partition $\pi_1 \times \pi_2$. □

While the required additional conditions on G_1 and G_2 (and more generally, $G_1, G_2, ..., G_N$) may be stringent, it is conjectured that this will provide a methodology for the design of interacting systems. Consider, for instance, transfer lines (with shared resources and synchronised events) and traffic flow (for both communication systems and public transport), in which component systems are designed as building blocks with the a priori knowledge that they can be combined in an efficient (from the point of view of hierarchical control) and meaningful (from the point of view of verified solutions to achieving component goals) manner.

CHAPTER 5

Future Research

This chapter presents suggestions for future lines of research for each of the topics covered in this thesis.

1. Suggested Research Related to Trace-*DC* Supervisory Control

- As noted in Chapter 2, the set of (non-blocking) IBC partitions is not closed under chain union. *Prima face*, this would appear to be because, in contrast to [**15**], mutual accessibility of out-sets (i.e. the set of states that are either goal states or from which transitions lead out of the block) is not required in the (non-blocking) IBC condition. It would be of value to determine if this is in fact the case and, if not, what further conditions would be required to attain closure under chain union. If suitable conditions are established for the closure of the set of IBC partitions under chain union, it would be possible to study the resulting lattice of IBC partitions. Such a structure would be useful (as in [**15**]) for the formulation of control hierarchies.
- The notion of an *observer* appeared in [**100**] and was utilised in [**60**] and [**99**]. Essentially, $\Theta : \Sigma^* \longrightarrow T^*$, a possibly history-dependent map, is an *observer* if Θ^{-1} commutes with the prefix closure operator. A future research area is to develop the connection between the Trace-DC property, the VL algorithm and the notion of an observer.
- It was assumed in Chapter 2 that the abstract representation of a supervisory automaton would itself be a supervisory automaton. A natural extension is to consider higher-level representations where control is implemented in a

different fashion (e.g. forced-event rather than permissive). Notions of consistency between models would be of interest independently of the hierarchical framework.

Longer Term Suggested Research

- The theory is presented for *finite* automata. A topic for future research is the application of a state-partitioning methodology to the case of *infinite* state devices. These could be modelled by Petri Nets, or perhaps push-down automata. The illustrations in Chapter 3 regarding buffers would seem to indicate that (non-blocking) IBC partitions for any finite subset of the state could be developed from the model dynamics.
- A quantitative approach would be of value in the examination of the existence of (non-blocking) IBC partitions. Specifically, counting arguments or simulation could be used to check all input/output devices (Moore representations, for instance) with N internal states for the existence and number of non-trivial (non-blocking) IBC partitions. Estimates of the relative frequency of non-trivial (non-blocking) IBC partitions with respect to the number of states or connectivity could then be developed. This would be of particular interest in the extension to hybrid systems.

2. Suggested Research Related to Manufacturing Layouts

- A goal in the worked examples in Chapter 3 is the emergence of primitives that would allow for the immediate description of control methodologies for arbitrary plant layouts. As each portion is added, a new partition can be formed (as was illustrated in Figure 3.10 for the extension from one buffer to two buffers). A recursive formulation of this process of alternating extensions and re-partitioning is a suggested future line of research. It would be of interest to investigate whether paradigmatic systems emerge for this recursion.
- It was noted in Chapter 3 that, in general, the IBC property is not preserved when forming partitions on a product state space by combining partitions on component state spaces. An investigation of alternative definitions for *weak interaction* under which the IBC property is preserved would be of value. For instance, a possibility is
  Systems $(G_1||G_2\cdots||G_m)$ *and* $(G_{m+1}||G_{m+2}||\cdots G_N)$ *interact weakly if the alphabets satisfy* $\Sigma_i \cap \Sigma_j = \emptyset, i = 1,..,m-1$ *and* $j = m+1,..,N$.
  Another candidate definition for *weak interaction* can be found in [**95**]. The

product

$$(G_1||G_2\cdots||G_n) = (G_1||G_2\cdots||G_m)||(G_{m+1}||G_{m+2}||\cdots G_N),$$

can be formed by analysing $G_m||_{MA}G_{m+1}$ first, then determining their impact on the remainder of the subsystems. The motivation would be that under the condition of *weak interaction*, hierarchical control (of the product) of the component models could easily be transferred to hierarchical control of the product system without the need for verification of properties in the product state space.

This also motivates a line of research on the design of a mezzo system (an "adjudicator") which is inserted between agents, i.e. $\cdots G_m||G_{adjudicator}||G_{m+1}\cdots$. The adjudicator is then designed to negotiate prioritisation between agents.

3. Suggested Research Related to the Multi-Agent Product

- Theorem 4.2 shows that controllability is preserved in the MA product of controllable languages. This was true for the interaction condition $C^{MA}_{L_1,L_2}(w,v,a,b)$ in Definition 2.1. A possible question for future research is for which interaction conditions does the property in Theorem 4.2 hold. There is perhaps a maximal interaction (in the sense that C_1 is greater than C_2 if C_1 implies C_2) for which the preservation of controllability holds. This would be a novel approach as it would employ the form of the agent interaction as a source of control.
- It was noted in Chapter 4 that $S_1 \subseteq L_1$ and $S_2 \subseteq L_2$ does not imply $S_1||_{MA}S_2 \subseteq L_1||_{MA}L_2$. Hence even though $S_1||_{MA}S_2$ may be controllable, it may not be implementable in the system $L_1||_{MA}L_2$. This means that the diagram

$$\begin{array}{ccc} S_1, S_2 & \xrightarrow{||_{MA}} & S_1||_{MA}S_2 \cap L_1||_{MA}L_2 \\ \downarrow (*)^{\uparrow} & & \downarrow (*)^{\uparrow} \\ E_1, E_2 & \xrightarrow{||_{MA}} & E_1||_{MA}E_2 \cap L_1||_{MA}L_2 \end{array}$$

may not commute. Hence, an immediate future line of research is the problem: *Find the maximal sublanguage of* $S_1||_{MA}S_2$ *such that* $S_1||_{MA}S_2$ *is controllable*

w.r.t. $L_1||_{MA}L_2$ **and** $S_1||_{MA}S_2 \subseteq L_1||_{MA}L_2$.

If one calculates $E_1||_{MA}E_2$, where E_i is the maximal controllable sublanguage of S_i w.r.t. L_i, this may not be a subset of $L_1||_{MA}L_2$. An iterative approach might take the form:

[a] Initial $E_i^0 = S_i$

[b] Repeat

$$S_i^{k+1} = I\!P_i(E_1^k||_{MA}E_2^k \cap L_1||_{MA}L_2)$$
$$E_i^{k+1} = (S_i^{k+1})^{\uparrow} w.r.t. L_i$$

[c] Until $E_i^{k+1} = E_i^k$.

- Partial observation of one agent by the other is a natural extension in the MA product setting. State estimate sets for each agent from each other agent could be developed. It would be of value to import notions from epistemic logic in [**32**] to this framework. A formal notion such as *common knowledge* [**32**] may perhaps have important implications for observation and control.

REFERENCES

[1] *Centre for intelligent machines: Research and publications of the hierarchical, hybrid and logic control group*, http://www.cim.mcgill.ca/ ~phubbard, May 1999.

[2] *The mathworks: Developers of matlab and simulink*, http://www.mathworks.com/, October 1999.

[3] *IEEE journal of intelligent systems and their applications*, http://www.iel.ihs.com, December 1999.

[4] *IEEE working group on discrete event systems*, http://yara.ecn.purdue.edu/ echong/des_ wg/Home.html, October 1999.

[5] *UMDES software library* , http://www.eecs.umich.edu/umdes/projects/lib/umdeslib.html, May 1999.

[6] *Ratnesh kumar's home page* , ftp://kumar.ee.engr.uky.edu/pub/HTTP/index.html, May 1999.

[7] R. Alur and T.A. Henzinger, *Computer aided verification*, Lecture Notes, Deptartment of Electrical Engineering and Computer Science, UC Berkeley, 1996.

[8] A. Asarin, O. Maler, and A. Pnueli, *Reachability analysis of dynamical systems having piecewise-constant derivatives*, Theoretical Computer Science **138** (1995), 35–66.

[9] V. Badami and N. Chbat, *Home appliances get smart*, IEEE Spectrum Magazine **35** (1998), no. 8, 36–43.

[10] S. Baranov, *Logic synthesis for control automata*, Kluwer Academic Publishers, 1994.

[11] N. Biggs, *Discrete mathematics*, Oxford University Press, 1985.

[12] B. Brandin and W.M. Wonham, *Supervisory control of timed discrete-event systems*, IEEE Transactions on Automatic Control **39** (1994), no. 2, 329–341.

[13] Y.P. Brave and M. Heymann, *Control of discrete event systems modeled as hierarchical state machines*, IEEE Transactions on Automatic Control **38** (1993), no. 12, 1803–1819.

[14] P. E. Caines, P. Hubbard, and G. Shen, *Multi-agent products for finite state systems*, In preparation (1999).

[15] P.E. Caines, V. Gupta, and G. Shen, *The hierarchical control of ST-finite state machines*, Systems and Control Letters **32** (1997), 185–192.

[16] P.E. Caines, P.J. Hubbard, and G. Shen, *State aggregation and hierarchical supervisory control*, Proc. of 36th IEEE CDC (San Diego, CA), December 1997, pp. 3590–3591.

[17] P.E. Caines and Y-J Wei, *Hierarchical hybrid control systems*, Control Using Logic-Based Switching (Steve Morse, ed.), Proceedings of the Block Island Workshop, Springer Verlag, 1996, pp. 39–48.

[18] ———, *Hierarchical hybrid control systems: A lattice theoretic formulation*, IEEE Transactions on Automatic Control (1998), 501–508.

[19] P.E. Caines and Y.J. Wei, *The hierarchical lattices of a finite machine*, Systems and Control Letters **25** (1995), 257–263.

[20] K.L. Calvert, M.B. Doar, and E.W. Zegura, *Modeling internet topology*, IEEE Communications Magazine (1997), 160–163.

[21] X. Cao and Y. Ho, *Models of discrete event dynamic systems*, IEEE Control Systems Magazine (1990), 69–76.

[22] C.G. Cassandras and S. Lafortune, *Introduction to discrete event systems*, Kluwer Academic Publishers, 1999.

[23] Y-L. Chen and S. Lafortune, *Resolving feature interaction using modular supervisory control with priorities*, Feature Interactions in Telecommunications and Distributed Systems IV (P. Dini, ed.), IOS Press, 1997, pp. 108–121.

[24] C.W. Churchman, R.L. Ackoff, and E.L. Arnoff, *Operations research*, John Wiley & Sons, 1957.

[25] 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** (1988), no. 3, 249–260.

[26] J.E.R. Cury, B.H. Krogh, and T. Niinomi, *Synthesis of supervisory controllers for hybrid systems based on approximating automata*, IEEE Transactions on Automatic Control **43** (1998), no. 4, 564–568.

[27] G. Cutler and E. Kissa, *Detergency - theory and technology*, Surfactant Science Series, 1995.

[28] H. D'Angelo, M. Caramanis, S. Finger, A. Mavretic, Y. Phillis, and E. Ramsden, *Event-driven model of unreliable production lines with storage*, Int. J. Prod. Res. **26** (1988), no. 7, 1173–1182.

[29] A. A. Desrochers and R. Y. Al-Jaar, *Applications of petri nets in manufacturing systems*, IEEE Press, 1995.

[30] S. Edwards, L. Lavagno, A.L. Lee, and A. Sangiovanni-Vincentelli, *Design of embedded systems: Formal models, validation and synthesis*, Proceedings of the IEEE **85** (1997), no. 3, 366–389.

[31] M. Fabian and A. Hellgren, *Plc-based implementation of supervisory control for discrete event systems*, Proceedings of the 37th IEEE CDC (Tampa, FA), December 1998, pp. 3305–3310.

[32] R. Fagin, J. Halpern, Y. Moses, and Y. Moshe, *Reasoning about knowledge*, MIT press, 1995.

[33] J-P. Forestier and P. Varaiya, *Multilayer control of large markov chains*, IEEE Transactions on Automatic Control **23** (1978), no. 2, 298–305.

[34] S.B. Gershwin, *Hierarchical flow control: A framework for scheduling and planning discrete events in manufacturing systems*, Proceedings of the IEEE **77** (1989), no. 1, 195–209.

[35] A. Guia and F. DiCesare, *Blocking and controllability of petri nets in supervisory control*, IEEE Transactions on Automatic Control **39** (1994), no. 4, 818–824.

[36] F. Harary, R.Z. Norman, and D. Cartwright, *Structural models: An introduction to the theory of directed graphs*, John Wiley & Sons, New York, 1965.

[37] J. Hartmanis and R.E. Stearns, *Algebraic structure theory of sequential machines*, Prentice Hall, 1966.

[38] M. Heymann and F. Lin, *Discrete event control of nondeterministic systems*, Proceedings of the 35th IEEE CDC, December 1996, pp. 4445–4450.

[39] J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, 1979.

[40] P. Hubbard and P.E. Caines, *A state aggregation approach to hirarchical supervisory control with applications to a transfer line example*, Proc. of the WODES98: Workshop on Discrete Event Systems (Cagliari, Italy), IEE, August 1998.

[41] ———, *Trace-dc hierarchical supervisory control with applications to transfer-lines*, Proceedings of the 37nd IEEE Conference on Decision and Control (Tampa, FL), 1998, pp. 3293–98.

[42] ———, *Initial investigations of hierarchical supervisory control for multi-agent systems*, submitted to the 38th IEEE CDC, Pheonix, AZ, 1999.

[43] Paul Hubbard and Caines P.E., *Trace-dc hierarchical supervisory control*, Submitted to IEEE transactions on Automatic Control (1999).

[44] T. Kam, T Villa, R. Brayton, and A. Sangiovanni-Vincentelli, *Synthesis of finite state machines: Functional optimization*, Kluwer Academic Publishers, 1997.

[45] A. Koestler, *The ghost in the machine*, The Macmillan Company, 1967.

[46] R.E. Korf, *Planning as search: A quantitative approach*, Artificial Intelligence (1987), no. 33, 65–68.

[47] R. Kumar and V.K. Garg, *Modeling and control of logical discrete event systems*, Kluwer Academic Publishers, 1995.

[48] R. Kumar, V.K. Garg, and S.I. Marcus, *Predicates and predicate transformers for supervisory control of discrete event systems*, IEEE Transactions on Automatic Control **32(2)** (1993), 232–247.

[49] R.J. Leduc and W.M. Wonham, *Discrete event systems modeling and control of a manufacturing testbed*, Proceedings of the CCGEI, 1995, pp. 793–797.

[50] E.S. Lemch and P.E. Caines, *Hierarchical hybrid systems: Partition deformations and applications to the acrobot system*, Hybrid Systems: Computation and Control (T. Henzinger and S. Sastry, eds.), Lecture Notes in Computer Science, no. 1386, Springer, 1998, pp. 237–252.

[51] ———, *Hybrid partition machines with disturbances: Hierarchical control via partition machines*, Proc. of 39th IEEE CDC (Phoenix, AZ), December 1999.

[52] ———, *On the existence of hybrid models for finite state machines*, Systems and Control Letters **36** (1999), 253–259.

[53] Y. Li, *On deadlock-free modular supervisory control of discrete-event systems*, IEEE Transactions on Automatic Control **42** (1997), no. 12, 1705–1708.

[54] Y. Li and W.M Wonham, *Controllability and observability in the state-feedback control of discrete-event systems*, Proc. of 29th IEEE CDC (New York), Dec. 1988, pp. 203–208.

[55] Y. Li and W.M. Wonham, *Control of vector discrete-event systems i-the base model*, IEEE Transactions on Automatic Control **38** (1993), no. 8, 1214–1227.

[56] ———, *Control of vector discrete-event systems ii-controller synthesis*, IEEE Transactions on Automatic Control **39** (1994), no. 3, 512–531.

[57] ———, *Concurrent vector discrete-event systems*, IEEE Transactions on Automatic Control **40** (1995), no. 4, 628–638.

[58] F. Lin and H. Mortazavian, *A normality theorem for decentralized control of discrete-event systems*, IEEE Transactions on Automatic Control **39** (1994), no. 5, 1089–1093.

[59] F. Lin and W.M. Wonham, *Decentralized supervisory control of discrete-event systems*, Information Sciences (1988), no. 44, 199–224.

[60] ———, *On observability of discrete-event systems*, Information Sciences (1988), no. 44, 173–198.

[61] L. Loeb and Cochran S.D. Sanford, P.B., *Soil removal as a rate process*, Journal of American Oil Chemists' Society (1964), 120–124.

[62] T. Mackling, *Contributions to automated-theorem proving and formal methods with applications to control systems*, Ph.D. thesis, McGill University, 1997.

[63] M.M. Mano, *Digital design*, Prentice-Hall, 1984.

[64] R. Maus and J. Keyes, *Handbook of expert systems in manufacturing*, McGraw-Hill, Inc., 1991.

[65] M.D Mesarovic, D. Macko, and Y. Takahara, *Theory of hierarchical, multi-level, systems*, Academic Press, 1970.

[66] A. Messmer and M. Papageorgiou, *Automatic control methods applied to freeway network traffic*, Automatica **30** (1994), no. 4, 691-702.

[67] Faron Moller and Graham (Eds.) Birtwistle, *Logics for concurrency - structure versus automata*, Lecture Notes in Computer Science, no. 1043, Springer, 1991.

[68] ———, *Logics for concurrency - structure versus automata*, Lecture Notes in Computer Science, no. 1043, Springer, 1991.

[69] J.O. Moody and P.J. Antsaklis, *Supervisory control of discrete event systems using petri nets*, Kluwer Academic Publishers, 1998.

[70] S.G. Ostroff, *Temporal logic for real time systems*, John Wiley & Sons, 1989.

[71] C. Ozveren and A.S. Willsky, *Observability of discrete event systems*, IEEE Transactions on Automatic Control **35** (1990), no. 7, 797–806.

[72] ———, *Aggregation and multi-level control in discrete event dynamic systems*, Automatica **26** (1992), no. 3, 565–577.

[73] T.S. Perry, *In search of the future of air traffic control*, IEEE Spectrum Magazine (1997), 18–35.

[74] R. Peterson, *Decision systems for inventory management and production planning*, Wiley, 1979.

[75] J. Raisch and S.D. O'Young, *Discrete approximation and supervisory control of continuous systems*, IEEE Transactions on Automatic Control **43** (1998), no. 3, 569–572.

[76] P.J. Ramadge and W.M. Wonham, *Modular feedback logic for discrete event systems*, SIAM J. Control and Optimization **25** (1987), no. 5, 1202–1218.

[77] ———, *Supervisory control of a class of discrete event systems*, SIAM J. Control and Optimization **25** (1987), no. 1, 206–230.

[78] ———, *The control of discrete event systems*, Proceedings of the IEEE **77** (1989), no. 1, 81–98.

[79] N. Rescher, *Complexity, a philosophical overview*, Transaction Publishers, 1998.

[80] K. Rudy and W.M. Wonham, *Think globally, act locally: Decentralized supervisory control*, IEEE Transactions on Automatic Control **37** (1992), no. 11, 1692–1708.

[81] M. Sampath, *A discrete event systems approach to failure diagnosis*, Tech. report, The University of Michigan, Decmeber 1995.

[82] M. Sampath, S. Lafortune, and D. Teneketzis, *Active diagnosis of discrete-event systems*, IEEE Transactions on Automatic Control **43** (1998), no. 7, 908–929.

[83] G. Shen and P.E. Caines, *Control consistency and hierarchically accelerated dynamic programming*, Proceedings of the 37th IEEE Conference on Decision and Control (Tampa, Florida), 1998, pp. 1686–91.

[84] Herbert A. Simon, *The sciences of the artificial*, The MIT Press, 1996.

[85] R. S. Sreenivas, *On supervisory policies that enforce liveness in a class of completely controlled petri nets obtained via refinement*, IEEE Transactions on Automatic Control **44** (1999), no. 1, 173–177.

[86] R. D. Sriram, *Intelligent systems for engineering : a knowledge-based approach*, Springer, 1997.

[87] C. Strumillo and T. Kudra, *Drying: Principle, applications and design*, Gordon and Breach Science Publishers, 1986.

[88] I. Suzuki and T. Murata, *A method for stepwise refinement and abstraction of petri nets*, Journal of Computer and System Sciences **27** (1983), 51–76.

[89] S. Takai, *Optimal stat space partition for control of des with static specification*, preprint, 1999.

[90] J. Talavage and B. Elliott, *Toward a theory of hierarchical coordination and conflict*, vol. Disaggregation, problems in manufacturing and service organizations, Martinus Nijhoff Pub., 1979.

[91] J.G. Thistle, *Supervisory control of discrete event systems*, Mathematical Computer Modeling (1996), 25–53.

[92] M. Tittus and L. Bengt, *Hierarchical supervisory control for batch processes*, IEEE Transactions on Automatic Control **7** (1999), no. 5, 542–554.

[93] C. Tomlin, G.J. Pappas, and S. Sastry, *Conflict resolution for air traffic management: A study in multiagent hybrid systems*, IEEE Transactions on Automatic Control **43** (1998), no. 4, 509–521.

[94] J.N. Tsitsiklis, *On the control of discrete event dynamical systems*, Mathematics of Control Signals and Systems **2** (1989), no. 2, 95–107.

[95] T. Ushio, Y. Li, and W.M. Wonham, *Concurrency and state feedback in discreet-event systems*, IEEE Transactions on Automatic Control **37** (1992), no. 8, 1180–1184.

[96] A.F. Vaz and W.M. Wonham, *On supervisor reduction in discrete-event systems*, Int. J. Control **44** (1986), no. 2, 475–491.

[97] N. Viswanadham and T.L. Johnson, *Fault detection and diagnosis fo automated manufacturing systems*, Proceedings of the 27th IEEE CDC, 1988, pp. 2301–2307.

[98] J. Von Neumann, *Theory of games and economic behaviour*, Princeton University Press, 1953.

[99] K.C. Wong and W.M. Wonham, *Hierarchical control of discrete-event systems*, Discrete Event Dynamical Systems **6** (1996), 241–273.

[100] W.M. Wonham, *Towards an abstract internal model principle*, IEEE Transactions on Systems, Man, and Cybernetics (1976), 735–740.

[101] W.M. Wonham and P.J. Ramadge, *On the supremal controllable sublanguage of a given language*, SIAM J. Control and Optimization **25** (1987), no. 3, 637–659.

[102] ———, *Modular supervisory control of discrete event systems*, Mathematics of Control, Signal and Systems (1988), no. 1, 13–30.

[103] H. Zhong and W.M Wonham, *On the consistency of hierarchical supervision in discrete-event systems*, IEEE Transactions on Automatic Control **35** (1990), no. 10, 1125–1134.

Document Log:

Manuscript Version 0 — 24 January 2000
Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX — 24 January 2000

PAUL HUBBARD

CENTER FOR INTELLIGENT MACHINES, MCGILL UNIVERSITY, 3480 UNIVERSITY ST., MONTRÉAL (QUÉBEC) H3A 2A7, CANADA

E-mail address: phubbard@cim.mcgill.ca

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-LaTeX