ANALYSIS AND CONTROL OF DISCRETE EVENT DYNAMIC SYSTEMS: A STATE SPACE APPROACH

by Cüneyt Mehmet Özveren

S.B., Electrical Engineering, M.I.T., June 1984
S.M., Electrical Engineering and Computer Science, M.I.T., January 1987
Electrical Engineer, M.I.T., January 1987
S.M., Sloan School of Management, M.I.T., June 1989

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 1989

Signature of Author _____
Department of Electrical Engineering and Computer Science
August 28, 1989

Certified by _____
Alan S. Willsky
Thesis Supervisor

Accepted by _____
Arthur C. Smith
Chairman, Department Committee on Graduate Students

ANALYSIS AND CONTROL OF DISCRETE EVENT DYNAMIC SYSTEMS: A STATE SPACE APPROACH

by

Cüneyt Mehmet Özveren

Submitted to the Department of Electrical Engineering
and Computer Science on August 27, 1989
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Abstract

This thesis is concerned with the development of a regulator theory for discrete-event dynamic systems (DEDS), which we model using finite state automata with partially controllable and observable events. We present a notion of stability that focuses on the ability of the system to recover from errors. This leads to a theory of feedback stabilization. In addition, we develop a theory of observability and observer design for DEDS in which only certain key events are observed. Our results on stability and observers lead to a theory of stabilization via dynamic output feedback. We also present results on tracking capabilities of DEDS, i.e. on the ability of a DEDS to follow prescribed event trajectories. This involves the introduction of the notion of resiliency, i.e. of the ability of the DEDS to resume correct tracking after an error. Finally, a crucial issue in the analysis of DEDS is computational complexity. To address this problem, we describe a theory of hierarchical modeling in which sequences of events at a low level are mapped into a single event (a task) at a higher level.

Thesis Supervisor: Alan S. Willsky
Title: Professor, Department of Electrical Engineering and Computer Science

Acknowledgements

I would like to thank Professor Alan Willsky for supervising my thesis and arranging for my financial support throughout my graduate work, but especially I would like to thank him for his friendship and giving me the opportunity to taste some of the best wines in the world. I would like to thank Professor George Verghese for his continued support and friendship since my undergraduate years. I would also like to thank Professor John Tsitsiklis for his valuable comments throughout the course of this thesis. I give special thanks to the staff of the Laboratory for Information and Decision Systems at MIT and the staff of Institut de Recherche en Informatique et Systèmes Aléatoires in Rennes, France, where my research was conducted. I also wish to acknowledge Professors Panos Antsaklis, Gilead Tadmore, Jean-Claude Raoult, Albert Benveniste, Mr. Paul Le Guernic, and Mr. Bernard Le Goff for their comments during the initial phase of this research.

I would like to thank all my friends without whom my life as a graduate student would not have been as pleasant: especially, my officemates Ken, Mike, Jerry, my friends Bernard, Carlos, Huan Yu, Claude, Valerie, Regine, Gildas, Silvie, Herve whom I met in France, and my friends Aleks, Melih, Murat, Emre, Memo, Selen, Gökhan, Oya, Süreyya, Memoş, Mumuş, Selin. Finally, I would like to thank Ayşe for all her love and support.

This research has been supported by the Air Force Office of Scientific Research under Grant AFOSR-88-0032 and by the Army Research Office under Grant DAAL03-86-K0171.

Finally, I would like to express my deepest love for my parents to whom I owe everything I have.

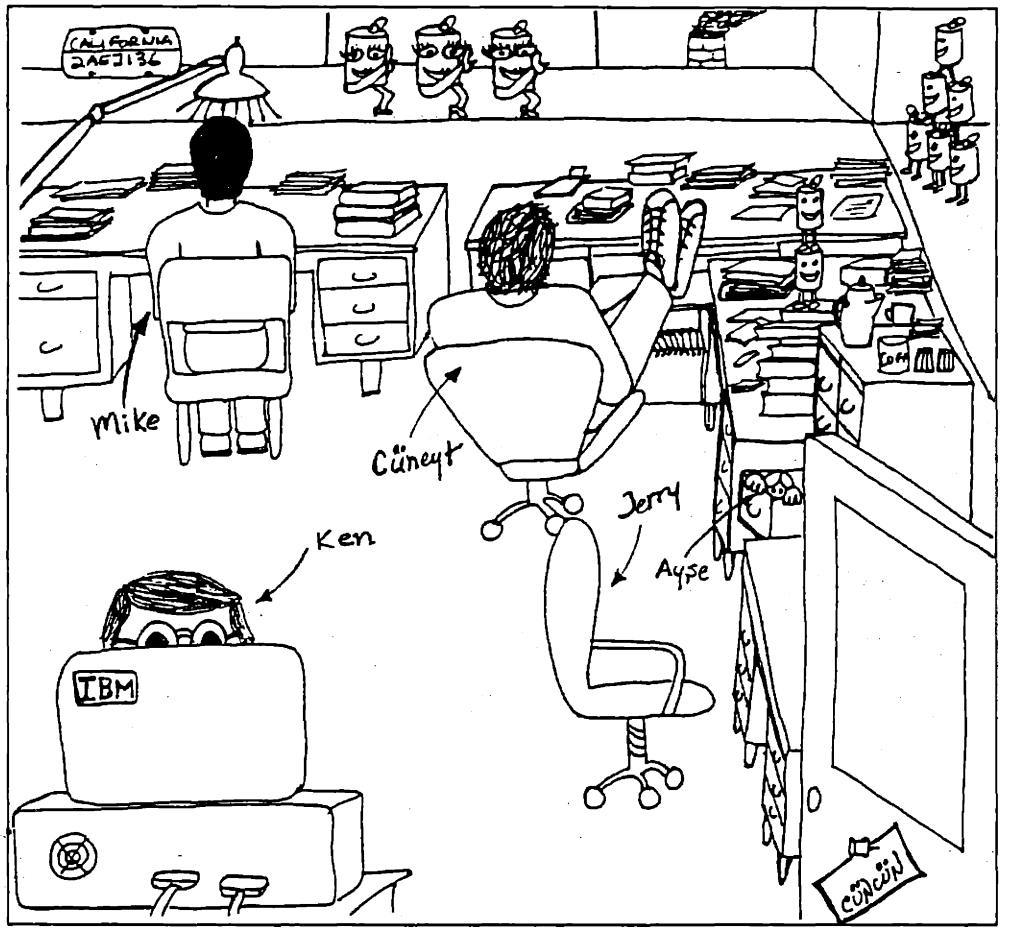

Figure 0: The Office

to peace, humanity, birth,

friends, wine, food,

and all other wonderful things in life

Contents

List of Figures

Chapter 1 Introduction

1.1 Motivation

In this thesis, we describe a framework for the analysis and control of discrete event dynamic systems (DEDS). DEDS are dynamic systems (typically asynchronous) in which the evolution of the state is triggered by the occurrence of discrete events in the system. Many large scale dynamic systems seem to have a DEDS structure, at least at some level of description. Some examples are manufacturing systems [32,42], communication systems (such as data networks, distributed systems) [10], expert systems (such as CPU design, air-traffic management) [16,19,45].

Although the analysis of DEDS has been of considerable interest to computer scientists for some time, the notion of the control of a DEDS was, to our knowledge, first explicitly introduced comparatively recently in the work of Wonham et al. [29, 34,38,39,47]. In this work, it is assumed that certain events in the system can be enabled or disabled. The control of the system is achieved by choice of control inputs that enable or disable these events. The objective is to have a closed loop system such that the event trajectory in this system is always in a given set of desired strings of events. This approach is generally classified as a linguistic approach since the objective is defined in terms of the language generated by the closed-loop system, i.e. the set of possible strings of events.

The work of Wonham et al. has prompted a considerable response by other researchers in the field, and one of the principal characteristics of this research has been the exploration of alternate formulations and paradigms that provide the opportunity for new and important developments building on the foundations of both computer science and control. The work presented here is very much in that spirit with, perhaps, closer ties to more standard control concepts. In particular, in our work we have in mind a paradigm much closer to that found in control theory, namely a regulator problem, which is concerned with tracking a reference signal in a reliable fashion. The development of such a theory requires several ingredients which we develop in this thesis. The most important ingredient is perhaps the development of a notion of stability in the context of DEDS. Like its counterpart for other classes of systems, stability for a DEDS corresponds to the evolution of the system to the desired modes of operation. Furthermore, stability serves as an important tool for formulating other notions related to error-recovery. In particular, in this thesis we introduce both a notion of resiliency, which deals with properties important in recovering from observation errors, and a notion of reliability, which deals with recovering from system failures. Such a formulation allows us to construct compensators, which, based on partial observations of the system, control the plant so that the closed loop system tracks a reference signal in spite of observation errors and system failures.

Another important issue in solving such problems in the context of DEDS is computational complexity. In particular, the computational cost of designing compensators for a complex DEDS may be quite high unless some type of structure is exploited. In order to address this issue we develop a concept of aggregation which allows us to represent chains of operations by a single high-level operation—much like the concept of words and letters in language theory—and to perform the analysis

and control at multiple levels.

In the next section, we review the work in the current literature that is related to the analysis and control of DEDS. In Section 1.3, we summarize the main contributions of this thesis and outline the organization of the rest of this thesis.

1.2 Past Work

1.2.1 Automata Theory

There is a rich literature on classical automata theory concerned with the analysis of DEDS primarily in the context of computer systems (see [1,2,5,7,15,24,25,26,33,28]). A fundamental difference between classical automata theory and ours is that in the former the events, that model the state transitions, are inputs to the system whereas we assume that the events are generated internally by the system and the inputs are the control inputs that can enable or disable *some* of these events. One of the goals of this thesis is to establish connections to various problem areas in computer science [11,13,14,40,44] to facilitate introducing the notion of control into these areas and to highlight overlaps between the two fields.

1.2.2 Wonham and Ramadge Framework

Our framework partially adopts the model of Wonham, et al. [29,34,38,39,47]. Motivated by computer networks, flexible manufacturing systems, and the start-up and shut-down procedures of industrial plants, they build a framework for the control of DEDS. The events that model the state transitions are internal to the system. Wonham, et al. introduce the notion of control by allowing a set of events to be enabled

or disabled. Specifically, they model a DEDS by an automaton A as follows:

$$A = (X, U \times \Sigma, f, x_0, X_m) \tag{1.1}$$

- X is a set of states.
- Σ is a set of events or transitions. $\Sigma_c \subset \Sigma$ is the set of controllable events that can be enabled or disabled by a control input.
- $U = \{0,1\}^{|\Sigma_c|}$ is the set of control inputs. Let $u \in U$ and $\sigma \in \Sigma_c$, if $u(\sigma) = 0$ then σ is disabled and if $u(\sigma) = 1$ then σ is enabled.
- $f : X \times U \times \Sigma \to X$ is the "controlled next state function" where f is a partial function, i.e., an event may not be defined at some state, and $f(x, u, \sigma) = y$ if σ is a transition from x to y and $u(\sigma) = 1$. The automaton is assumed to be deterministic in that f is single valued, i.e. if some event defines a transition from a state, then there is only one next state for that transition.
- x_0 is the initial state.
- X_m is the set of marked states that mark the completion of some objective, such as the completion of a part in a manufacturing system.

The approach in Wonham, et al. first requires a set of constraints that characterize the desired behavior of the DEDS. Given these constraints, they construct a desired *legal* behavior that satisfies these constraints. Then, given this legal behavior they construct a controller, termed a supervisor, so that the behavior of the closed loop system is as rich as possible but yet is guaranteed to be legal. They also address the problem of constructing minimal supervisors. In Chapter 7, we address similar problems in our context and establish connections to the Wonham and Ramadge framework.

Finally, to achieve the *marked* objectives, Wonham, et al. impose the constraint that the supervisor is such that the behavior of the closed loop system contains the marked behavior which is the set of event trajectories that take the system from a marked state to some marked state. However, this does *not* in general guarantee that any marked string ever actually occurs or any marked state is ever visited. We, in a straightforward way, address issues such as finding the supremal set such that the marked states are visited from all other states in a finite number of transitions. We achieve this by finding the supremal set stabilizable with respect to the marked states (see Chapter 2 and let E be the set of marked states).

1.2.3 Partial Observations

Cieslak, et al. [10] present the model developed by Wonham, et al. with extensions to allow for partial observations and uncertainty in the state transitions. Specifically, together with the above model, they assume an observation function $M : \Sigma \rightarrow \Delta \cup \{\epsilon\}$ which maps the transitions into a set of output symbols Δ and the 'null string' ϵ, i.e. some transitions may go unobserved. The output function we assume is also similar to this map but more restrictive in that no two events can generate the same output unless this output is ϵ. Also, Cieslak, et al. assume that A is nondeterministic in that f is set valued (we also adopt this more general model throughout this thesis). As potential areas of application, they emphasize data networks and VLSI manufacture. They are also concerned with implementation. In particular, they are concerned with the translation of the model language into the language of the system. For example, in a data communications network, a command "send packet to B" may be realized by a series of instructions: Read packet, calculate check bits, find address of B, prepare packet format, etc. This translation process is also captured by our notion of

aggregation in Chapter 8.

1.2.4 Computational Issues

The major emphasis of the work of Wonham and Ramadge is on the existence of supervisors that satisfy the requirements of the language theoretic control problems described above. However, less emphasis is given to the development of efficient algorithms for constructing the desired supervisors. Consequently, there exists a gap between their formalism and its applications to real-life problems. The work of Maimon and Tadmore [31,42,41] mostly involves finding efficient algorithms to implement control policies. In particular, they are concerned with applications to flexible manufacturing systems. All of the results in this thesis are constructive in that explicit algorithms for constructing the desired compensators are presented, and special attention is paid to the development of efficient algorithms.

Another important implementation issue in the analysis of DEDS is computational complexity. Tsitsiklis [46] addresses the issue of computational complexity for the problems formulated by Wonham, et al. and Cieslak et al. He provides the following results:

- For the case of perfect knowledge of the initial state and all the transitions, supervisor reduction is NP complete. (The problem of supervisor reduction is an attempt to minimize the cardinality of the state space of the controller designed by the approach of Wonham et al.).

- For the partial observations case:

  - The conditions, given by Cieslak, et al., for the existence of a supervisor can be tested in polynomial time,

- but Tsitsiklis gives an example of a family of systems that requires a controller that has a state space exponential in the cardinality of the system, and he shows that under partial observations, the problem of constructing the desired supervisor is in general PSPACE-complete.

This thesis shows that the approach of Wonham, et al. and Cieslak, et al. can be simplified both conceptually and computationally if the problem is considered in two parts: controller design under perfect state information, and observer design under partial observation of events. Ramadge [37], in a later work, formulates the observation problem consistently with our approach. He uses a nondeterministic automaton $A = (X, \Sigma, f)$ as the model. Specifically, he addresses the problem of determining the current state of the system from a sequence of past events and state observations. His model is equivalent to that of Cieslak, et al. if each transition is replaced by the corresponding output symbol in the model of Cieslak, et al. (also the control inputs and the initial state are ignored since Ramadge only addresses the observation problem). However, unobserved transitions are *not* modelled in Ramadge's framework, i.e. the range of the observation function does *not* include the null transition.

1.2.5 Quantitative Problems

Finally, there is also a considerable literature that addresses more quantitative problems related to DEDS (see for example [12,18,23,30]). Specifically, these deal with formulating and evaluating performance criteria that are useful in the quantitative analysis. Our work focuses on qualitative aspects and does not involve quantitative analysis directly. However, our aggregation and higher-level modelling methodology described in Chapter 8 should provide a basis for efficient quantitative analysis as they lead to extremely simple and regular higher-level models on which such quan-

titative analysis can be based.

1.3 Main Contributions and Thesis Organization

Let us now provide an outline of the main contributions and the organization of this thesis:

- **Models:** In Chapter 2, we describe the model adopted in this thesis. We start with the finite state automaton model presented in the previous section and generalize it in several directions. These include lack of initial state specification, ability to assign controllable events independently at each state, intermittent observations model, and a notion of tracking events. We also study models for compensators and how we can use a composition of the compensator and the plant to model the closed loop system.

- **Stability and stabilizability:** In Chapter 3 we develop notions of stability and stabilizability for DEDS which might, more concretely be thought of as properties of error-recovery. We present polynomial algorithms for testing stability and stabilizability, and a polynomial algorithm for constructing a stabilizing state feedback. Stability also allows us to establish connections to various notions in the computer science literature, and consequently it allows us to introduce control for such notions. Finally, much like the importance of stability in system theory, our notion of stability also plays a central role in the development of this thesis, and in fact, stability is used on various occasions in every subsequent chapter.

- **Observability, observers and resiliency:** In Chapter 4, we focus on the questions of observability and state reconstruction, given intermittent observations

of the event trajectory. A polynomial algorithm for testing observability is presented, and we show that this test is best characterized in terms of the stability of an observer. Also, while the implementation of the observer requires only polynomial time, we show that the size of the state space of an observer may, in general, be exponential in the size of the state space of the system. This fact also shows that one of the sources of the NP-completeness in the partial observation problems of interest in the DEDS context, such as output compensation studied in Chapter 5 or in the work of Cieslak et al. [10], is the cardinality of the state space of the observer. Finally, we define a notion of resiliency which allows us to characterize resilient observers which generate correct estimates in a finite number of transitions following a burst of measurement errors. Like stability, resiliency characterizes the ability of the observer to return to its desired mode of operation following a disruption.

- **Output feedback:** Chapter 5 combines the development in Chapters 3 and 4 to address a problem of stabilization by dynamic output feedback under partial observations. Specifically, we construct stabilizing compensators by cascading an observer and a stabilizing full-state feedback defined on the state space of the observer. While this is a well-established control-theoretic approach, there are several important distinguishing features of the DEDS compensation problem. First of all, unlike the case in the context of linear systems, observability together with stability by state feedback does not necessarily imply the existence of a stabilizing output compensators. Secondly, since the observers we construct for DEDS keep track of all possible states in which the DEDS can be, it is possible to re-cast the output stabilization problem as the stabilization of the observer by state feedback. Finally, it is important to characterize

the complexity in designing and implementing a stabilizing compensator. This chapter addresses these issues and also presents our treatment of the problem of resilient output stabilization.

- **Invertibility:** In Chapter 6, we address a problem of event trajectory reconstruction, which we term invertibility. Such a problem may arise in the monitoring of a complex system or in post-mortem analysis or troubleshooting. Also, the dual of this problem, the generation of input sequences to achieve specified output sequences is of considerable interest in characterizing the tracking and regulation capabilities of a DEDS (see the next item). In addition, as we will see, the error behavior of an inverter for a DEDS can be quite non-resilient. In particular, much as in catastrophic error propagation in sequential decoding [36], some DEDS inversion problems have the undesirable property that a finite burst of observation errors can lead to an unbounded sequence of inversion errors. The development of our notion of resilient invertibility allows us to characterize systems for which we can construct inverters that do not exhibit this undesirable property.

- **Tracking and Restrictability:** In Chapter 7, we focus on analyzing the tracking capabilities of a DEDS. Tracking is roughly the dual of invertibility, as it is concerned with the construction of a compensator in order to force the output trajectory to follow a specified string. We also formulate a notion of restrictability—a slight generalization of the notion of controllability of Wonham and Ramadge in [39]—which is defined as the ability to control the system so that its behavior is in a desired language defined over the tracking alphabet. A particularly important notion that we also formulate in Chapter 7 is a notion of reliability which is closely related to stability. In particular, reliable

restrictability allows us to characterize the ability of the system to return to the desired, restricted behavior within a finite time following a burst of system failures.

- **Aggregation and multi-level control:** In Chapter 8, we develop an aggregation scheme which allows us to characterize the behavior of a system in terms of a set of primitives. This higher-level characterization allows us to represent sets of states by a single state and sets of strings by a single event. We thus achieve both a spatial and a temporal aggregation. We then consider systems that consist of a number of subsystems and we show that the overall system can be expressed by the interaction of the higher-level models of each component. Yet higher-level representations can also be achieved by using this aggregation scheme. We illustrate this notion, in the context of flexible manufacturing systems, via tasks and procedures as primitives. This aggregation scheme also addresses an important aspect of the problem of computational complexity in the DEDS framework.

Finally, in Chapter 9, we present the conclusions of this thesis and outline directions for future work.

Chapter 2
Modelling DEDS

2.1 Basic Model

The class of systems we consider are nondeterministic finite-state automata with intermittent event observations. The basic object of interest is the quintuple:

$$G = (X, \Sigma, U, \Gamma, \Xi) \tag{2.1}$$

where X is the finite set of states, with $n = |X|$[1], Σ is the finite set of possible events, U is the set of admissible control inputs consisting of a specified collection of subsets of Σ, corresponding to the choices of sets of controllable events that can be enabled; $\Gamma \subset \Sigma$ is the set of observable events, and $\Xi \subset \Sigma$ is the set of tracking events, i.e., events whose sequential behavior we will wish to track. The dynamics defined on G are of the form:

$$x[k+1] \in f(x[k], \sigma[k+1]) \tag{2.2}$$

$$\sigma[k+1] \in (d(x[k]) \cap u[k]) \cup e(x[k]) \tag{2.3}$$

Here, $x[k] \in X$ is the state after the kth event, $\sigma[k+1] \in \Sigma$ is the $(k+1)$st event, and $u[k] \in U$ is the control input after the kth event. The function $d : X \to 2^{\Sigma}$ is a set-valued function that specifies the set of possible events defined at each state

[1]The notation $|X|$ denotes the cardinality of X.

(so that, in general, not all events are possible from each state), $e : X \to 2^{\Sigma}$ is a set valued function that specifies the set of events that cannot be disabled at each state, and the function $f : X \times \Sigma \to 2^{X}$ is also set-valued, so that the state following a particular event is not necessarily known with certainty. The functions d, e, and f are extended to act on sets of states in a straightforward fashion. For example, given $Q \subset X$, $d(Q)$ is defined as $\bigcup_{x \in Q} d(x)$. The function f is also extended to act on sets of events so that given $Q \subset X$ and $S \subset \Sigma$, $f(Q, S)$ is defined as $\bigcup_{x \in Q, \sigma \in S} f(x, \sigma)$ where we let $f(x, \sigma) = \emptyset$ if $\sigma \notin d(x)$. Without loss of generality, we assume that $e(x) \subset d(x)$ for all x. The set $d(x)$ represents an "upper bound" on the set of events that can occur at state x, whereas the set $e(x)$, is a lower bound. The effect of our control action is adjusting the set of possible events between these bounds, by disabling some of the controllable events, i.e., elements of the set $d(x) \cap \overline{e(x)}$. Note that in this general framework, there is no loss of generality in taking $U = 2^{\Sigma}$. Also, by appropriate choice of $e(x)$, we can model situations in which we have enabling/disabling control over some events only at certain states. In the earlier chapters of this thesis, we will not use all parts of this complete model. For example, in the beginning of the next chapter, since we will be concerned with analyzing the uncontrolled system with no outputs, we will define systems over only X and Σ. At the beginning of every chapter, we indicate the model that we will use in that chapter. Also, in calculating the complexity of algorithms that we present in this thesis, we will assume that the number of transitions defined at each state, $|f(x, \Sigma)|$ for each $x \in X$, is small. It is otherwise straightforward to recompute the complexity of algorithms in order to account for $|f(x, \Sigma)|$.

Our model of the output process is quite simple: whenever an event in Γ occurs, we observe it; otherwise, we see nothing. Specifically, we define the output function

$h : \Sigma \to \Gamma \cup \{\epsilon\}$, where ϵ is the "null transition", by

$$h(\sigma) = \begin{cases} \sigma & \text{if } \sigma \in \Gamma \\ \epsilon & \text{otherwise} \end{cases} \tag{2.4}$$

Then, our output equation is

$$\gamma[k+1] = h(\sigma[k+1]) \tag{2.5}$$

In a very simple way, h can be thought of as a map from strings over Σ to strings over Γ. That is, if we let Σ^* denote the set of all strings of finite length with elements in Σ, including the empty string ϵ, then $h : \Sigma^* \to \Gamma^*$ so that $h(s_1 s_2) = h(s_1)h(s_2)$. Since we will extend other functions in this way, let us introduce the following function to characterize such extensions: Given a string $s \in \Sigma^*$, suppose that $s = p\sigma$, where $p \in \Sigma^*$ and $\sigma \in \Sigma$. We define $s \downarrow \Gamma$, the projection of s into Γ^*, via the following recursive definition:

$$s \downarrow \Gamma = \begin{cases} (p \downarrow \Gamma)\sigma & \text{if } \sigma \in \Gamma \\ p \downarrow \Gamma & \text{otherwise} \end{cases} \tag{2.6}$$

and $s \downarrow \Gamma = \epsilon$ if $s = \epsilon$. Then, the extension of h to the domain Σ^* can be simply expressed by $h(s) = s \downarrow \Gamma$ for $s \in \Sigma^*$. Finally, we extend h to act on sets of strings over Σ as follows: Given $L \subset \Sigma^*$, $h(L)$ is defined as $\bigcup_{s \in L} h(s)$.

The set Ξ, which we term the tracking alphabet, represents events of interest for tracking purposes. This formulation allows us to define tracking over a selected alphabet so that we do not worry about listing intermediary events that are not important in tracking. We use $t : \Sigma^* \to \Xi^*$, to denote the projection of strings over Σ into Ξ^*, i.e., $t(s) = s \downarrow \Xi$ for $s \in \Sigma^*$. Similar to h, we also extend t to act on sets of strings over Σ.

The hextuple $A = (G, f, d, e, h, t)$ representing our system can also be visualized graphically as in Figure 2.1. Here, circles denote states, and events are represented

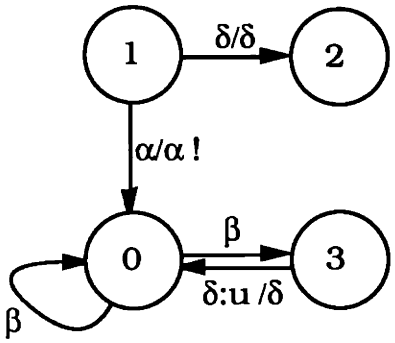

Figure 2.1: A Simple Example

by arcs. The first symbol in each arc label denotes the event, while the symbol following "/" denotes the corresponding output (if the event is observable). Finally, we mark the controllable events by ":u" and tracking events by "!". Thus, in this example, $X = \{0,1,2,3\}$, $\Sigma = \{\alpha,\beta,\delta\}$, $\Gamma = \{\alpha,\delta\}$, $\Xi = \{\alpha\}$, and δ is controllable at state 3 but not at state 1. Also, $d(1) = e(1) = \{\alpha,\delta\}$, $d(3) = \{\delta\}$, $e(3) = \emptyset$, $f(0,\beta) = \{0,3\}$ etc. A transition, also denoted as $x \rightarrow^{\sigma} y$, consists of a source state, x, an event, $\sigma \in d(x)$, and a destination state, $y \in f(x,\sigma)$.

In general, the automaton A is a nondeterministic finite state automaton. If $f(x,\sigma)$ is single-valued for each $x \in X$ and $\sigma \in d(x)$, then A is termed a deterministic finite state automaton. In general, if $f(x,\sigma) = \{y\}$ for some $x, y \in X$ and $\sigma \in d(x)$, then, for notational simplicity, we will write $f(x,\sigma) = y$.

In the subsequent sections, we use the following terminology concerning state and event trajectories:

- A finite string of states, $\mathbf{x} = x_0 x_1 \cdots x_j$ is termed a path or a state trajectory from x_0 if $x_{i+1} \in f(x_i, d(x_i))$ for all $i = 0 \cdots j-1$. We say $x \in \mathbf{x}$ if $x_i = x$ for some i. Let $\mathcal{X}(A,x)$ denote the set of all possible paths from x. A path is termed

a cycle if $x_0 = x_j$ and a cycle is termed a primary cycle if there exists no distinct pair $i_1, i_2 \in 0 \cdots j-1$ such that $x_{i_1} = x_{i_2}$, i.e., if it contains no other cycles. For example, in Figure 2.1, 12, 3003, and 030 are all paths, 3003, and 030 are cycles, and 030 is a primary cycle. In general, there may be infinitely many cycles, but only a finite number of primary cycles. For example, the primary cycles of Figure 2.1 are 00, 030, and 303.

- Similarly, a finite string of events $s = \sigma_1 \cdots \sigma_j$ is termed an event trajectory from $x \in X$ if $\sigma_1 \in d(x)$ and $\sigma_{i+1} \in d(f(x, \sigma_1 \cdots \sigma_i))$ for all i, where we extend f to Σ^* via

$$f(x, \sigma_1 \cdots \sigma_i) = f(f(x, \sigma_1 \cdots \sigma_{i-1}), \sigma_i)$$

with $f(x, \epsilon) = x$. In Figure 2.1, $\alpha\beta\beta\delta$ is an event trajectory. Let $L(A, x)$ denote the set of all possible event trajectories starting from state x, and let $L(A) = \bigcup_{x \in X} L(A, x)$.

2.2 Forced Events

A class of events, different from those defined above, has been introduced by Golazewski and Ramadge in [20]. These events, termed forced events, can be forced to occur instantaneously regardless of the other events defined at the current state and in fact can only occur if they are forced. Let us now show that we can model such events in our present context. Given $x \in X$ let $d_1(x)$ denote the set of forced events defined at x and let $d_2(x)$ denote the other events (controllable or uncontrollable) defined at x. We introduce a new controllable event μ and a new state x' as follows: We redefine $d(x)$ as $d_1(x) \cup \mu$ so that all events defined at x are now controllable and we define $f(x, \mu) = x'$. Also, we define $d(x') = d_2(x)$ so that $f(x', \sigma) = f(x, \sigma)$ for

all $\sigma \in d_2(x)$. If in addition, we impose the restriction that only one event can be enabled at a time at state x, then we can treat forced events as controllable events in our framework. Thus, if we decide to force an event at x, then we enable only that event, and if we decide not to force any events, then we enable μ. Forced events will be useful in the development of Chapter 8.

2.3 Languages

A collection of strings $L \subset \Sigma^*$ is termed a language over the alphabet Σ (see [39]). For example, given $x \in X$, $L(A, x)$ is a language over Σ which we refer to as the language generated by x in A. In Figure 2.1, $L(A, 0)$ can be expressed as $\{\beta, \beta\delta\}^*$. In some cases, it will be more convenient to use the algebraic notation $(\beta + \beta\delta)^*$, where "+" denotes the union of β and $\beta\delta$.

A language is termed a regular language if it can be expressed by using the operations of concatenation, $\cup$ or $+$, and * on a finite set of strings in Σ^* a finite number of times (see [25]). In Figure 2.1, $L(A, 0)$ is a regular language. On the other hand, the language $\{\alpha^i\beta^i | \alpha, \beta \in \Sigma, i \in \mathbf{N}\}$, where $\mathbf{N}$ is the set of natural numbers, is not a regular language. Given a regular language L, a deterministic finite state automaton A with a specified initial state $x_0 \in X$, (A, x_0) is termed a recognizer for L if there exists a set of final states $X_F \subset X$ such that for all $s \in L$, $f(x_0, s) \in X_F$ (see [25]). We will assume that all states in the recognizer state space X can be reached from the initial state x_0 of the recognizer and all states in X can reach a final state, i.e., for all $x \in X$ there exists $s \in L(A, x_0)$ such that $x = f(x_0, s)$ and there exists $p \in L(A, x)$ such that $f(x, p) \in X_F$. Finally, for any regular language L, there exists a recognizer which satisfies the above properties and furthermore, there exists a unique recognizer with the least possible number of states (see [25]). Such a

recognizer is termed minimal.

Given a string $s \in L$, we use $|s|$ to denote the length of s. If $s = pqr$ for some p, q and r over Σ then we say that p is a prefix of s and r is a suffix of s. We also use s/pq to denote the suffix r. Also, we say that q is a substring of s. Finally, we need the following characterization of the notion of liveness and completeness in the context of languages (see Chapter 7 and Chapter 8):

Definition 2.1 Given L, $s \in L$ has an infinite extension in L if for all integers $i \geq |s|$, there exists $r \in L$, $|r| = i$ such that s is a prefix of r. L is prefix closed if all the prefixes of any $s \in L$ are also in L. L is a complete language if each string in L has an infinite extension in L and L is prefix closed. ❑

For any language L, we let L^c denote the prefix closure of L, i.e.

$$L^c = \{p \in \Sigma^* | p \text{ is a prefix of some } s \in L\} \tag{2.7}$$

2.4 Range of a State and Liveness

Let us now characterize the range of a state: First, recall that $\rightarrow^\sigma$ denotes a transition labeled by σ. Similarly, we let $\rightarrow^s$ denote a string of transitions s, and we let $\rightarrow^*$ denote any number of transitions, including no transitions. Then, the range of a state x is defined by

$$R(A, x) = \{y \in X | x \rightarrow^* y\} \tag{2.8}$$

representing the set of states that can be reached from x. Note that, $R(A, x)$ always includes x. For example, in Figure 2.1, the range of 1 is X itself. Also, given $Q \subset X$, we let $R(A, Q) = \bigcup_{x \in Q} R(A, x)$.

Let us now present an algorithm for computing the range of a set of states. It immediately follows from the definition that the range of a set of states, X_0, is the fixed point of $R = f(R, \Sigma)$ such that $R \supset X_0$. Thus, we have the following algorithm:

Proposition 2.2 The following algorithm computes $R(A, X_0)$ for any $X_0 \subset X$ and it has complexity $O(n)$:

Algorithm Let $R_0 = Q_0 = X_0$ and iterate:

$$R_{k+1} = R_k \cup f(Q_k, \Sigma)$$

$$Q_{k+1} = R_{k+1} \cap \overline{R_k}$$

Terminate when $R_{k+1} = R_k$. Then, $R(A, X_0) = R_k$.

Proof: Clearly, the algorithm terminates in a finite number of steps, say r, and $R_r = R(A, X_0)$. Since each state is visited only once, the complexity of the algorithm is $O(n)$. ❑

For example, in Figure 2.1, in order to compute the range of 1, we have: $R_0 = \{1\}, R_1 = \{0, 1, 2\}, R_2 = X, R_3 = X$, and the algorithm terminates. Thus, the range of 1 is X.

Note that in computing the complexity of above algorithm, we have not considered the number of transitions defined at each state. Specifically, if we let n_s be the maximum of $|f(x, \Sigma)|$ over all $x \in X$, then the complexity of this algorithm is $O(nn_s)$. However, as we have stated, we assume that n_s is small and we say that the complexity is $O(n)$. We will make this assumption throughout this thesis.

Given a set $D \subset X$, an important problem of interest is to compute the maximal set of states that avoid D. This situation arises, for example, in mutual exclusion problems, in the context of manufacturing systems, [31], and computer systems, [2, 4,13], where a number of users compete for a limited number of resources, say r. In

that case, the states that represent $p > r$ users attempting to use the resources are undesirable and one wants to find the maximal set of states, X_D, from which one can avoid this condition.[2]

In order to characterize X_D let us introduce an automaton A^{-1} which denotes A with the transitions reversed, i.e., $A^{-1} = (G, f^{-1}, d^{-1}, e^{-1}, h, t)$ where:

$$f^{-1}(x,\sigma) = \{y \in X | x \in f(y,\sigma)\} \tag{2.9}$$

$$d^{-1}(x) = \{\sigma \in \Sigma | \exists y \in X \text{ such that } x \in f(y,\sigma)\} \tag{2.10}$$

$$e^{-1}(x) = \{\sigma \in \Sigma | \exists y \in X \text{ such that } x \in f(y,\sigma) \text{ and } \sigma \in e(y)\} \tag{2.11}$$

Then,

$$X_D = \overline{R(A^{-1}, D)} \tag{2.12}$$

where $\overline{S}$ denotes the complement of the set S.

As we will see, X_D is also useful in testing liveness: We say that a system is alive if it can never reach a point at which no event is possible:

Definition 2.3 A state $x \in X$ is alive if $d(y) \neq \emptyset$ for all $y \in R(A,x)$. A subset Y of X is termed a live set if all $x \in Y$ are alive. A system A is termed alive if X is a live set. ❑

Let

$$D_a = \{x \in X | d(x) = \emptyset\} \tag{2.13}$$

denote the set of states which have no events defined, and term these the dead states. For example, in Figure 2.1, $D_a = \{2\}$ and 0 is alive, whereas 1 is not. Clearly, the class of live sets is closed under arbitrary unions and intersections. The maximal live

[2]Golazewski and Ramadge, [21], address this problem, in the context of DEDS problems that consist of many interacting components. Our contributions to this problem are presented at the end of Chapter 3.

subset of X, X_a, is given by the set of states that cannot reach the dead states, i.e., $X_a = \overline{R(A^{-1}, D_a)}$. For example, in Figure 2.1, $D = \{2\}$ and $X_a = \{0, 3\}$. Note that the state 1 is not alive since there exists a trajectory from 1 which goes to state 2, which is a dead state.

2.5 Composition

Another important notion that we need in the analysis of DEDS is the composition of two automata, $A_i = (G_i, f_i, d_i, e_i, h_i, t_i)$ which share some common events. Specifically, let $S = \Sigma_1 \cap \Sigma_2$ and, for simplicity, assume that $\Gamma_1 \cap S = \Gamma_2 \cap S$ (i.e., any shared event observable in one system is also observable in the other), and $\Xi_1 \cap S = \Xi_2 \cap S$. The dynamics of the composition are specified by allowing each automaton to operate as it would in isolation except that when a shared event occurs, it must occur in both systems. Mathematically, we denote the composition by $A_{12} = A_1 \parallel A_2 = (G_{12}, f_{12}, d_{12}, e_{12}, h_{12}, t_{12})$, where

$$G_{12} = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, 2^{\Sigma_1 \cup \Sigma_2}, \Gamma_1 \cup \Gamma_2, \Xi_1 \cup \Xi_2) \tag{2.14}$$

$$f_{12}(x, \sigma) = f_1(x_1, \sigma) \times f_2(x_2, \sigma) \tag{2.15}$$

$$d_{12}(x) = (d_1(x_1) \cap \overline{S}) \cup (d_2(x_2) \cap \overline{S}) \cup (d_1(x_1) \cap d_2(x_2)) \tag{2.16}$$

$$e_{12}(x) = (e_1(x_1) \cap \overline{S}) \cup (e_2(x_2) \cap \overline{S}) \cup (e_1(x_1) \cap e_2(x_2)) \tag{2.17}$$

$$h_{12}(\sigma) = \begin{cases} h_1(\sigma) & \text{if } \sigma \in \Gamma_1 \\ h_2(\sigma) & \text{if } \sigma \in \Gamma_2 \\ \epsilon & \text{otherwise} \end{cases} \tag{2.18}$$

$$t_{12}(\sigma) = \begin{cases} t_1(\sigma) & \text{if } \sigma \in \Xi_1 \\ t_2(\sigma) & \text{if } \sigma \in \Xi_2 \\ \epsilon & \text{otherwise} \end{cases} \tag{2.19}$$

Here we have extended each f_i to all of $\Sigma_1 \cup \Sigma_2$ in the trivial way, namely, $f_i(x_i, \sigma) = x_i$ if $\sigma \notin \Sigma_i$. Note also that h_{12} and t_{12} are well-defined.

2.6 Compensators

In the development of this thesis, we will use three kinds of compensators for controlling DEDS. Let us now define these compensators. In Chapter 3, when we address problems of stabilizability, we use a state feedback law which we define as a map $K : X \to U$. Given a state feedback K, we let $A_K = (G, f, d_K, e)$ denote the closed loop system where

$$d_K(x) = (d(x) \cap K(x)) \cup e(x) \tag{2.20}$$

The compensators we use in Chapter 7 are defined by $C : X \times \Sigma^* \to U$ specifying the set of controllable events that are enabled given the current state and the entire event trajectory up to present time. Given such a compensator C, the closed loop system A_C is the same as A but with

$$\sigma[k+1] \in d_C(x[k], s[k]) \triangleq (d(x[k]) \cap C(x[k], s[k])) \cup e(x[k]) \tag{2.21}$$

where $s[k] = \sigma[0] \cdots \sigma[k]$ with $\sigma[0] = \epsilon$. Here we have used somewhat modified notation in that we allow d_C to depend both on $x[k]$ and $s[k]$. It is not difficult to show that we can always write A_C as an automaton (with corresponding "d") depending only on the state, which will take values in an expanded state space, representing the cross-product of the state spaces of A and C. For an arbitrary choice of C, its state space (i.e., an automaton realizing the desired map) may be infinite. As we will see, for our purposes we can restrict attention to compensators which can be realized by finite state machines (see Chapter 7).

Finally, the output compensators we use in the context of partial observations of Chapter 5 and Chapter 8 are defined as maps of the form $C : \Gamma^* \rightarrow U$. Then, the closed loop system A_C is the same as A but with:

$$\sigma[k+1] \in d_C(x[k], s[k]) \triangleq (d(x[k]) \cap C(h(s[k]))) \cup e(x[k]) \tag{2.22}$$

In subsequent chapters, we will see that the interaction of these compensators with A can also be represented as the composition of A and a finite state automaton which realizes the particular compensator.

Chapter 3
Stability

3.1 Motivation

As discussed in Chapter 1, the goal in the work of Wonham, Ramadge, et al. is to restrict the behavior of the system so that all strings generated from the given initial state are in a given set of "legal" strings. In a sense, one interpretation of some of the work we present is to develop control methods for re-establishing such legal behavior following the occurrence of one or more anomalous events. For example, a manufacturing system is always subject to failures. Thus, in the Wonham and Ramadge context, one would include all possible strings with failures and successful recoveries in the legal language (the set of legal strings). In our formulation, we focus on states rather than strings. Specifically, assume that we have identified the set of "good" states, i.e., the set of initial states, from which only legal strings are generated. Our focus then is to test if all trajectories from other states visit the "good" states infinitely often, so that the system recovers from any possible error in a finite number of transitions. If, in fact, failures do not happen very often, then the system will exhibit legal behavior "most of the time".

Another goal of this chapter is to establish connections with related notions in computer science. In particular, the concept of stability we use here has been introduced by researchers in a number of different computer science contexts. What

distinguishes our work and makes it of potential interest in computer science as well as in control theory is the introduction of control and feedback to formulate and solve the problem of stabilizing systems. For example, our notion of pre-stability and the algorithm we provide are exactly the same as the notion of inevitable reachability and the associated algorithm of Sifakis [40]. Other notions of Sifakis can be characterized using our concepts of stability and transition-function-invariance (f-invariance of Section 3.2.2). Thus, our results in stabilizability can be directly applied to his notions if control were to be introduced in his framework.

Our final goal in this chapter is to establish ties with the well-known control/systems concepts such as stability, A-invariance, etc., in the DEDS framework. As in systems theory, we will see that stability plays a central role in various problems of interest. In fact, our notion of stability is critical to the development in all of the subsequent chapters of this thesis. In particular, the notions of observability introduced in Chapter 4 and of eventual tracking and eventual restrictability introduced in Chapter 7 can be characterized using stability. Furthermore, the notion of resiliency formulated in Chapters 4 and 6, and the notion of reliability formulated in Chapter 7 are concepts of error-recovery much like our notion of stability. Finally, our notions of invariance play a critical role in Chapter 7.

3.2 Stability

In this section, we define our notion of stability and provide an algorithm that tests stability. Since we concentrate here on the uncontrolled system with perfect knowledge of the state and event trajectories, we only need to use a portion of the complete model introduced in the previous chapters. Specifically, the systems we consider in this section are of the form $A = (G, f, d)$ where $G = (X, \Sigma)$. The dynamics in this

case are:

$$x[k+1] \in f(x[k], \sigma[k+1]) \tag{3.1}$$

$$\sigma[k+1] \in d(x[k]) \tag{3.2}$$

3.2.1 Pre-Stability

The notion of stability we wish to capture can be thought of as a concept of error recovery. Specifically, as in Wonham and Ramadge, one can imagine a set of desired event trajectories for a DEDS. For example, in a manufacturing system, these sequences might consist of a concatenation of subsequences each of which corresponds to the successful production of an individual component and the return of the system to a "start-up" state, from which it can initiate the next production task. Because of the possibility of failures or errors, actual behavior may deviate from this ideal, and what one would like is that after such a failure, the system recovers. To capture this idea, we suppose that we have identified a subset E, of the state space X, so that returning to E corresponds to being in a position to continue desired behavior from that point on. For example, in a manufacturing system, E might be the set of all non-failure states or it might simply be the set of start-up states, which, in the Wonham and Ramadge terminology, can be thought of as the possible initial states from which legal event trajectories are generated. Error recovery or stability then corresponds to a return to E in a finite number of transitions following any excursion out of E. There is a useful linguistic interpretation of this concept. Define the desired language as

$$L(A, E) = \bigcup_{x \in E} L(A, x) \tag{3.3}$$

What we would like is the following. Suppose that for some reason, we are in a state $x \notin E$. Then, we want all possible event trajectories from x to differ from a desired

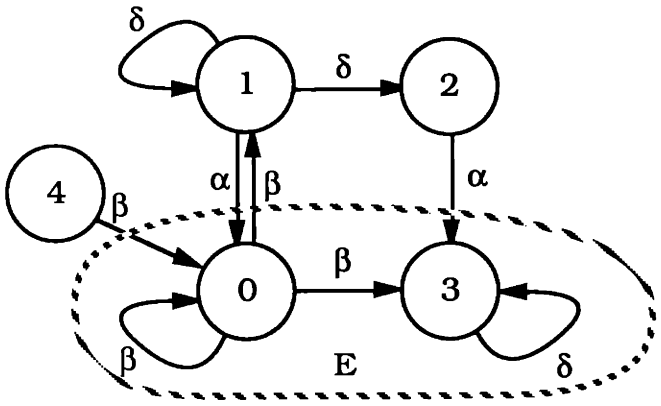

Figure 3.1: Stability Example

trajectory by at most a finite prefix.

Given E, we define a state $x \in X$ to be E-stable if all paths from x go through E in a finite number of transitions and then visit E infinitely often. For example, in Figure 3.1, where $E = \{0, 2\}$, only 2 and 3 are stable states. State 1 is not stable since the system can loop at 1 for an infinite number of transitions. States 0 and 4, although 0 is in E, are not stable either since the system can make a transition to state 1 and stay there forever. This notion is similar to but not exactly the same as the notion of Büchi acceptance [44]: An infinite event trajectory is Büchi acceptable if there exists a corresponding state trajectory which visits a specified set of "terminal state" infinitely often. In our notion of stability, if we let E be the set of terminal states, we require that all possible state trajectories, from a stable state, visit E infinitely often.

Our notion of stability is captured in two stages. We term x pre-stable if all paths from x go to E in a finite number of transitions. In other words, no path from x ends up in a cycle that does not go through E. For example, in Figure 3.1, 0, 2, 3, and 4

are pre-stable. This notion is exactly the same as the notion of inevitable-reachability of Sifakis [40]. A state is then stable if all the states in its reach are pre-stable. In Figure 3.1, 0 and 4 are not stable since they can reach 1, which is not pre-stable.

We formalize pre-stability as follows:

Definition 3.1 Given a live system A and some $E \subset X$, a state $x \in X$ is pre-stable with respect to E (or E-pre-stable) if for all $\mathbf{x} \in \mathcal{X}(A, x)$ such that $|\mathbf{x}| \geq n$, there exists $y \in \mathbf{x}$ such that $y \in E$. ❑

We say that a set of states is E-pre-stable if all its elements are pre-stable and a system A is E-pre-stable, if all states in X are pre-stable.

In the above definition, we require liveness in order for a system to be stable. However, on occasion (see, for example, Section 3.4.2), it is useful to allow trajectories to die provided that they die in E. To address this issue, suppose that A is not alive but all the dead states of A are in E. Then, let A' be an automaton which is the same as A but with a self-loop at each dead state of A. Then, it is straightforward to show that all the trajectories in A go to E in a finite number of transitions iff A' is E-pre-stable. Therefore, all of our results in this chapter also hold for this slightly more general notion of pre-stability. However, for the sake of simplicity, we will continue to assume that A is alive in presenting these results.

If all paths from x go through E, then they do so in less than n transitions since otherwise x has a a cycle that does not go through E, and thus, there exists a path which never goes through E. Equivalently, pre-stability can be characterized in terms of the primary cycles. To formalize this, let $R_E(A, x)$ denote the set of states that can be reached by trajectories from x that do not go out of E if they enter E at all. For example, in 3.1, $R_E(A, 4) = \{0, 3, 4\}$. In other words, $R_E(A, x) = R(A', x)$ where A' is an automaton created from A by removing all transitions, from states in E,

which take that state outside of E. For example, in Figure 3.1, we only remove the transition $0 \rightarrow^{\beta} 1$. Then, x is pre-stable if and only if all primary cycles in $R_E(A, x)$ go through E. In Figure 3.1, the self-loop at 1 is a primary cycle and does not go through E.

Proposition 3.2 Given a live A and $x \in X$, x is E-pre-stable iff all primary cycles in $R_E(A, x)$ include at least one state in E. In general, A is E-pre-stable iff all primary cycles in X include at least one state in E.

Proof: Straightforward by assuming the contrary in each direction. ❑

The class of sets, that are E-pre-stable, is closed under arbitrary unions and intersections. Now, we derive an algorithm that computes X_P, the maximal subset of X that is E-pre-stable. Our algorithm, which is the same as the one given for inevitable reachability in Sifakis, is based on starting from E and growing the currently known set of pre-stable states by including, at each step, those states x such that $f(x, d(x))$ is a subset of the current set.

In developing this algorithm, we first need the following lemma, which states that if some state x is pre-stable, then either x is in E or all the events defined from x take x to a pre-stable state:

Lemma 3.3 A state $x \in X$ is E-pre-stable iff $x \in E$ or $f(x, d(x))$ is E-pre-stable.

Proof: Straightforward. ❑

Also, note that given a set of pre-stable states Q, that include E, we can test pre-stability of other states by testing Q-pre-stability.

Lemma 3.4 Given Q_1, $Q_2 \subset X$ such that Q_1 is pre-stable with respect to E and $Q_1 \supset E$, Q_2 is E-pre-stable iff Q_2 is also Q_1-pre-stable.

Proof: ($\rightarrow$) Obvious since $Q_1 \supset E$.

($\leftarrow$) Suppose that some $x_2 \in Q_2$ goes to a cycle that avoids E. By Proposition 3.2, this cycle goes through some state $x_1 \in Q_1$. Then, x_1 cannot be E-pre-stable, and we have a contradiction. ❑

These two lemmas lead to the following algorithm:

Proposition 3.5 The following algorithm computes X_P, and it has complexity $O(n^2)$:

Algorithm Let $X_0 = E$ and iterate:

$$X_{k+1} = \{x | f(x, d(x)) \subset X_k\} \cup X_k$$

Terminate when $X_{k+1} = X_k$. Then $X_P = X_k$.

Proof: Clearly, X_0 is E-pre-stable. Suppose that X_k is E-pre-stable. By Lemma 3.3, X_{k+1} is X_k-pre-stable and by Lemma 3.4, it is also E-pre-stable. This algorithm terminates in at most n steps. Let us say that it terminates in r steps. Suppose that there exists some $x_1 \in X_P$ such that $x_1 \notin X_r$, then there exits $\sigma \in d(x_1)$ and $x_2 \notin X_r$ such that $x_2 \in f(x_1, \sigma)$. The same holds true for x_2 and some $x_3 \notin X_r$, etc. Thus, there exists a path which never reaches X_r. Since also $X_r \supset E$, x_1 is <u>not</u> E-pre-stable and we have a contradiction. Finally, to justify complexity, note that this algorithm terminates in at most n iterations. Since all states can be visited at most once at each iteration, the complexity of the algorithm is $O(n^2)$. ❑

In Figure 3.1, $X_1 = X_2 = X_P = \{0, 2, 3, 4\}$. Note that the number of steps in which this algorithm terminates, is a notion of radius for the pre-stable part of the system where E is taken as the center. That is, it is precisely the length of the maximum length trajectory between any pre-stable state and the state in E at which this trajectory enters E for the first time. In Figure 3.1, this radius is one. In

fact, if some x is included at step k of the algorithm, then the maximum number of transitions it takes to go from x to E is k.

3.2.2 Stability and f-Invariance

As motivated in the previous sections, we define stability as follows:

Definition 3.6 Given a live system A and some $E \subset X$, a state $x \in X$ is stable with respect to E (or, E-stable) if all infinite state trajectories starting from x visit E infinitely often. More precisely, x is stable if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}(A, x)$ so that $\mathbf{x}_2 = \mathbf{x}_1\mathbf{z}$, with $|\mathbf{z}| \geq n$, then there exists $y \in \mathbf{z}$ such that $y \in E$. ❑

This definition states that at any point in a trajectory from a stable state, we know that the trajectory will re-visit E within a finite number of transitions. In Figure 3.1, clearly, 1 is not stable. States 4 and 0 are not stable because there exist trajectories that start from these states and go to state 1, and subsequently, these trajectories may loop in state 1 forever.

An immediate consequence of this definition is the following, which states that the stability of a state is equivalent to the pre-stability of its reach.

Proposition 3.7 Given a live A and $x \in X$, x is E-stable iff $R(A, x)$ is E-pre-stable. ❑

A subset of X is stable if all its elements are stable, and a system A is termed stable if X is a stable set. We immediately have:

Proposition 3.8 A is a system E-stable iff it is also E-pre-stable. ❑

We also have the following counterpart of Proposition 3.2:

Proposition 3.9 Given a live A and $x \in X$, x is E-stable iff all primary cycles in $R(A, x)$ include at least one state in E. In general, A is E-stable iff all primary cycles in X include at least one state in E.

Proof: Straightforward. ❑

If we compare this to Proposition 3.2, note that the second statements are exactly the same. This is due to Proposition 3.8.

The class of sets that are E-stable is closed under arbitrary unions and intersections. Let X_S denote the maximal set E-stable (Note that X_S can be the empty set, for example, if we let $E = \{0\}$ in Figure 3.1). Then, X_S is the set of states in X_P from which we can only reach pre-stable states. Let us first formalize this notion of staying within a given set of states (corresponding to the notion of A-invariant subspaces of system theory):

Definition 3.10 A subset Q of X is f-invariant if $f(Q, d) \subset Q$ where

$$f(Q, d) = \bigcup_{x \in Q} f(x, d(x))$$ ❑

It immediately follows that any trajectory that starts in an f-invariant set stays in that set:

Proposition 3.11 A set Q is f-invariant iff $R(A, Q) \subset Q$.

Proof: Straightforward. ❑

The class of f-invariant sets is closed under arbitrary unions and intersections. We then have:

Proposition 3.12 The set X_S is the maximal f-invariant set in X_P.

Proof: ($\subset$) Clearly, $X_S \subset X_P$. Also, X_S is f-invariant since if a state $x \in X_S$ can reach a state that is not stable, then x cannot be stable.

($\supset$) Let X_f denote any f-invariant set in X_P. Any path from a state in X_f goes through E, and if gets out of E it stays in X_f and thus in X_P. Therefore, X_f is stable. ❑

Note that the maximal f-invariant set in Q is the set of states in Q that cannot reach any state in $\overline{Q}$, i.e., it is $\overline{R(A^{-1}, \overline{Q})}$. Thus, we can compute X_S as follows:

$$X_S = \overline{R(A^{-1}, \overline{X_P})} \tag{3.4}$$

3.3 Stabilizability

So far, we have dealt with notions that are close to those commonly seen in the automata theory literature. In this section, we introduce control and reconsider the notions formulated in the previous section. We define pre-stabilizability (respectively stabilizability) as finding a state feedback such that the closed loop system is pre-stable (stable). We present a polynomial algorithm for constructing a pre-stabilizing feedback. This algorithm is a natural extension of the algorithm in Proposition 3.5 and it generates a state feedback which is maximally restrictive, in the sense that it disables as many events as possible at each state, and path minimizing, in the sense that the maximum length path from pre-stable states to E is minimized. We also present an algorithm for constructing a minimally restrictive feedback. Finally, we introduce a notion of (f,u)-invariance, achieving f-invariance by choice of state feedback, and use this notion, together with the constraint that the closed loop system needs to be alive, to develop a polynomial algorithm for constructing a stabilizing feedback. In this section, since we examine the effect of control on the system, our models will have the form $A = (G, f, d, e)$ where $G = (X, \Sigma, U)$. The dynamics in this case are given by Equations (2.2) and (2.3).

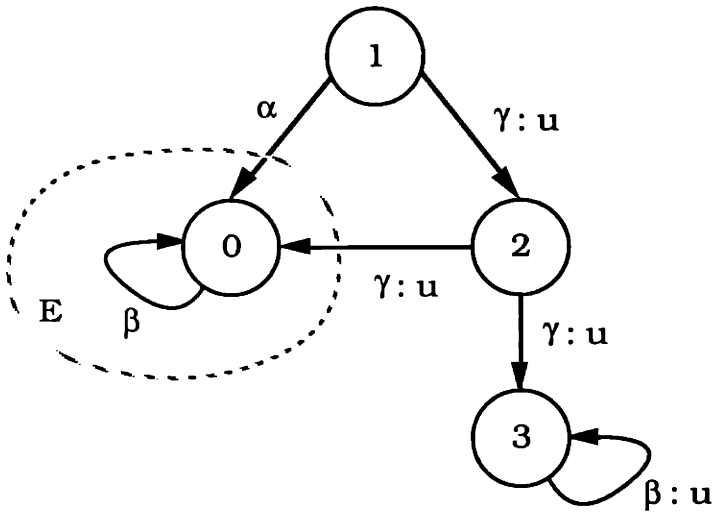

Figure 3.2: Example for the Notion of Pre-Stabilizability

3.3.1 Pre-Stabilizability

We define pre-stabilizability as follows:

Definition 3.13 Given a live system A and some $E \subset X$, $x \in X$ is pre-stabilizable with respect to E (or, E-pre-stabilizable) if there exits a state feedback K such that x is alive and E-pre-stable in A_K. A set of states, Q, is a pre-stabilizable set if there exists a feedback law $K(x)$ so that every $x \in Q$ is alive and pre-stable in A_K, and A is a pre-stabilizable system if X is a pre-stabilizable set. ❑

Often in our development here and in later chapters, when there is no chance of ambiguity, we will refer to stability and stabilizability without specific mention of E.

Figure 3.2 illustrates the importance of the liveness requirement in the above definition. Note that 1 is pre-stabilizable since disabling γ pre-stabilizes 1. On the other hand, disabling γ at 2 leaves no other defined event at 2. Thus, neither 2 nor 3 is pre-stabilizable.

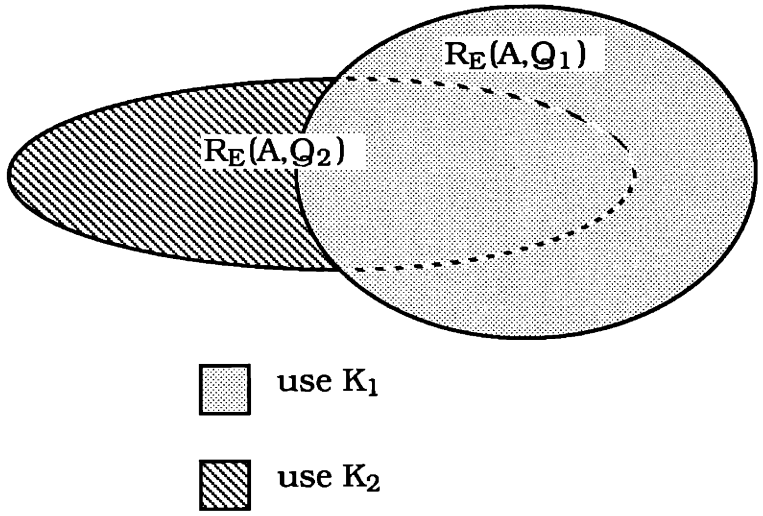

Figure 3.3: Feedback that Pre-Stabilizes the Union of Two Sets

Let Q_1 and Q_2 be two pre-stabilizable sets. Clearly, any feedback that pre-stabilizes either one of them also pre-stabilizes their intersection. Thus, pre-stabilizable sets are closed under intersections. The following result states that they are also closed under union:

Proposition 3.14 Given pre-stabilizable sets Q_1 and Q_2, $Q_1 \cup Q_2$ is also pre-stabilizable.

Proof: We show this by constructing a feedback that pre-stabilizes the union. First let K_i pre-stabilize Q_i. Then, pick, say, K_1 for the reach of Q_1, and K_2 for those states in the reach of Q_2, but not in the reach of Q_1 (see Figure 3.3). More precisely, we pick a feedback F as follows:

$$F(x) = \begin{cases} K_1(x) & \text{if } x \in R_E(A_{K_1}, Q_1) \\ K_2(x) & \text{if } x \in R_E(A_{K_2}, Q_2) \cap \overline{R_E(A_{K_1}, Q_1)} \\ \text{don't care} & \text{otherwise} \end{cases}$$

Recall that $R_E(A, Q)$ is the set of states that can be reached from Q by trajectories that do not exit E once they enter it. Clearly, Q_1 is pre-stable in the closed loop system A_F. By Lemma 3.4, Q_2 is also pre-stable, since the trajectories, from a state in Q_2, either go to E or go to a state that is also in the range of Q_1, in which case, they will eventually go to E. Thus, F pre-stabilizes $Q_1 \cup Q_2$. ❑

We immediately have the following corollary:

Corollary 3.15 A unique maximal set that is E-pre-stabilizable exists. Let $P(E)$ denote this set. ❑

Recall, from Lemma 3.3, that a necessary and sufficient condition for the pre-stability of a state x is the pre-stability of $f(x, d(x))$. A natural generalization of this condition is that the set of events defined at x can be restricted, say by $K \subset \Sigma$, so that there is at least one event defined from x, i.e., $d_K(x) = (d(x) \cap K) \cup e(x) \neq \emptyset$, and all those events take x to pre-stabilizable states, i.e., $f(x, d_K(x))$ is a pre-stabilizable set:

Lemma 3.16 A state $x \in X$ is E-pre-stabilizable iff $x \in E$ or there exists some $K \subset \Sigma$ such that $d_K(x) = (d(x) \cap K) \cup e(x) \neq \emptyset$ and $f(x, d_K(x))$ is E-pre-stabilizable.

Proof: ($\rightarrow$) Immediate from Lemma 3.3.

($\leftarrow$) Let K_1 be a feedback that pre-stabilizes $f(x, d_K(x))$. Let

$$K_2(x') = \begin{cases} K_1(x') & \text{if } x' \in R_E(A_{K_1}, f(x, d_K(x))) \\ K & \text{if } x' = x \\ \text{don't care} & \text{otherwise} \end{cases}$$

Then, also by Lemma 3.4, K_2 pre-stabilizes x. Note that we do not care about the feedback for states other than the ones we have included in the above equation for K_2, since x cannot reach those states under K_2 ❑

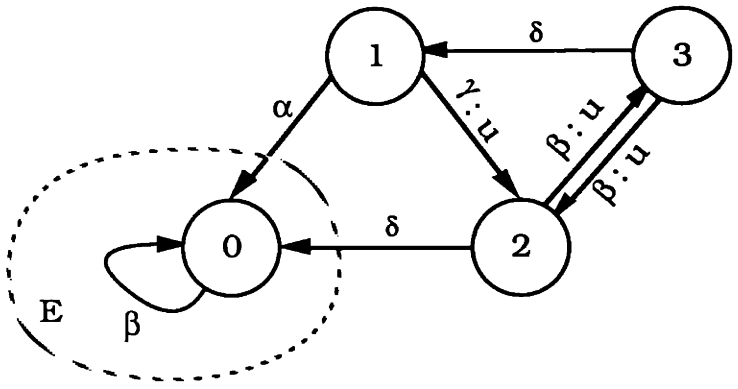

Figure 3.4: Example for the Pre-Stabilizability Algorithm

Now, we can construct a natural counterpart of the algorithm in Proposition 3.5 using the above lemma. As in Proposition 3.5, we start with E and then add in the states that satisfy Lemma 3.16. In particular, at each step, we include the states x for which $e(x) \neq \emptyset$ and $f(x, e(x))$ is a subset of the current set of states which are known to be pre-stabilizable, or (if $e(x) = \emptyset$), we can find an event $\sigma \in d(x)$ such that $f(x, \sigma)$ is a subset of the current set. For example, in Figure 3.4, we start with the state 0. At the first step, we include 1 and 2, and at the second step we include 3.

Proposition 3.17 The following algorithm computes $P(E)$ and a feedback that pre-stabilizes it. It has complexity $O(n^2)$:

Algorithm Let $X_0 = E$ and iterate:

$$P_{k+1} = \left\{ x \mid \begin{array}{l} (e(x) \neq \emptyset \text{ and } f(x, e(x)) \subset X_k) \text{ or} \\ \quad (e(x) = \emptyset \text{ and } \exists \sigma \in d(x) \text{ such that } f(x, \sigma) \subset X_k) \end{array} \right\}$$

$$K(x) = \left\{ \begin{array}{ll} \emptyset & \text{if } e(x) \neq \emptyset \\ \text{some } \sigma \text{ such that } f(x, \sigma) \subset X_k & \text{otherwise} \end{array} \right. , \qquad \text{for } x \in P_{k+1}$$

$$X_{k+1} = X_k \cup P_{k+1}$$

Terminate when $X_{k+1} = X_k$. Then $P(E) = X_k$.

Proof: Straightforward by following the proof of Proposition 3.5 and using Lemma 3.16. ❑

In Figure 3.4, γ is disabled at 1 and β is disabled at 2 in the first step, and β is disabled at 3 in the second step.

The above algorithm leads to a feedback that disables as many events as possible. We formalize this as follows:

Definition 3.18 A pre-stabilizing state feedback K is maximally restrictive, if for any pre-stabilizing feedback K' such that $d_{K'}(x) \subset d_K(x)$ for all $x \in P(E) \cap \overline{E}$, $K' = K$. ❑

We immediately have the following result:

Proposition 3.19 A pre-stabilizing feedback K is maximally restrictive iff

$$K(x) = \begin{cases} \sigma \in d(x) & \text{if } e(x) = \emptyset \\ e(x) & \text{otherwise} \end{cases},$$

for all $x \in P(E) \cap \overline{E}$.

Proof: Straightforward. ❑

Thus, the algorithm in Proposition 3.17 leads to a maximally restrictive feedback.

The feedback presented in Proposition 3.17, also minimizes the maximum number of transitions it takes to go from a state to E. Clearly, it also minimizes the radius. In Figure 3.4, it takes a single transition to go from 1 or 2 to 0, and two transitions to go from 3 to 0. To formalize this, let us assume, without loss of generality, that A is stabilizable, and let $r(A, x)$ denote the length of the longest path from x to a state in E, where $r(A, x) = 0$ for all $x \in E$:

Definition 3.20 Given a stabilizable system A, a pre-stabilizing state feedback K is path minimizing if for any pre-stabilizing feedback K' such that $r(A_{K'}, x) \leq r(A_K, x)$ for all x, $r(A_{K'}, x) = r(A_K, x)$ for all x. ❑

As the following result shows, for any two path-minimizing feedbacks, the longest path lengths are equal at each state:

Proposition 3.21 For any two path minimizing feedbacks K_1, and K_2, $r(A_{K_1}, x) = r(A_{K_2}, x)$ for all x.

Proof: We prove this by assuming the contrary and constructing a feedback, F which leads to shorter path lengths than those of K_1 and K_2.

If we assume the contrary, then there exists $x, y \in X$ such that $r(A_{K_1}, x) < r(A_{K_2}, x)$ and $r(A_{K_1}, y) > r(A_{K_2}, y)$. To construct F, we want to use K_1 for the reach of x and K_2 for the reach of y. However, we need to make sure that we do not introduce any loops by doing this. Thus, we want to show that if $x \in R_E(A_{K_2}, y)$ then $y \notin R_E(A_{K_1}, x)$, and vice versa.

Assume the contrary to this, i.e., assume that $x \in R_E(A_{K_2}, y)$, and $y \in R_E(A_{K_1}, x)$. If $x \in R_E(A_{K_2}, y)$, then $r(A_{K_2}, y) > r(A_{K_2}, x)$. On the other hand, if $y \in R_E(A_{K_1}, x)$, then $r(A_{K_1}, x) > r(A_{K_1}, y)$. Since also $r(A_{K_1}, x) < r(A_{K_2}, x)$ and $r(A_{K_1}, y) > r(A_{K_2}, y)$, $r(A_{K_2}, x) > r(A_{K_2}, y)$. Thus, we have a contradiction.

Therefore, the following choice of F is pre-stabilizing, if we assume, without loss of generality, that $x \in R_E(A_{K_2}, y)$:

$$F(x') = \begin{cases} K_1(x') & \text{if } x' \in R_E(A_{K_x}, x) \\ K_2(x') & \text{otherwise} \end{cases}$$

F also leads to shorter path lengths than those of K_1 and K_2, and thus, neither K_1 nor K_2 is path-minimizing. Therefore, $r(A_{K_1}, x) = r(A_{K_2}, x)$ for all x. ❑

Let $\overline{r}(x)$ denote this minimal path length for x. Finally, we have the following:

Proposition 3.22 The feedback presented in Proposition 3.17, K, is path-minimizing. Furthermore, $x \in P_k$ at step k of the algorithm, if and only if $k = \overline{r}(x)$, and thus, the algorithm also constructs $\overline{r}(x)$.

Proof: Suppose that K is path minimizing for X_k at step k of the algorithm in Proposition 3.17. Then, for some $x \in P_{k+1}$, $f(x, d_K(x)) \cap P_k \neq \emptyset$ since otherwise $x \in P_i$ for some $i < k+1$. Therefore, K is path minimizing for X_{k+1} and by induction, K is path-minimizing for $P(E)$. The proof of the second statement is straightforward. ❑

If all the trajectories in the desired behavior consist of states in E, then a maximally restrictive feedback is desirable for stabilization since it does not restrict the desired behavior, and in addition, it ensures returning to E in a minimum number of transitions. However, if the desired behavior involves states outside of E (for example, if E is simply a set of desired initial states), then one would prefer a less restrictive feedback so that all stable trajectories of the desired behavior are enabled. In what follows, we present an algorithm to construct a feedback that disables as few events as possible:

Definition 3.23 A pre-stabilizing feedback K is minimally restrictive, if for any pre-stabilizing feedback K' such that $d_{K'}(x) \supset d_K(x)$ for all $x \in P(E) \cap \overline{E}$, $K' = K$. ❑

Our algorithm is based on the following lemma, which states that a feedback, K, is minimally restrictive if and only if enabling any event at any state, which is otherwise disabled, makes that state unstable, i.e., creates a cycle that does not go through E. That is, K is minimally restrictive if for any state x, enabling any σ that had been disabled by K can move the system to a new state from which it

can return to x without going through E. For example, in Figure 3.4, consider the feedback which disables γ at 1 and β at 3. Enabling either of these events makes the corresponding state unstable.

Lemma 3.24 A pre-stabilizing state feedback K is minimally restrictive iff for all $x \in P(E)$ and $\sigma \in d(x) \cap \overline{K(x)} \cap \overline{e(x)}$, $x \in R_E(A_K, f(x,\sigma))$.

Proof: ($\rightarrow$) Straightforward by assuming the contrary.

($\leftarrow$) Since we cannot enable any event without making some state unstable, K is certainly a minimally restrictive feedback. ❑

In order to compute a minimally restrictive feedback, we start with a maximally restrictive feedback, and add events, that are otherwise disabled, until the condition of the the above lemma is satisfied. Our algorithm visits all states in $P(E) \cap \overline{E}$ and for each state x, it includes all events $\sigma \in d(x) \cap \overline{K(x)} \cap \overline{e(x)}$, such that $x \notin R_E(A_K, f(x,\sigma))$, in $K(x)$. Since K is possibly modified after visiting a state, when the next state is visited, the new feedback should be used in computing $R_E(A_K, f(x,\sigma))$. For example, in Figure 3.4, if we start with state 3, we get the minimally restrictive feedback which disables γ at 1 and β at 3. Depending on the order the states are visited, different minimally restrictive feedbacks may be generated. In Figure 3.4, if we start with state 1 or 2, we get the minimally restrictive feedback which disables β at 2.

On the other hand, we do not need to compute $R_E(A_K, f(x,\sigma))$ for each σ and x. Instead, we can compute, for each x, the set of states that can reach x and check to see if any element of $f(x,\sigma)$ is in this set:

Lemma 3.25 Given x and $\sigma \in d(x) \cap \overline{K(x)} \cap \overline{e(x)}$, $x \in R_E(A_K, f(x,\sigma))$ iff

$$f(x,\sigma) \cap R_E(A_K^{-1}, x) \neq \emptyset$$

Proof: Straightforward. ❑

We then have the following algorithm:

Proposition 3.26 The following algorithm computes a minimally restrictive feedback. It has complexity $O(n^2)$:

Algorithm For all $x \in P(E)$ do:

$$\begin{aligned} S &= K(x) \cup \{\sigma \in d(x) \cap \overline{K(x)} \cap \overline{e(x)} | f(x,\sigma) \cap R_E(A_K^{-1}, x) = \emptyset\} \\ K(x) &= S \end{aligned}$$

Proof: The proof follows from Lemma 3.24. The complexity is $O(n^2)$ since all states are visited and the reach operation has complexity $O(n)$. ❑

3.3.2 Stabilizability and (f,u)-Invariance

Stabilizability, like pre-stabilizability, is defined as a natural extension of stability. A state x is stabilizable if we can find a state feedback such that x is stable in the closed loop system:

Definition 3.27 Given a live system A and some $E \subset X$, $x \in X$ is stabilizable with respect to E (or, E-stabilizable) if there exits a state feedback K such that x is alive and E-stable in A_K. A set of states, Q, is a stabilizable set if there exists a feedback law $K(x)$ so that every $x \in Q$ is alive and stable in A_K, and A is a stabilizable system if X is a pre-stabilizable set. ❑

Let Q_1 and Q_2 be two stabilizable sets. Clearly, any feedback that stabilizes either one of them also stabilizes their intersection. Thus, stabilizable sets are closed under intersections. The following result states that they are also closed under union:

Proposition 3.28 Given stabilizable sets Q_1 and Q_2, $Q_1 \cup Q_2$ is also stabilizable.

Proof: We show this by constructing a feedback that stabilizes the union. First let K_i stabilize Q_i. Then, pick, say, K_1 for the reach of Q_1, and K_2 for those states in the reach of Q_2, but not in the reach of Q_1. More precisely, we pick a feedback F as follows:

$$F(x) = \begin{cases} K_1(x) & \text{if } x \in R(A_{K_1}, Q_1) \\ K_2(x) & \text{if } x \in R(A_{K_2}, Q_2) \cap \overline{R(A_{K_1}, Q_1)} \\ \text{don't care} & \text{otherwise} \end{cases}$$

F clearly stabilizes $Q_1 \cup Q_2$. ❑

We immediately have the following corollary:

Corollary 3.29 A unique maximal set that is E-stabilizable exists. Let $S(E)$ denote this set. ❑

To achieve stabilizability, we try to make $P(E)$ f-invariant by choice of feedback. Thus, we first need to define the following counterpart of the notion of (A, B)-invariance:

Definition 3.30 A subset Q of X is (f,u)-invariant if there exists a state feedback K such that Q is f-invariant in A_K. ❑

Let $A_\emptyset$ denote A with all controllable events disabled, i.e., $A_\emptyset = (X, f, e, e)$. Note that if some Q is f-invariant in $A_\emptyset$ then it is also (f,u)-invariant in A. The following result formalizes this and it also establishes a connection with the well-known result in [48] that a subspace $\mathcal{V}$ is (A, B)-invariant iff $A\mathcal{V} \subset \mathcal{V} + \mathcal{B}$, where $\mathcal{B}$ is the range of B (compare to item 2 below):

Proposition 3.31 The following statements are equivalent:

1. Q is (f,u)-invariant in A.
2. $\forall x \in Q$, $f(x, d(x)) \subset Q \cup \overline{f(x, e(x))}$.
3. Q is f-invariant in $A_\emptyset$.

Proof: $(1 \rightarrow 2)$ Suppose that there exists $x \in Q$, $\sigma \in d(x)$, $y \in f(x, \sigma)$ such that $y \notin Q$ and $y \notin \overline{f(x, e(x))}$. But, then $y \in f(x, e(x))$ and we have a contradiction since the transition to y is undesired and cannot be disabled.
$(2 \rightarrow 3)$ Assume the contrary. By Proposition 3.11, there exists $x \in Q$, $\sigma \in e(x)$ such that $f(x, \sigma) \not\subset Q$. By (2), $f(x, \sigma) \subset \overline{f(x, e(x))}$ and we have a contradiction.
$(3 \rightarrow 1)$ Simply use the feedback $K(x) = e(x)$. ❑

The class of (f,u)-invariant sets is closed under arbitrary unions and intersections. Thus, a unique maximal (f,u)-invariant subset of Q exits, and let $T(Q)$ denote this set. Clearly, $T(Q) = \overline{R(A_\emptyset^{-1}, \overline{Q})}$. This notion of (f,u)-invariance, however, is not sufficient for stabilizability since we also need to keep $S(E)$ alive. So, we define the following notion which requires that we can find a state feedback such that Q is both alive and f-invariant in the closed loop system:

Definition 3.32 A subset Q of X is a sustainably (f,u)-invariant set if there exists a state feedback K such that Q is alive and f-invariant in A_K. ❑

The class of sustainably (f,u)-invariant sets is closed under arbitrary unions but not intersections. In Figure 3.5, Q_1 (respectively Q_2) can be made (f,u)-invariant and alive by disabling δ (respectively α) at state 1. On the other hand, $Q_1 \cup Q_2$ is clearly sustainably (f,u)-invariant, but $Q_1 \cap Q_2$ is not, since if both α and δ are disabled, the state 1 is no longer alive. Let $I(Q)$ denote the maximal sustainable (f,u)-invariant subset of Q.

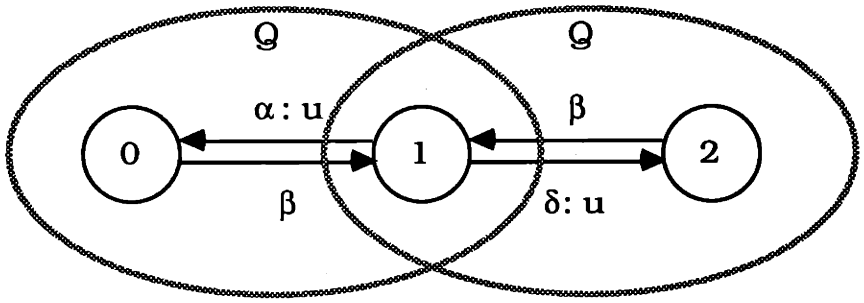

Figure 3.5: Example for the Sustainable (f,u)-Invariance of Unions and Intersections

The characterization of sustainable (f,u)-invariance requires a slightly more careful look at what it means if an (f,u)-invariant set is not sustainable. Specifically, for any Q and all states x in $T(Q)$, we know that all events that may take x outside of $T(Q)$ are controllable, and can therefore be disabled to achieve the desired invariance. However, x may have no other events defined, and thus, making $T(Q)$ f-invariant will disable all events from x. Then, $T(Q)$ is not sustainable. Our algorithm for computing $I(Q)$ is based on first computing $T(Q)$ and then throwing away all states that are no longer alive. We then apply the same procedure to this new set. This iteration continues until no states are discarded on a step. The following result states that we then have a sustainable (f,u)-invariant set:

Proposition 3.33 Given $Q \subset X$, let

$$Q' = T(\{x \in Q | \text{ there exists some } \sigma \in d(x) \text{ such that } f(x,\sigma) \subset Q\})$$

Then, Q is sustainably (f,u)-invariant iff $Q' = Q$.

Proof: ($\rightarrow$) Obvious

($\leftarrow$) If $Q' = Q$ then $Q = T(Q)$ and for all $x \in Q$ there exists some $\sigma \in d(x)$ such that

$f(x,\sigma) \subset Q$. Therefore, Q is alive and f-invariant in A_K with $K(x) = \{\sigma | f(x,\sigma) \subset Q\}$. ❑

This result implies that in order to find $I(Q)$, we can apply the operation T iteratively by throwing away the states that are no longer alive after the last step:

Proposition 3.34 The following algorithm computes $I(Q)$ and it has complexity $O(n^2)$:

Algorithm Let $X_0 = Q$. Iterate:

$$X_{k+1} = T(\{x \in X_k | \text{ there exists } \sigma \in d(x) \text{ such that } f(x,\sigma) \subset X_k\})$$

Terminate when $X_{k+1} = X_k$. Then $I(Q) = X_k$.

Proof: Clearly, this algorithm terminates in a finite number of steps, say r steps. By Lemma 3.33, $X_r \subset I(Q)$. On the other hand, $I(Q) \subset X_0$. Suppose that $I(Q) \subset X_k$ for some k. Then, $X_{k+1} \supset T(I(Q)) = I(Q)$. Thus, $I(Q) \subset X_k$ for all k, implying that $I(Q) = X_r$. To justify the computational complexity, note that we visit each state at most once at each iteration, and there can be at most n iterations. ❑

Now, we proceed with deriving an algorithm for the maximal stabilizable set, $S(E)$. We begin by computing $P(E)$, the maximal pre-stabilizable set with respect to E. If $P(E)$ were sustainably (f,u)-invariant, we are done, with $S(E) = P(E)$. More generally, however, there may be some states in $P(E)$ for which it is impossible to find a feedback which keeps trajectories within $P(E)$. Furthermore, since all elements of $P(E)$ are pre-stabilizable, some of these troublesome states must be in E. Thus, what we must do is to compute $I(P(E))$ and then discard elements of E not in $I(P(E))$, reducing E to a new set E'. However, there may now be states which were E-pre-stabilizable but not with respect to E', and we therefore must repeat the process. For example, in Figure 3.6, $E = \{0,3\}$, $P(E) = \{0,1,2,3\}$ and $I(P(E)) = \{1,2,3\}$,

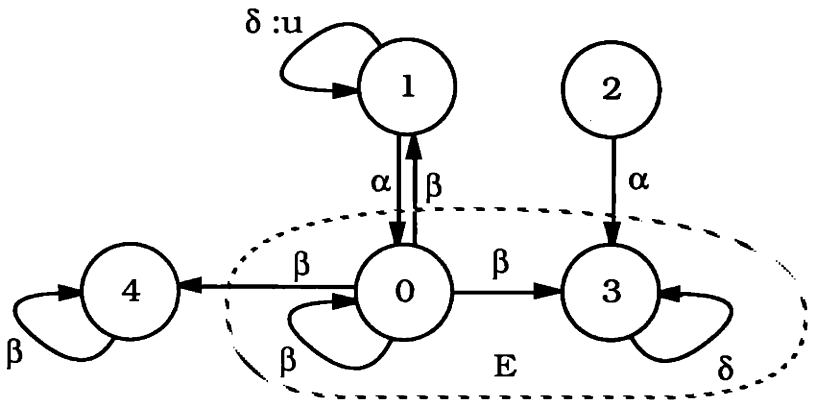

Figure 3.6: Example for the Stabilizability Algorithm

so that $E' = \{3\}$. Now, $\{1\}$ is not E'-pre-stabilizable and must be discarded. The iteration in this case would produce $P(E') = \{2,3\} = I(P(E'))$. It is not difficult to check that $\{2,3\} = S(E)$ as well, and indeed the following result states this more generally and provides the basis for our algorithm:

Lemma 3.35 Given $E, Q \subset X$, let

$$Q' = I(P(E \cap Q))$$

If $Q' = Q$ then Q is E-stabilizable.

Proof: We first show that $Q = P(E \cap Q)$. If $Q' = Q$, then, clearly, $Q \subset P(E \cap Q)$. In order to show that $Q \supset P(E \cap Q)$, suppose that there exists some $x \in P(E \cap Q)$ such that $x \notin Q$ (see Figure 3.7). Then, there exists some feedback such that all paths from x go to $E \cap Q$ with trajectories that only lie in $P(E \cap Q)$. Since by assumption $Q = I(P(E \cap Q))$ but $x \notin Q$, x can reach some $y \in E \cap Q$ and $\sigma \in e(y)$ such that $f(y, \sigma) \notin P(E \cap Q)$. But then, Q cannot be sustainably (f,u)-invariant, which is a contradiction. Therefore, $Q = P(E \cap Q)$, and this immediately implies that $Q = I(Q)$ as well.

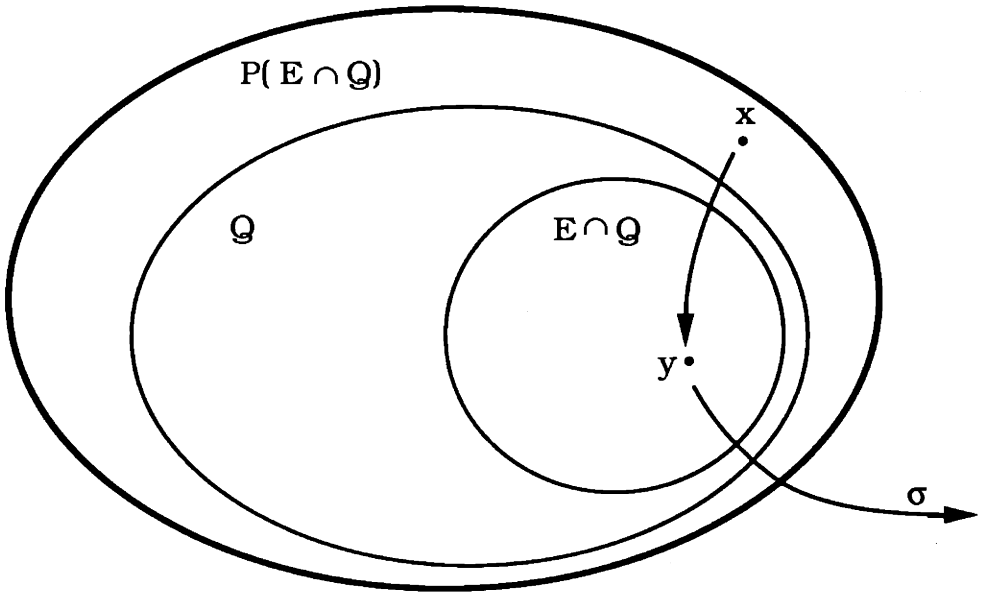

Figure 3.7: Illustration for the Proof of Lemma 3.35

In order to show that Q is stabilizable, let K_P denote a feedback that pre-stabilizes $P(E \cap Q)$ and let

$$K(x) = \begin{cases} K_P(x) & \text{if } x \in \overline{E} \cap Q \\ \{\sigma | f(x,\sigma) \subset Q\} & \text{if } x \in E \cap Q \end{cases}$$

Q is clearly alive in A_K. Q is also pre-stable in A_K since K_P insures that all paths from states in $Q \cap \overline{E}$ go to $Q \cap E$. Q is then stable in A_K. Therefore, Q is stabilizable, and K is a stabilizing feedback. ❑

This result leads to the following algorithm:

Proposition 3.36 The following algorithm computes $S(E)$ and a feedback that stabilizes it. It has complexity $O(n^3)$.

Algorithm Let $X_0 = X$ and iterate:

$$X_{k+1} = I(P(E \cap X_k))$$

Terminate when $X_{k+1} = X_k$. Then $S(E) = X_k$.

Proof: To show that $X_{k+1} \subset X_k$ for all k, first note that $X_1 \subset X_0$. Assume that $X_{k+1} \subset X_k$ for some k. Let P_k denote $P(E \cap X_k)$. Note that $P_{k+1} \subset P_k$ since $E \cap X_{k+1} \subset E \cap X_k$. Then, $I(P_{k+1}) \subset I(P_k)$ and thus $X_{k+2} \subset X_{k+1}$. Therefore, this algorithm terminates in a finite number of steps, say r steps. By Lemma 3.35, $X_r \subset S(E)$. On the other hand, $S(E) \subset X_0$. Assume that $S(E) \subset X_k$ for some k. Then, clearly, $S(E) \subset P(E \cap X_k)$. Since also $I(S(E)) = S(E)$, $S(E) \subset X_{k+1}$, so that $S(E) = X_r$. The feedback which achieves pre-stability at step r, say F, also achieves stability for $S(E)$, since states in $\overline{S(E)}$ are not $E \cap S(E)$-pre-stable and thus F cannot be enabling any event that takes a state in $S(E)$ outside of $S(E)$. Also, for states in $E \cap S(E)$, all events that take those states outside of $S(E)$ should be disabled. In short, the stabilizing feedback K is

$$K(x) = \begin{cases} F(x) & \text{if } x \in S(E) \cap \overline{E} \\ \{\sigma \in d(x) | f(x,\sigma) \subset S(E)\} & \text{if } x \in S(E) \cap E \\ \text{don't care} & \text{otherwise} \end{cases}$$

To justify computational complexity, recall that the computation of $P(E)$ is $O(n^2)$, and note that the above algorithm terminates in at most n steps. ❑

Note that a stabilizing feedback has two components: one component for pre-stability and another for invariance. The first component is concerned with states in $S(E) \cap \overline{E}$, whereas the second component is concerned with states in $S(E) \cap E$. Note also that given a sustainably (f,u)-invariant set V, there exists a unique minimally restrictive feedback K_V that achieves f-invariance: at each $x \in V$, we let

$$K_V(x) = \{\sigma \in d(x) | f(x,\sigma) \subset V\} \tag{3.5}$$

Thus, in order to construct a minimally restrictive stabilizing feedback, we first choose a minimally restrictive pre-stabilizing feedback, and then for all $x \in S(E) \cap E$ we let

$K(x) = \{\sigma \in d(x) | f(x,\sigma) \subset S(E)\}$. Also, in order to construct a maximally restrictive stabilizing feedback K, we first choose a maximally restrictive pre-stabilizing feedback and then for all $x \in S(E) \cap E$, we let

$$K(x) = \begin{cases} e(x) & \text{if } e(x) \neq \emptyset \\ \sigma \in d(x) | f(x,\sigma) \subset S(E) & \text{otherwise} \end{cases} \tag{3.6}$$

3.4 Stabilizability of Composite Systems

The algorithms we have described in the previous sections have complexity that is polynomial in $n = |X|$. One difficulty with this is that in many cases n can be quite large. In this section we describe methods aimed at one of the most important cases in which large cardinality state sets are encountered. In particular, many DEDS consist of compositions of two or more subsystems so that the overall cartesian product state set is quite large. However, as we will see, depending on how these subsystems are connected, it may be possible to construct algorithms for testing the stability and stabilizability of the composed system that achieve significant computational savings, especially if the interconnections are sparse. In particular, what we will see here is that if interactions occur only when the subsystems are in a small subset of their respective state spaces, we can reduce computational complexity significantly by designing efficient algorithms for testing stability and stabilizability.

In this section we focus on a DEDS $A = A_1 \parallel A_2$ consisting of the composition of two subsystems $A_1 = (G_1, f_1, d_1, e_1)$ and $A_2 = (G_2, f_2, d_2, e_2)$ with shared events $S = \Sigma_1 \cap \Sigma_2$, and with $n_i = X_i$. Note that even if both A_1 and A_2 are alive, A is not necessarily alive. In particular, A is alive iff for all $x = (x_1, x_2) \in X$, such that $d_1(x_1) \subset S$ and $d_2(x_2) \subset S$, $d_1(x_1) \cap d_2(x_2)$ is <u>not</u> empty. Therefore, it is very easy to check if A is alive and we will assume that this is the case.

In the next subsection we address the problem of computing reachable sets, while in subsequent subsections we look at stability and stabilizability. As we will see, the algorithms we describe provide efficiencies if the portion of the composite state space in which interactions take place is small.

3.4.1 Range of Composite States

We begin with some notation:

- We define functions $Pr_i : X \rightarrow X_i$ so that given $x = (x_1, x_2) \in X$, $Pr_i(x) = x_i$. We also extend Pr_i to act on subsets of X: Given $Q \subset X$, we let $Pr_i(Q) = \bigcup_{x \in Q} Pr_i(x)$.

- Let $Y_i^c \subset X_i$ denote the set of states $x \in X_i$ such that $d_i(x) \cap S \neq \emptyset$. We also let $Y^c = Y_1^c \times Y_2^c$, $m_i = |Y_i^c|$, and $l_i = |Pr_i(f(Y^c, S))|$. Note that Y^c is the set of states in the composite in which A_1 and A_2 may interact through the occurrences of shared events. Our reachability algorithm exploits the fact that the interactions between A_1 and A_2 only take place in the set Y^c. Essentially, it is only in Y^c that we need to consider the joint evolution of A_1 and A_2. In addition, we will see that the set of states to which shared events take states in Y^c, i.e., the set $f(Y^c, S)$, also plays an important role in our algorithm. Consequently, we will see that the complexity of our algorithm in this section decreases as m_i and l_i decrease.

- We let $A_i|\overline{S}$ denote the automaton A with all events in S removed.

- Given $x \in X$, we let $R_x = R(A_1|\overline{S}, Pr_1(x)) \times R(A_2|\overline{S}, Pr_2(x))$. Also, given $Q \subset X$, we let $R_Q = \bigcup_{x \in Q} R_x$. Note that R_x can be computed by computing

the reach independently in each subsystem, as no shared events are involved. This will also contribute significantly to the efficiency of our algorithm.

What we would now like to do is to compute $R(A, X_0)$ for any specified $X_0 \subset X$. Because of the structure of the system, we need only consider the joint evolution of A_1 and A_2 when we are in Y^c. Thus, we will focus first on computing $R(A, X_0) \cap Y^c$. Once we have this, we will see that we can compute all of $R(A, X_0)$ by considering only the decoupled evolution of A_1 and A_2 when shared events are eliminated.

The following fixed point result is the basis for our computation on $R(A, X_0) \cap Y^c$:

Lemma 3.37 Given $X_0 \subset X$, and Q such that

$$R_{X_0} \cap Y^c \subset Q \subset R(A, X_0) \cap Y^c$$

let

$$\begin{aligned} P &= f(Q, S) \\ Q' &= Q \cup (Y^c \cap R_P) \end{aligned}$$

Then, $Q' \subset R(A, X_0) \cap Y^c$. Moreover, $Q = R(A, X_0) \cap Y^c$ iff $Q = Q'$.

Proof: (of first part) Clearly, $Q' \subset Y^c$. On the other hand,

$$Q \subset R(A, X_0) \rightarrow R(A, Q) \subset R(A, X_0) \rightarrow P \subset R(A, X_0) \rightarrow R(A, P) \subset R(A, X_0)$$

Since $R_p \subset R(A, P)$ for all $p \in P$, $Q' \subset R(A, X_0)$.

Proof: (of second part) ($\rightarrow$) Clearly, $Q' \supset Q$. By the proof of the first part, $Q' \subset Q$. ($\leftarrow$) By assumption, $Q \subset R(A, X_0) \cap Y^c$. Therefore we only need to show the opposite inclusion under the assumption that $Q = Q'$. To do this, let us assume the contrary. Then, let some $x \in R(A, X_0) \cap Y^c$ so that $x \notin Q$. Since $Q \supset R_{X_0} \cap Y^c$, any trajectory from X_0 to x must involve at least one shared event. Thus, we can conclude that

there exists $x_1 \in Q$ such that $x \in R(A, x_1)$. Let **x** denote any trajectory from x_1 to x. Next, remove all those states in **x** that are <u>not</u> in Y^c, and let **x'** denote the remaining string of states. Note that x_1 is also the first state of **x'** and x is the last state. Let x_j be the first state in **x'** such that $x_j \in Q$ but the next state in **x'**, $x_{j+1} \notin Q$. Such a state x_j clearly exists since $x \notin Q$. However, for all x_j in **x'**, x_{j+1} is such that $x_{j+1} \in Y^c \cap R_{f(x_j,S)}$, so that $x_{j+1} \in Q'$. However, since $Q = Q'$, we have a contradiction. Therefore, $Q \supset R(A, X_0) \cap Y^c$. ❑

We can now present our algorithm for the efficient computation of the range of composite states:

Proposition 3.38 The following algorithm computes $R(A, X_0) \cap Y^c$ in at most $m_1 m_2$ steps, and it has complexity

$$O(m_1 m_2 + (p_1 + l_1)n_1 + (p_2 + l_2)n_2)$$

where $p_i = |Pr_i(X_0)|$.

Algorithm Let $Q_0 = R_{X_0} \cap Y^c$. Iterate:

$$\begin{aligned} P_i &= f(Q_i, S) \\ Q_{i+1} &= Q_i \cup (Y^c \cap R_{P_i}) \end{aligned}$$

Terminate when $Q_{i+1} = Q_i$. Then $R(A, X_0) \cap Y^c = Q_i$.

Proof: By the first part of Lemma 3.37, this algorithm terminates in a finite number of steps, say r steps. By the second part of Lemma 3.37, $Q_r = R(A, X_0) \cap Y^c$. It is clear that r can be at most $m_1 m_2$. Finally, the complexity is due to the maximum value of r, and the computation of R_{X_0} and R_{P_i}. ❑

Finally, we have the following result for computing <u>all</u> the states in the reach of X_0:

Proposition 3.39 If $Q = R(A, X_0) \cap Y^c$ then $R(A, X_0) = Q \cup R_{X_0} \cup R_{f(Q,S)}$.

Proof: ($\supset$) Straightforward.

($\subset$) Take any $x \in R(A, X_0)$. If $x \notin R_{X_0}$, then it must be reached only by strings containing elements of S. Let $s = s_1 s_2$ be a string such that $x \in f(X_0, s)$ and such that only the first event in s_2 is in S. Take any $x_1 \in f(X_0, s_1)$ such that $x \in f(x_1, s_2)$. Then obviously $x_1 \in R(A, X_0) \cap Y^c = Q$ and therefore, by the structure of s_2, $x \in R_{f(Q,S)}$. ❑

Since the elements needed to compute $R(A, X_0)$ are already computed by the algorithm in Proposition 3.38, the complexity of computing $R(A, X_0)$ is also

$$O(m_1 m_2 + (p_1 + l_1)n_1 + (p_2 + l_2)n_2) \tag{3.7}$$

Note that for small m_i and l_i we achieve substantial savings in the complexity as compared to applying the reachability algorithm in Chapter 2 directly to the composite. To illustrate this, suppose that $n_1 = n_2 = 100$, and suppose that we wish to compute the reach of a single state. Then the complexity of using the algorithm in Proposition 2.2 is on the order of 10,000. Now, suppose that $m_i = l_i = 10$, then the complexity of using the algorithm in Proposition 3.38 is on the order of 2,300 and therefore we achieve more than 75% savings in computational complexity. Note that since the complexity of our original algorithm in Chapter 2 is $O(n)$, it is not surprising that the savings realized by the above algorithm are relatively modest as compared to the savings realized by the algorithms in subsequent subsections that address the more complex stability problems of earlier sections of this chapter.

3.4.2 Stability of a Composite of Two Systems

Let us now turn our attention to computationally efficient algorithms for testing the stability of a composite of two systems. We first define the following notion of strong coupling which is identical to the notion of fairness in computer science literature (see, for example [11]). As stated mathematically below, strong coupling, requires that neither of the systems can make transitions indefinitely while the other does not make any transitions at all:

Definition 3.40 A_1 and A_2 are strongly coupled if for all $x \in X$ and $s \in L(A,x)$, such that $|s| > \max(n_1, n_2)$, $s \downarrow \Sigma_1 \neq \epsilon$, for $i = 1, 2$. ❑

Note that if A_1 and A_2 are not strongly coupled, then, assuming without loss of generality that A_1 can make transitions indefinitely while A_2 does not make any transitions at all, A is E-pre-stable iff $E = Q_1 \times X_2$ for some $Q_1 \subset X_1$ and A_1 is Q_1-pre-stable. Therefore, without the strong coupling assumption, testing stability is trivial; we just test the individual subsystems independently. Also, since fairness is generally a desirable property, we assume that A_1 and A_2 are strongly coupled in the rest of this chapter. The following result presents a straightforward test for strong coupling:

Proposition 3.41 Given alive A_1 and A_2, they are strongly coupled iff $A_i|\overline{S}$ is pre-stable with respect to $\{x \in X_i | d_i(x) \subset S\}$, for $i = 1, 2$.

Proof: Straightforward by assuming the contrary in each direction. ❑

Let us now turn our attention to constructing a computationally efficient pre-stability test. Let us first define some notation: Given $E \subset X$, we let

$$\widetilde{X} = \widetilde{X_1} \times \widetilde{X_2} \triangleq (Y_1^c \cup Pr_1(E)) \times (Y_2^c \cup Pr_2(E)) \tag{3.8}$$

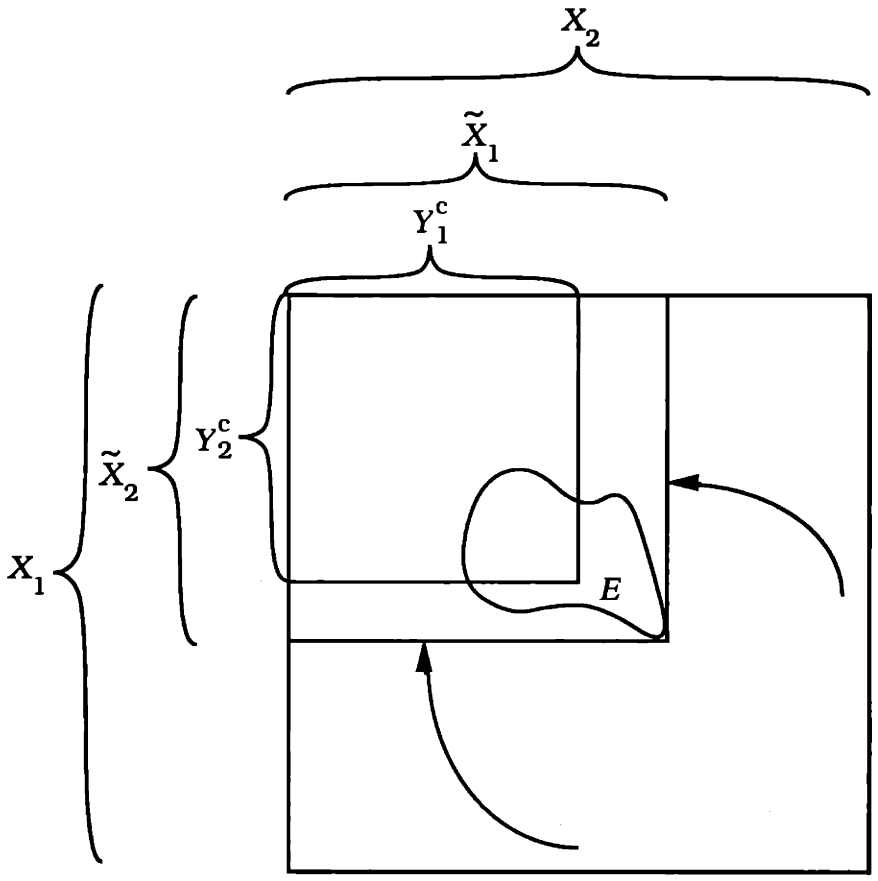

Figure 3.8: Composite State Space: Horizontal line (respectively, vertical line) corresponds to the state space of A_1 (respectively A_2). The square itself represents the full composite state space.

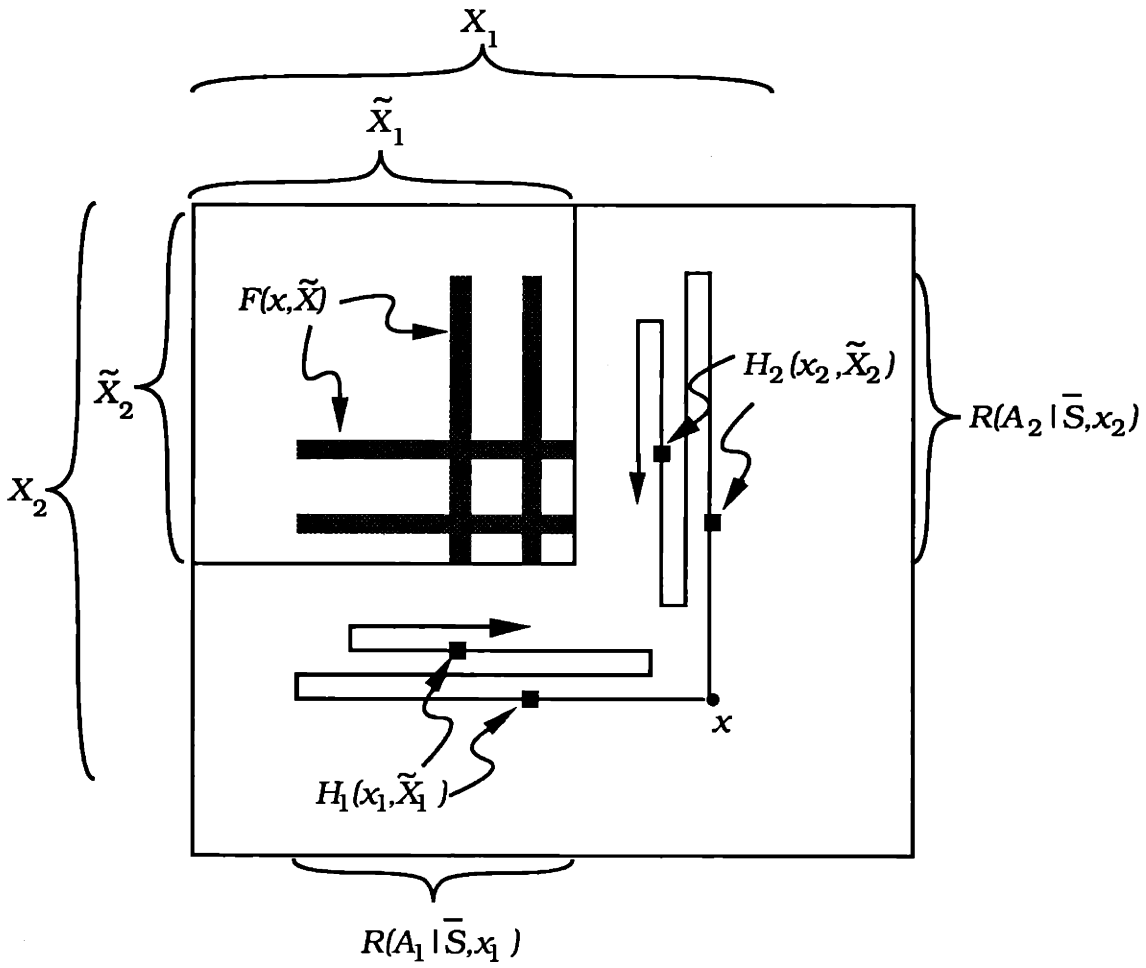

Figure 3.9: Computation of $F(x, \widetilde{X})$: Square dots represent $H_i(x_i, \widetilde{X}_i)$, and heavy lines represent $F(x, \widetilde{X})$.

These sets are illustrated in Figure 3.8. As we will see, much like the importance of the cardinality of Y^c in the complexity of computing the reach of X_0, the cardinality of $\widetilde{X}$ will play an important role in the complexity of testing stability. In particular, we will first test the stability of states in $\widetilde{X}$ and then we will show that the stability of other states can be tested by independent computations in each subsystem. To present such an approach, we also need to define an "extended" transition function which will allow us to characterize state trajectories in A in terms of the states they visit when they enter $\widetilde{X}$. Specifically, we first let $F(x, \widetilde{X})$ denote the set of states that paths from x visit when they enter $\widetilde{X}$ for the first time (see Figure 3.9), i.e,

$$F(x,\widetilde{X}) = \begin{cases} \{x\} & \text{if } x \in \widetilde{X} \\ \left\{ \begin{array}{ll} y \in \widetilde{X} | & \exists \mathbf{x} \in \mathcal{X}(A,x) \text{ such that either } \mathbf{x} = xy \\ & \text{or } \mathbf{x} = xz_1 \cdots z_p y, z_i \notin \widetilde{X}, i = 1, \ldots, p \end{array} \right\} & \text{otherwise} \end{cases} \tag{3.9}$$

Let us now consider how we can compute $F(x, \widetilde{X})$: Let y be a state that a path from x visits when it enters $\widetilde{X}$ for the first time. Then, no event on the path can be a shared event, and therefore both systems cannot enter their respective $\widetilde{X}_i$'s at the same time. Then, when the path enters y, one subsystem must already be in $\widetilde{X}_i$ (and it must have arrived at $\widetilde{X}_i$ using no shared events) while the other enters $\widetilde{X}_i$ as the path enters y. Thus, let $H_i(x, \widetilde{X}_i)$ denote the set of states in $\widetilde{X}_i$ visited by paths from x that use no shared events (see Figure 3.9). That is,

$$H_i(x,\widetilde{X}_i) = \left\{ \begin{array}{ll} y \in \widetilde{X}_i | & \exists \mathbf{x} \in \mathcal{X}(A_i|\overline{S},x) \text{ such that either (i) } x \notin \widetilde{X}_i \text{ and } \mathbf{x} = xy \\ & \text{or (ii) } x \text{ is arbitrary and } \mathbf{x} = xz_1 \cdots z_p y, z_p \notin \widetilde{X}_i \} \end{array} \right\} \tag{3.10}$$

Then we have the following:

Proposition 3.42 (1) If $x \in \widetilde{X}$ then $F(x, \widetilde{X}) = \{x\}$.

(2) If $x = (x_1, x_2) \in \overline{\widetilde{X}}_1 \times \overline{\widetilde{X}}_2$ then

$$F(x, \widetilde{X}) = (H_1(x_1, \widetilde{X}_1) \times (R(A_2|\overline{S}, x_2) \cap \widetilde{X}_2)) \cup ((R(A_1|\overline{S}, x_1) \cap \widetilde{X}_1) \times H_2(x_2, \widetilde{X}_2))$$

(3) If $x = (x_1, x_2) \in \widetilde{X}_1 \times \overline{\widetilde{X}}_2$ then

$$\begin{aligned} F(x, \widetilde{X}) \;=\; & (H_1(x_1, \widetilde{X}_1) \times (R(A_2|\overline{S}, x_2) \cap \widetilde{X}_2)) \\ & \cup (R(A_1'|\overline{S}, x_1) \times (f_2(R(A_2''|\overline{S}, x_2), \overline{S}) \cap \widetilde{X}_2)) \\ & \cup ((R(A_1|\overline{S}, f_1(R(A_1'|\overline{S}, x_1), \overline{S}) \cap \overline{\widetilde{X}}_1) \cap \widetilde{X}_1) \times H_2(x_2, \widetilde{X}_2)) \end{aligned}$$

where A_i' is the same as A_i but with all transitions that take states in $\widetilde{X}_i$ outside of $\widetilde{X}_i$ removed, and A_i'' is the same as A_i but with all transitions that take states in $\overline{\widetilde{X}}_i$ to states in $\widetilde{X}_i$ removed. If any one of the three components of the above equation in the form $Y_1 \times Y_2$ is such that Y_1 or Y_2 is the empty set, then $Y_1 \times Y_2$ is also defined to be the empty set. Finally, if $x \in \overline{\widetilde{X}}_1 \times \widetilde{X}_2$ then simply interchange the indices 1 and 2 in the above equation.

Proof: (1) The proof of this is trivial by the definition of F.

(2) To prove the inclusion, let us consider a trajectory **x** from x to a state $x' = (x_1', x_2') \in F(x, \widetilde{X})$. Let $x'' = (x_1'', x_2'')$ be the state right before x' in **x** so that $x'' \rightarrow^{\sigma} x'$ for some $\sigma \in \Sigma$. Note that all events in this trajectory, including σ are in $\overline{S}$. Suppose that $\sigma \in \Sigma_1$. Then, $x_2'' \in \widetilde{X}_2$, and since also x_2 must have arrived at x_2'' in A_2 only using events that are not shared,

$$x_2'' \in R(A_2|\overline{S}, x_2) \cap \widetilde{X}_2$$

Also, $x_1'' \notin \widetilde{X}_1$ and since also $x_1' \in \widetilde{X}_1$,

$$x_1 \in H_1(x_1, \widetilde{X}_1)$$

Analyzing the case $\sigma \in \Sigma_2$ similarly, we conclude

$$F(x, \widetilde{X}) \subset (H_1(x_1, \widetilde{X}_1) \times (R(A_2|\overline{S}, x_2) \cap \widetilde{X}_2)) \cup ((R(A_1|\overline{S}, x_1) \cap \widetilde{X}_1) \times H_2(x_2, \widetilde{X}_2))$$

To prove the reverse inclusion, let

$$x' = (x'_1, x'_2) \in (H_1(x_1, \widetilde{X}_1) \times (R(A_2|\overline{S}, x_2) \cap \widetilde{X}_2)) \cup ((R(A_1|\overline{S}, x_1) \cap \widetilde{X}_1) \times H_2(x_2, \widetilde{X}_2))$$

Assume, without loss of generality, that

$$\begin{aligned} x'_1 &\in H_1(x_1, \widetilde{X}_1) \\ x'_2 &\in R(A_2|\overline{S}, x_2) \cap \widetilde{X}_2) \end{aligned}$$

Let x''_1 be the state right before x'_1 in a trajectory from x_1 to x'_1 in $A_1|\overline{S}$. Let us now construct the desired trajectory from x to x' in A': First, keep A_2 at x_2 and let x_1 go to x''_1. Second, keep A_1 at x''_1 and let x_2 go to x'_2. Finally, let x''_1 go to x'_1. Since $x_2 \notin \widetilde{X}_2$ and $x''_1 \notin \widetilde{X}_1$, the first time this trajectory enters $\widetilde{X}$, it enters x'. Hence we have proven (2).

(3) Note that the first component of this equation is the same as the first component of the equation in case (2). Note also that the union of the the second and third components is a subset of the second component of the equation in case (2). The reason behind their difference is that in case (3), because $x_1 \in \widetilde{X}_1$, x cannot reach some of the states in

$$(R(A_1|\overline{S}, x_1) \cap \widetilde{X}_1) \times H_2(x_2, \widetilde{X}_2)$$

without going through $\widetilde{X}$. Let us first prove the inclusion using the definitions of x' and x'' in the proof of case (2). If $\sigma \in \Sigma_1$, then the proof is the same as that of case (2). If $\sigma \in \Sigma_2$, then there are two possibilities: First, suppose that while x goes to x' in A with the trajectory $\mathbf{x}$, the projection of $\mathbf{x}$ into X_1, $Pr_1(\mathbf{x})$, does <u>not</u> exit $\widetilde{X}_1$. Then,

$x_1' \in R(A_1'|\overline{S}, x_1)$ and $Pr_2(\mathbf{x})$ must enter $\widetilde{X}_2$ only once and at state x_2'. Therefore,

$$x_2' \in f_2(R(A_2''|\overline{S}, x_2), \overline{S}) \cap \widetilde{X}_2$$

Second, suppose that $Pr_1(\mathbf{x})$ does exit $\widetilde{X}_1$, then

$$x_1' \in R(A_1|\overline{S}, f_1(R(A_1'|\overline{S}, x_1), \overline{S}) \cap \overline{\widetilde{X}}_1) \cap \widetilde{X}_1$$

and the rest of the proof of inclusion is similar to the proof of case (2). Let us now prove the reverse inclusion for each of the three components of the equation. The proof for the first component is the same as the proof for case 2. To prove the reverse inclusion for the second component, let

$$x' = (x_1', x_2') \in R(A_1'|\overline{S}, x_1) \times (f_2(R(A_2''|\overline{S}, x_2), \overline{S}) \cap \widetilde{X}_2)$$

To construct the desired trajectory, we first let x_1 go to x_1'. Note that, thanks to the definition of A_2'', x_2 can go to x_2' without entering $\widetilde{X}_2$ before arriving at x_2'. Thus, we let x_2 go to x_2' in such a fashion. Since also $x_2 \notin \widetilde{X}_2$, the first time this trajectory from x to x' enters $\widetilde{X}$, it enters x'. Finally, let

$$x' = (x_1', x_2') \in (R(A_1|\overline{S}, f_1(R(A_1'|\overline{S}, x_1), \overline{S}) \cap \overline{\widetilde{X}}_1) \cap \widetilde{X}_1) \times H_2(x_2, \widetilde{X}_2)$$

Let x_2'' be the state right before x_2' in a trajectory from x_2 to x_2' in $A_2|\overline{S}$. To construct the desired trajectory, we first let x_1 exit $\widetilde{X}_1$, say to some state $x_1'' \in \overline{\widetilde{X}}_1$. Second, we let x_2 go to x_2''. Third, we let x_1'' go to x_1' and finally we let x_2'' go to x_2'. Since $x_2 \notin \widetilde{X}_2$, $x_1'' \notin \widetilde{X}_1$, and $x_2'' \notin \widetilde{X}_2$, the first time this trajectory from x to x' enters $\widetilde{X}$, it enters x'. Hence we complete the proof of (3). ❑

Thus, in order to compute $F(x, \widetilde{X})$, we first need to compute $H_i(x_i, \widetilde{X}_i)$. Thus, let us concentrate on computing $H_i(x_i, \widetilde{X}_i)$. To do this, note that the state right before

a trajectory enters $\widetilde{X}_i$ is a state that is not in $\widetilde{X}_i$, and note also that all the events up to and including the event that the trajectory enters $\widetilde{X}_i$, are not shared events. So, let $\widetilde{H}_i$ denote the states that are not in $\widetilde{X}_i$ and that x_i can reach without using shared events, i.e.,

$$\widetilde{H}_i = R(A_i|\overline{S}, x_i) \cap \overline{\widetilde{X}}_i \tag{3.11}$$

Then, $H_i(x_i, \widetilde{X}_i)$ is simply the set of states in $\widetilde{X}$ that $\widetilde{H}_i$ can reach with one event which is not a shared event. Thus, we have

$$H_i(x_i, \widetilde{X}_i) = f_i(\widetilde{H}_i, \overline{S}) \cap \widetilde{X}_i = f_i(R(A_i|\overline{S}, x_i) \cap \overline{\widetilde{X}}_i, \overline{S}) \cap \widetilde{X}_i \tag{3.12}$$

Note that the complexity of computing $H_i(x_i, \widetilde{X}_i)$ is $O(n_i)$, and from the equations in Proposition 3.42 we see that the complexity of computing $F(x, \widetilde{X})$ is $O(n_1 + n_2)$. Finally, for the pre-stability algorithm that we present below, we will need to pre-compute $F(x, \widetilde{X})$ for all $x \in X$ and the complexity of this computation is $O(n_1^2 + n_2^2)$ since all we need to do to compute this is to compute $H_i(x_i, \widetilde{X}_i)$, $R(A_i'|\overline{S}, x_i)$, $R(A_i''|\overline{S}, x_i)$, $R(A_i|\overline{S}, x_i)$, etc., for all $x_1 \in X_1$ and $x_2 \in X_2$.

Now, we proceed with constructing an algorithm that computes $V_P \triangleq X_P \cap \widetilde{X}$, where X_P, as defined earlier in this chapter, is the maximal E-pre-stable set. First, consider the following counterpart of Lemma 3.3:

Lemma 3.43 Given $E \subset X$, $x \in \widetilde{X}$ is E-pre-stable iff $x \in E$ or $\bigcup_{y \in f(x,d(x))} F(y, \widetilde{X})$ is E-pre-stable.

Proof: ($\rightarrow$) Assume the contrary. Then $x \notin E$ and there exists $y \in f(x, d(x))$ and $z \in F(y, \widetilde{X})$ such that z is not E-pre-stable. Since $x \notin E$, z is the state that a path from x enters when it enters $\widetilde{X}$ for the first time. Since $E \subset \widetilde{X}$, this path from x to z does not go through E before it reaches z. Finally, since z is not E-pre-stable, x cannot be E-pre-stable either and we establish a contradiction.

($\leftarrow$) If $x \in E$, then clearly x is E-pre-stable. Suppose that $x \notin E$, then thanks to liveness and strong coupling, x is $F(x, \widetilde{X})$-pre-stable. Since $F(x, \widetilde{X})$ is E-pre-stable, $F(x, \widetilde{X}) \cup E$ is also E-pre-stable and then, by Lemma 3.4, x is also E-pre-stable. ❑

Then we have the following:

Proposition 3.44 The following algorithm computes V_P in at most $|\widetilde{X}|$ steps and it has complexity $O(|\widetilde{X}|^2 + n_1^2 + n_2^2)$:

Algorithm Let $X_0 = E$. Iterate

$$X_{k+1} = \{x \in \widetilde{X} | x \in X_k \text{ or } \bigcup_{y \in f(x,d(x))} F(y, \widetilde{X}) \subset X_k\}$$

Terminate when $X_{k+1} = X_k$. Then, $V_P = X_k$.

Proof: The algorithm terminates in at most $|\widetilde{X}|$ steps since the iteration is only on states in $\widetilde{X}$. Let us say that it terminates in r steps. Then, by Lemma 3.43, $X_r = X_P \cap X$. The first part of the bound on computation follows from the maximum number of iterations and the fact that all states in $\widetilde{X} \cap \overline{X}_k$ are considered at each iteration. The rest of the bound is due to the bound on the complexity of computing $F(x, \widetilde{X})$. ❑

Now let us turn our attention to testing the pre-stability of states that are <u>not</u> in $\widetilde{X}$. It is easier to first construct states that are <u>not</u> pre-stable. So, if we let $W_P = \overline{X_P} \cap \widetilde{X}$, then a state in $\overline{\widetilde{X}}$ is <u>not</u> pre-stable if it can reach a state in W_P only using events in $\overline{S}$. Thus, we have the following:

Proposition 3.45 The set X_P can be computed by the following equation:

$$X_P = \overline{R(A_1^{-1}|\overline{S}, Pr_1(W_P)) \times R(A_2^{-1}|\overline{S}, Pr_2(W_P))} \cup V_P$$ ❑

Finally, the maximal set of E-stable states, X_S, can be computed in a straightforward fashion using Equation 3.4 and the algorithm in Proposition 3.38. The complexity

associated with this computation is also $O(|\widetilde{X}|^2+n_1^2+n_2^2)$. The savings in complexity, as compared to applying Equation 3.4 directly, is quite substantial. For example, if we let $n_1 = n_2 = 100$ and $|\widetilde{X}_1| = |\widetilde{X}_1| = 20$, then the complexity of the approach in this section is in the order of 10,400. Compared to $(10{,}000)^2$, this reflects dramatic savings.

3.4.3 Stabilizability of a Composite of Two Systems

In this section, we address the problem of stabilizing control of a composite system. We will assume that only shared events can be disabled, i.e. we will assume that $d(x) \cap \overline{e(x)} \subset S$ for all $x \in X$. This assumption is perhaps somewhat restrictive, however, it can be relaxed at the expense of additional complexity. In the same way in which we extended the pre-stability algorithm in order to construct a pre-stabilizability algorithm, we now extend our pre-stability algorithm for the composition of two systems as follows (recall that in the following, $P(E)$ denotes the maximal E-pre-stabilizable set):

Proposition 3.46 The following algorithm computes $P(E) \cap \widetilde{X}$ in at most $|\widetilde{X}|$ steps and it has complexity $O(|\widetilde{X}|^2 + n_1^2 + n_2^2)$:

Algorithm Let $X_0 = E$. Iterate:

$$P_{k+1} = \left\{ x \in \widetilde{X} \mid \begin{array}{l} (e(x) \neq \emptyset \text{ and } \bigcup_{y \in f(x,e(x))} F(y, \widetilde{X}) \subset X_k) \text{ or} \\ \quad (e(x) = \emptyset \text{ and } \exists \sigma \in d(x) \text{ such that } \bigcup_{y \in f(x,\sigma)} F(y, \widetilde{X}) \subset X_k) \end{array} \right\}$$

$$X_{k+1} = X_k \cup P_{k+1}$$

Terminate when $X_{k+1} = X_k$. Then, $P(E) \cap \widetilde{X} = X_k$.

Proof: Straightforward. ❑

Since only shared events are controllable, we can directly use Equation (3.45) in order to compute $P(E)$, i.e., if we let $V_P = P(E) \cap \widetilde{X}$ and $W_P = \overline{P(E)} \cap \widetilde{X}$, then

$$P(E) = \overline{R(A_1^{-1}|\overline{S}, Pr_1(W_P)) \times R(A_2^{-1}|\overline{S}, Pr_2(W_P))} \cup V_P \tag{3.13}$$

Let us now turn our attention to computing the maximal E-stabilizable set $S(E)$. As we present below, $S(E)$ can be computed in a straightforward fashion using the basic idea of the algorithm in Proposition 3.36 together with the algorithm in Proposition 3.38 and the algorithm in the above proposition: Recall from Section 3.3 that computing $S(E)$ involves computing the maximal sustainably (f,u)-invariant subset of Q, $I(Q)$, given $Q \subset X$, and computing $I(Q)$ involves computing the maximal (f,u)-invariant subset of Q, $T(Q)$. Similarly, given $Q \subset \widetilde{X}$, we let $T'(Q)$ denote the maximal set in Q which satisfies the following: We can find a state feedback so that for each trajectory in the closed loop system that starts from a state in $T'(Q)$, whenever this trajectory enters $\widetilde{X}$, it only enters states in $T'(Q)$. Note that the definition of T' does not care about visiting states in $\overline{\widetilde{X}}$. Then, as stated below, we can characterize $T'(Q)$ using $T(Q \cup \overline{\widetilde{X}})$:

Proposition 3.47 Given $Q \subset \widetilde{X}$, the following equation characterizes $T'(Q)$:

$$T'(Q) = T(Q \cup \overline{\widetilde{X}}) \cap \widetilde{X}$$

Proof: ($\subset$) Clearly, $T'(Q) \subset \widetilde{X}$. Let K be a feedback that satisfies the definition of T'. Then,

$$T'(Q) \subset R(A_K, T'(Q)) \subset T(Q \cup \overline{\widetilde{X}})$$

Therefore, $T'(Q) \subset T(Q \cup \overline{\widetilde{X}}) \cap \widetilde{X}$.

($\supset$) There exists a state feedback such that all trajectories from states in $T(Q \cup \overline{\widetilde{X}})$ stay in the set $T(Q \cup \overline{\widetilde{X}})$ in the closed loop system. Then, in the closed loop system,

all trajectories from $T(Q \cup \widetilde{\overline{X}}) \cap \widetilde{X}$ only enter $T(Q \cup \widetilde{\overline{X}}) \cap \widetilde{X}$ when they enter $\widetilde{X}$. Therefore, $T'(Q) \supset T(Q \cup \widetilde{\overline{X}}) \cap \widetilde{X}$. ❑

Recall that $T(Q) = \overline{R(A_{\emptyset}^{-1}, \overline{Q})}$. Similarly, we have

$$T'(Q) = \overline{R(A_{\emptyset}^{-1}, \overline{Q} \cap \widetilde{X}) \cap \widetilde{X}} \tag{3.14}$$

Like $T'(Q)$, we let $I'(Q)$ denote the maximal set in Q which satisfies the following: We can find a state feedback so that each trajectory in the closed loop system satisfies the above condition stated for T' and in addition, the closed loop system is alive. We have the following:

Proposition 3.48 Given $Q \subset \widetilde{X}$, the following equation characterizes $I'(Q)$:

$$I'(Q) = I(Q \cup \widetilde{\overline{X}}) \cap \widetilde{X}$$

Proof: Similar to the proof of Proposition 3.47. ❑

Also, similar to the algorithm in Proposition 3.34 for computing $I(Q)$, we have the following algorithm for computing $I'(Q)$ where we use T' instead of T and F instead of f:

Proposition 3.49 Given $Q \subset \widetilde{X}$, the following algorithm computes $I'(Q)$ in at most $|\widetilde{X}|$ steps and it has complexity $O(|\widetilde{X}|^2 + n_1^2 + n_2^2)$:

Algorithm Let $X_0 = Q$. Iterate

$$X_{k+1} = T'(\{x \in X_k | \exists \sigma \in d(x) \text{ such that } \bigcup_{y \in f(x,\sigma)} F(y, \widetilde{X}) \subset X_k\})$$

Terminate when $X_{k+1} = X_k$. Then, $I'(Q) = X_k$. ❑

Then we have the following counterpart of the algorithm in Proposition 3.36:

Proposition 3.50 The following algorithm computes $S(E) \cap \widetilde{X}$ in at most $|\widetilde{X}|$ steps and it has complexity $O(|\widetilde{X}|^3 + n_1^2 + n_2^2)$:

Algorithm Let $X_0 = X$. Iterate

$$X_{k+1} = I'(P(E \cap X_k) \cap \widetilde{X})$$

Terminate when $X_{k+1} = X_k$. Then $S(E) \cap \widetilde{X} = X_k$. ❑

Finally, if we let $V_S = S(E) \cap \widetilde{X}$ and $W_S = \overline{S(E)} \cap \widetilde{X}$, then

$$S(E) = \overline{R(A_1^{-1}|\overline{S}, Pr_1(W_S)) \times R(A_2^{-1}|\overline{S}, Pr_2(W_S))} \cup V_S \tag{3.15}$$

Clearly, the complexity of computing $S(E)$ is $O(|\widetilde{X}|^3 + n_1^2 + n_2^2)$. The savings on complexity in this case are even greater than before. For example, if we let $n_1 = n_2 = 100$ and $|\widetilde{X}_1| = |\widetilde{X}_1| = 20$, then the complexity of the approach in this section is on the order of 18,000. Compared to $(10{,}000)^3$, this reflects dramatic savings.

This approach can also be used to test stability of systems that are composed of more than two subsystems. Let us outline, without detailed proof, one straightforward approach and the related issues for addressing this problem:

- Given m subsystems $A_i = (G_i, f_i, d_i, e_i)$, for $i \in M \triangleq \{1, \ldots, m\}$, let

  $$A = \|_{i \in M} A_i \triangleq A_1 \| \cdots \| A_m$$

  Let $S = \bigcup_{i,j \in M} (\Sigma_i \cap \Sigma_j)$, and given E, let

  $$\widetilde{X} = \widetilde{X}_1 \times \cdots \times \widetilde{X}_m \triangleq (Y_1^c \cup Pr_1(E)) \times \cdots \times (Y_m^c \cup Pr_m(E))$$

  where Y_i^c is the set of states in X_i from which an event shared with at least one other subsystem is defined. Let $m_i = |\widetilde{X}_i|$. Also, given $P \subset M$, let $A_P = \|_{i \in P} A_i$ denote the composition of subsystems whose indices are in P (with some abuse of notation, we will say that these are subsystems that are in P), and let S_P denote the events shared between the set of systems P and $\overline{P}$.

- We generalize our strong coupling definition to this case by requiring that any sufficiently long string of events includes at least one event from each subsystem, or equivalently, for any $P \subset M$, A_P and $A_{\overline{P}}$ are strongly coupled in the sense of Definition 3.40. We also assume that if an event is controllable, then it must be shared by at least two subsystems.

- If the A_i are strongly coupled, testing E-stability of A efficiently is straightforward as illustrated below. However, note that the computation of F is more involved since we need to trace trajectories in each subsystem. For example, given $x = (x_1, \ldots, x_m) \in \overline{\widetilde{X}}_1 \times \cdots \times \overline{\widetilde{X}}_m$,

$$\begin{aligned} F(x, \widetilde{X}) \quad = \bigcup_{i \in M} \quad & R(A_1|\overline{S}, x_1) \times \cdots \\ & \times R(A_{i-1}|\overline{S}, x_{i-1}) \times H_i(x_i, \widetilde{X}_i) \times R(A_{i+1}|\overline{S}, x_{i+1}) \times \cdots \\ & \times R(A_m|\overline{S}, x_m) \end{aligned}$$

  In fact, the following is an $O(\prod_{i \in M} m_i + \sum_{i \in M} n_i^2)$ algorithm for computing $V_P = X_P \cap \widetilde{X}$:

  Algorithm Let $X_0 = E$. Iterate

$$X_{k+1} = \{x \in \widetilde{X} | x \in X_k \text{ or } \bigcup_{y \in f(x, d(x))} F(y, \widetilde{X}) \subset X_k\}$$

  Terminate when $X_{k+1} = X_k$. Then, $V_P = X_k$.

  Note that this algorithm is exactly the same as the algorithm in Proposition 3.44.

- Let us now consider testing strong coupling. We first claim the following:

  The A_i are strongly coupled iff A_P and $A_{\overline{P}}$ are strongly coupled for each $P = \{i\}$, where $i \in M$.

The proof of this polynomial test for strong coupling is straightforward: The proof of the forward direction is trivial. To prove the reverse direction, assume the contrary. Then there exists $P \subset M$ such that A_P and $A_{\overline{P}}$ are not strongly coupled. In particular, assume, without loss of generality that A_P has a cycle that consists of events that are not shared with any of the subsystems in $\overline{P}$. Pick some $i \in \overline{P}$ and let $P' = \{i\}$. Then, $A_{P'}$ and $A_{\overline{P}'}$ are not strongly coupled and we establish a contradiction.

- The strong coupling assumption may be too strong for some systems. For example, due to built-in redundancies, a subset of the workstations of a manufacturing system may be capable of producing a part while the others are idle. A weaker condition which could cover such applications (see also the next item) is to require that M can be decomposed into disjoint sets such that the subsystems in each set are strongly coupled and such that the two sets are completely independent. In this case, we can consider each subset independently using results of this section. If such a decomposition is not readily available, we can construct a polynomial algorithm for searching for the existence of such decompositions: We start by using the above algorithm for testing strong coupling. Suppose that for some $P = \{i\}$, A_P and $A_{\overline{P}}$ are <u>not</u> strongly coupled. Then we apply the strong coupling test to $\overline{P}$. We repeat this until we find $P' \subset M$ such that $A_{P'}$ and $A_{\overline{P}'}$ are not strongly coupled but the subsystems in P' <u>are</u> strongly coupled. If furthermore the subsystems in $\overline{P}'$ are strongly coupled, and $A_{P'}$ and $A_{\overline{P}'}$ do <u>not</u> share any events, then we have constructed the desired decomposition. If the subsystems of $A_{\overline{P}'}$ are not strongly coupled, further decompositions may be searched for by applying this algorithm to $\overline{P}'$.

- For some applications, such decompositions may not exist although the strong coupling assumption is not satisfied. For example, suppose that in a flexible manufacturing system Machine 1 and Machine 2 can manufacture Part 1, and Machine 2 and Machine 3 can manufacture Part 2. In this case, Machine 1 and Machine 2 are strongly coupled, but they are not completely independent of Machine 3. Similarly, Machine 2 and Machine 3 are strongly coupled, but they are not completely independent of Machine 3. Therefore, the desired decomposition does not exist. Taking advantage of the sparse structure in such a situation remains an open problem.

Also, recall that we have assumed that only shared events are controllable. However, for some cases, this assumption may be too restrictive. We believe that it is important to do further research to relax this assumption and our thoughts on this issue is explained in Chapter 9.

3.5 Discussion

In this chapter, we have introduced notions of stability and stabilizability for discrete-event systems described by finite-state automata, and we have developed polynomial algorithms to test for stability and stabilizability and to construct maximal stable and stabilizable sets, and for the latter, also a feedback control law that stabilizes a set or a system. Our work in this chapter has drawn on a blend of concepts from computer science and from dynamic systems and control. In particular, the notion of pre-stability used here is well-known in the computer science literature, while the concepts of state feedback, f-invariance, and (f,u)-invariance that are of critical importance for our study of stability and stabilizability, are concepts drawn from

systems and control.

Also, as stated at the beginning of this chapter, one of the goals of this chapter was to establish connections with the computer science literature. In addition to connections to Sifakis that we mentioned before, let us also point to the following two connections: In [14], a system is defined to be self-stabilizing if starting at any unsafe state, it is guaranteed to reach a "safe" state within a finite number of transitions. This is also the same as our notion of stability, and therefore our results of stabilizability can be applied to this case. Also, fair execution sequences, in concurrent systems, are defined as those execution sequences in which each process is executed infinitely often, [11]. This notion is also connected to our notion of stability and our results on stabilizability can tell us how to achieve fairness. Fairness, as we commented before, is also directly related to our notion of strong coupling.

Finally, the stability concepts that we introduced here are of central importance to the rest of this thesis, and, in fact, all the subsequent chapters use stability to address various issues. In some cases, we will see that the notions that we formulate can be tested using stability. For example, we will see in the following chapter that our notion of observability can be tested using stability. In other cases, the notions that we formulate are closely related to stability as they also deal with concepts of error-recovery. For example, our notion of resiliency formulated in the next chapter deals with recovering state estimation capability after the occurrence of observation errors, and our notion of reliability in Chapter 7 is concerned with recovering desired closed-loop event trajectory restriction following one or more system failures.

Chapter 4
Observability

4.1 Motivation and Background

Partial observation problems have been the subject of several investigations in the literature. In particular, Cieslak, et al. [10] formulate a supervisor control problem that can be thought of as a dynamic output compensation problem. Ramadge [37], on the other hand, explicitly addresses the observability problem. In particular, as in this chapter, Ramadge addresses the problem of determining the current state of the system. In his framework, partial observations may be available concerning both the system state and events. In this thesis, we assume what might be thought of as an intermittent observation model: no direct measurements of the state are made, and we only observe a specified subset of possible events, i.e., if an event outside this subset occurs, we will not observe it and indeed will not even know that an event has occurred. The more substantive difference between [37] and this thesis is in the notion of observability that is adopted. In particular, Ramadge requires exact reconstruction of the current state after each system event, while in our work, we allow state ambiguities to develop (as they must if some events are unobserved) but require that these be resolvable after a bounded interval of events. While this difference in formulations is quite fundamental, we will see that the concept of indistinguishability introduced by Ramadge plays an important role in our work as well.

In addition to characterizing observability and constructing observers, we also introduce a notion of stability that we feel is of some importance more generally in characterizing desirable behavior in a DEDS. In particular, we introduce the notion of resiliency for an observer, corresponding to its ability to recover from a finite burst of errors.

In this chapter, since we concentrate on the uncontrolled system with partial knowledge of the event trajectory, we only need to use a portion of the complete model introduced in Chapter 2. Specifically, the systems we consider in this chapter are of the form $A = (G, f, d, h)$ where $G = (X, \Sigma, \Gamma)$. Since we are only observing events in Γ in our automaton A, we will not want it to be possible for our DEDS to generate arbitrarily long sequences of unobservable events, i.e., events in $\overline{\Gamma}$, the complement of Γ. A necessary condition for this is that if we remove the observable events, the resulting automaton $A|\overline{\Gamma} = (G, f, d \cap \overline{\Gamma}, h)$ must not be alive. However, we actually want more than this, namely that every trajectory in $A|\overline{\Gamma}$ is killed in finite time by being forced into a state x for which $d(x) \cap \overline{\Gamma} = \emptyset$. It is not difficult to see that an equivalent condition to our DEDS being unable to generate arbitrarily long sequences of unobservable events is that if we remove the observable events, the resulting automaton $A|\overline{\Gamma} = (G, f, d \cap \overline{\Gamma}, h)$ must be D-stable, where D is the set of states that only have observable transitions defined, i.e., $D = \{x \in X | d(x) \cap \overline{\Gamma} = \emptyset\}$. This is not difficult to check and will be assumed.

Finally, let us introduce some notations that we will find useful:

- Let

$$Y_0 = \{x \in X | \nexists y \in X, \gamma \in \Sigma, \text{ such that } x \in f(y, \gamma)\} \tag{4.1}$$

$$Y_1 = \{x \in X | \exists y \in X, \gamma \in \Gamma, \text{ such that } x \in f(y, \gamma)\} \tag{4.2}$$

$$Y = Y_0 \cup Y_1 \tag{4.3}$$

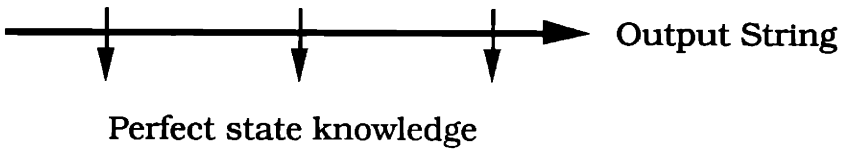

Figure 4.1: Notion of Observability: The state is known perfectly only at the indicated instants. Ambiguity may develop between these but is resolved in a bounded number of steps.

Thus, Y is the set of states x such that either there exists an observable transition defined from some state y to x (as captured in Y_1) or x has no transitions defined to it (as captured in Y_0). Let $q = |Y|$.

- Given $s \in L(A, x)$ for some $x \in X$, let s_f denote the final event in s and let

$$L_f(A, x) = \{s \in L(A, x) \text{ and } s_f \in \Gamma\} \tag{4.4}$$

be the set of strings in $L(A, x)$ that have an observable event as its final event. Similarly, $L_1(A, x)$ denotes the set of strings of $L_f(A, x)$ that contain one observable event, and given some $\gamma \in \Gamma$, $L_\gamma(A, x)$ denotes the set of strings of $L_1(A, x)$ that have γ as the observable event.

4.2 Observability

4.2.1 State Observability

As mentioned in the beginning of this chapter and as illustrated in Figure 4.1, we term a system observable if we can use the observation sequence $\gamma[k]$ to determine

the current state exactly at intermittent (but not necessarily fixed) points in time separated by a bounded number of events. The precise definition is as follows:

Definition 4.1 A is observable if there exists some integer n_o such that $\forall x \in X$, $\forall s \in L(A,x)$ such that $|s| \geq n_o$, there exists a prefix of s, $p \in L_f(A,x)$, such that $|s/p| \leq n_o$, $f(x,p)$ is single valued, and $\forall y \in X, t \in L_f(A,y)$: $h(t) = h(p) \Longrightarrow f(y,t) = f(x,p)$. ❑

This definition states the following: Take any sufficiently long string, s, that can be generated from any initial state x. For an observable system, we can then find a prefix p of s such that p takes x to a unique state and the length of the remaining suffix is bounded by some integer n_o. Also, for any other string t, from some initial state x', such that t has the same output string as p, we require that t takes x' to the same, unique state to which p takes x.

Let us note some very important implications of this definition. First, the string p need not be of length one. Thus, while from the definition we will know the state after p is observed, we may not know it at earlier points. Furthermore, since $p \in L_f(A,x)$, when we do know the state, that state will necessarily lie in Y. That is, since we only observe events in Γ, the only possible times at which we might know the state is at points at which events in Γ occur, i.e., points at which $x[k] \in Y$. Observability is in fact weaker, since in particular, in an observable system, we need not know the state every time it enters Y or even every time it visits a particular state in Y: all we can be assured is that we will know the state at points separated by n or fewer events, and that when we know the state, it will be in Y.

This suggests a straightforward design of an observer that produces "estimates" of the state of the system after each observation $\gamma[k] \in \Gamma$. Each such estimate is a subset of Y corresponding to the set of possible states into which A transitioned when

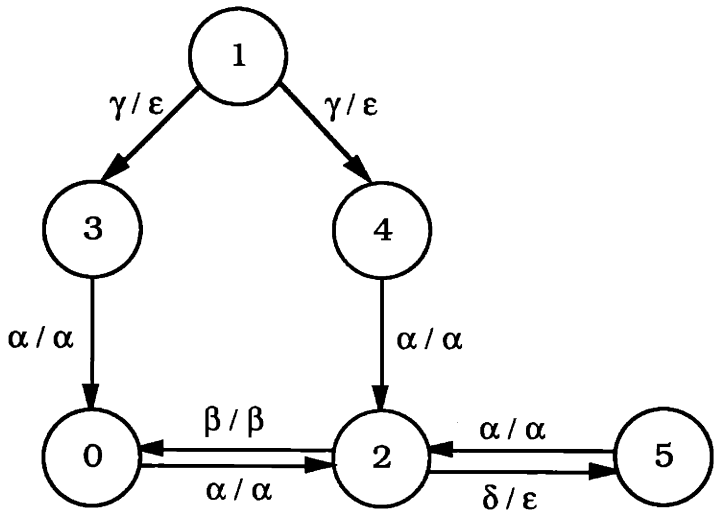

Figure 4.2: A Simple Example

the last observable event occurred. The state space for the observer is a subset Z of 2^Y, and the events and observable events are both Γ. What this observer must do is the following: Suppose that the present observer estimate is $\hat{x}[k] \in 2^Y$ and that the next output is $\gamma[k+1]$. The observer must then account for the possible occurrence of one or more unobservable events prior to $\gamma[k+1]$ and then the occurrence of $\gamma[k+1]$:

$$\hat{x}[k+1] = w(\hat{x}[k], \gamma[k+1]) \triangleq \bigcup_{x \in R(A|\overline{\Gamma}, \hat{x}[k])} f(x, \gamma[k+1]) \tag{4.5}$$

$$\gamma[k+1] \in v(\hat{x}[k]) \triangleq h(\bigcup_{x \in R(A|\overline{\Gamma}, \hat{x}[k])} d(x)) \tag{4.6}$$

The set Z is then in the reach of $\{Y\}$ using these dynamics, i.e., we start the observer in the state corresponding to a complete lack of state knowledge and let it evolve.

Our observer then is the DEDS $O = (F, w, v, i)$, where $F = (Z, \Gamma, \Gamma)$ and i is the identity output function. Consider the system in Figure 4.2. The observer for this system is illustrated in Figure 4.3. Note that the set of allowable events $v(\hat{x}[k])$ defined in (4.6) characterizes all possibilities for the next observable event given the

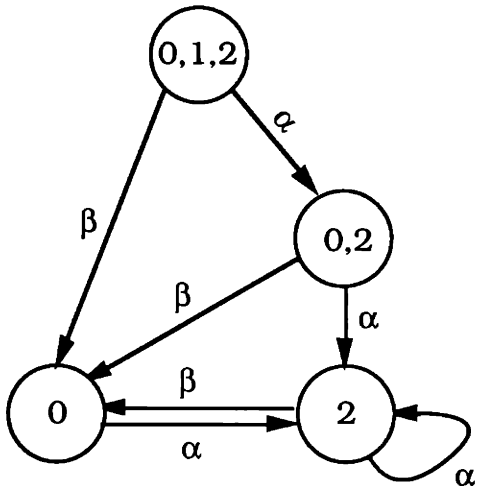

Figure 4.3: Observer for the system in Figure 4.2

set of possible states $\hat{x}[k]$. In general, $v(\hat{x}[k]) \not\equiv \Gamma$ for all $\hat{x}[k]$, i.e., not all sequences in Γ^* can actually occur in our system A. If such an unallowable sequence is observed, an error has obviously occurred. In Section 4.4, we will deal with this in order to define the composition of A and O in our treatment of resiliency. Observability, however, can be considered by examining O by itself. Specifically, let $E = (\bigcup_{x \in Y} \{x\}) \cap Z$ be the singleton states of O. The following result ties observability with stability:

Proposition 4.2 *A is observable iff E is nonempty and O is E-stable.*

Proof: Note first that E must necessarily be nonempty for the system to be observable. Thus we assume that this is true and focus then on necessity and sufficiency of E-stability. To prove necessity, assume the contrary. Then (see Chapter 3), there exists a cycle $\hat{x}_1\hat{x}_2 \cdots \hat{x}_k = \hat{x}_1$ in O for which $|\hat{x}_i| > 1$ for all i. Let s denote the output sequence producing this cycle. Then, an arbitrarily long repetition of this sequence is a feasible output sequence for A. If this occurs, we will never know the current state exactly.

Now suppose that O is E-stable. Therefore, the trajectories from all observer states go through E. Since we also assumed that A cannot generate arbitrarily long sequences of unobservable events, for any output that the system can generate, the observer goes through singleton states at intervals of finite number of events, and thus A is observable. ❑

Note that the observer DEDS in Figure 4.3 is stable with respect to $\{0, 2\}$ so that the system in Figure 4.2 is observable.

It is interesting to contrast our notion of observability with that used in [37]. In particular, in [37] it is required that the state is known at all times. Therefore, it must be that $E = X$ and that once the observer enters E, it is trapped there forever. In contrast, we may have E substantially smaller than X and furthermore, we allow the observer state to leave E, as long as it returns in the future.

Let us also make a first few comments about computational complexity. Note that the cardinality of Z, the observer state space is bounded by 2^q. Thus, using the stability test in Chapter 3 we immediately have an $O(2^{2q})$ test for observability. In Section 4.3, we will provide tighter bounds on the size of Z. Independently of this, however, we can devise an observability test that is polynomial in q. In particular, the reason for the apparent complexity of the test for observability is the size of the observer state space. An important point to note is that the observer is a deterministic automaton, i.e., it tells us exactly the set of possible current states given the observed output. To test for observability, however, all we really want to know is if there are recurring points in time at which all ambiguity in the current state vanishes. Fortunately, it is possible to construct a nondeterministic automaton that captures this with a dramatically smaller state space. Specifically, given A, construct A', a nondeterministic automaton with state space Y and event set Γ such that A'

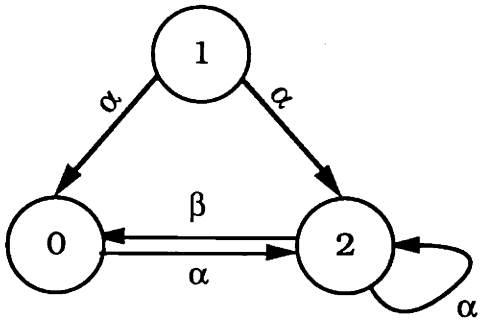

Figure 4.4: A' Corresponding to the Example in Figure 4.2

generates the same output language as A (see Figure 4.4 for A' corresponding to the example in Figure 4.2). Let $P = Y \times Y$ and construct the pair automaton O_P with state space P and event set Γ such that

$$f_{O_P}(p,\gamma) = (f'(x,\gamma) \cup f'(y,\gamma)) \times (f'(x,\gamma) \cup f'(y,\gamma)) \quad (4.7)$$

$$d_{O_P}(p) = d'(x) \cup d'(y) \quad (4.8)$$

where f' is the transition map of A', $p = (x,y) \in P$, $\gamma \in \Gamma$, and we define $f'(x,\gamma)$ as $\emptyset$ if $\gamma \notin d'(x)$. Note that since it is nondeterministic, O_P is certainly not an observer for A. However, if its state ever evolves deterministically to a state of the form (x,x), the automaton A must be in state x. Thus, we have:

Proposition 4.3 *A is observable iff O_P is E_P-stable where $E_P = \{(x,x)|x \in Y\}$*

Proof: Straightforward by assuming contrary in each direction. ❑

Since $|P| = q^2$, this gives us a test for observability that has complexity $O(q^4)$. This also leads to an upper bound on the maximum number of transitions it takes to reach a singleton state, n_o (see Definition 4.1):

Corollary 4.4 *If A is observable, then $n_o \leq nq^2$.*

Proof: If A is observable, then all trajectories from an observer state reach a singleton

state in at most q^2 transitions, since otherwise O_P is not E_P-stable. In addition, between each observable transition, there can be at most n unobservable transitions. Therefore, an upper bound for n_o is nq^2. ❑

Let us analyze the two components of n_o separately. Specifically, let n_{o1} be the maximum number of transitions it takes a path from the initial state Y of the observer to enter a singleton state for the first time, and let n_{o2} be the length of the longest chain of unobservable events in A. It is straightforward to see that a relatively tight bound on n_o is $n_{o1} + n_{o1}n_{o2} = n_{o1}(1 + n_{o2})$. For example, in Corollary 4.4 we had $n_{o1} = q^2$ and $n_{o2} = n$. In fact, a tighter upper bound for n_{o1} is $\frac{q(q-1)}{2} + 1$ since there are at most $\frac{q(q-1)}{2}$ pairs in $Y \times Y$ so that each pair consists of distinct elements of Y and so that no two pairs (x_1, y_1) and (x_2, y_2) are such that $\{x_1, y_1\} = \{x_2, y_2\}$. The following class of systems illustrate that this bound on n_{o1} is quite tight:

Example 4.5 Consider the class of systems, indexed by an even integer k, illustrated in Figure 4.5, where all the events are observable so that $Y = X$ and $n = q = k + 1$. Note that γ defines two transitions to state 0, α and β are both defined at states 2 through $\frac{k}{2} - 1$, and α and δ are both defined at states $\frac{k}{2} + 2$ through $k - 1$. We claim that each system in this class of is observable and that there exists a cycle free path in the observer from the observer initial state X to state $\{0\}$ such that the length of this path is $1 + (\frac{k}{2} - 1)(k - 1)$ transitions, and such that no intermediate state on this path is a singleton. We will prove these claims for the special case $k = 6$, and the corresponding system is illustrated in Figure 4.6. Let us first show that this system is observable by assuming the contrary. In this case, there exist two cycles $x_{11} \cdots x_{1j}x_{11}$ and $x_{21} \cdots x_{2j}x_{21}$ in the system such that $x_{11} \in f(x_{11}, s)$ and $x_{21} \in f(x_{21}, s)$ for some s, and such that $x_{1i} \neq x_{2i}$ for all i. Note that state 0 cannot be an element of either of these cycles and that state

6 must be an element of both of these cycles. Without loss of generality, assume that $x_{11} = 6$. The cycle that starts with the state 6 will be termed Cycle 1. Note that there are two events defined at 6: events δ and β. If β is the first event of s then $x_{21} = 2$ so that it can also generate event β. We will assume that the first event of s is β (the proof corresponding to the case with δ as the first event of s is very similar to the proof for the case that we will present). Then, $x_{12} = 1$ and $x_{22} = 3$. Since only α is defined at both of these states, the second event of s must be α. Then, $x_{13} = 2$ and $x_{23} = 4$. Since, again, only α is defined at both of these states, the third event of s must be α. Then, $x_{14} = 3$ and $x_{24} = 5$. After another occurrence of α, $x_{15} = 4$ and $x_{25} = 6$. Since these states share no events, we establish a contradiction. We establish another contradiction carrying out the case when δ is the first event of s, and therefore the system in Figure 4.6 must be observable. With similar reasoning, it can be shown that any element of this class of systems is observable. In order to construct the desired path from the observer initial state to state $\{0\}$, suppose that the string $s = \alpha^5\delta\beta\alpha^3\gamma$ occurs in the system in Figure 4.6. Careful examination of this figure reveals that s takes the observer initial state X to state $\{0\}$ and no proper prefix of s takes X to a singleton state. Evaluating $1 + (\frac{k}{2} - 1)(k - 1)$ at $k = 6$, we get 11 and this is precisely the length of s. For a given k, the corresponding string is

$$\alpha\alpha^{k-2}\delta\beta\alpha^{k-3}\delta\alpha\beta\alpha^{k-4}\delta\cdots\delta\alpha^{\frac{k}{2}-3}\beta\alpha^{\frac{k}{2}}\gamma$$

Since the length of this string is $1 + (\frac{k}{2} - 1)(k - 1)$, n_{o1} is at least $1 + \frac{(q-3)(q-2)}{2}$. Comparing this to $\frac{q(q-1)}{2}$, we see that our bound on n_{o1} is of the correct order and in fact, it is quite tight. Finally, since $n_{o2} = 0$ in this case, $\frac{q(q-1)}{2}$ is also a bound on n_o. ❑

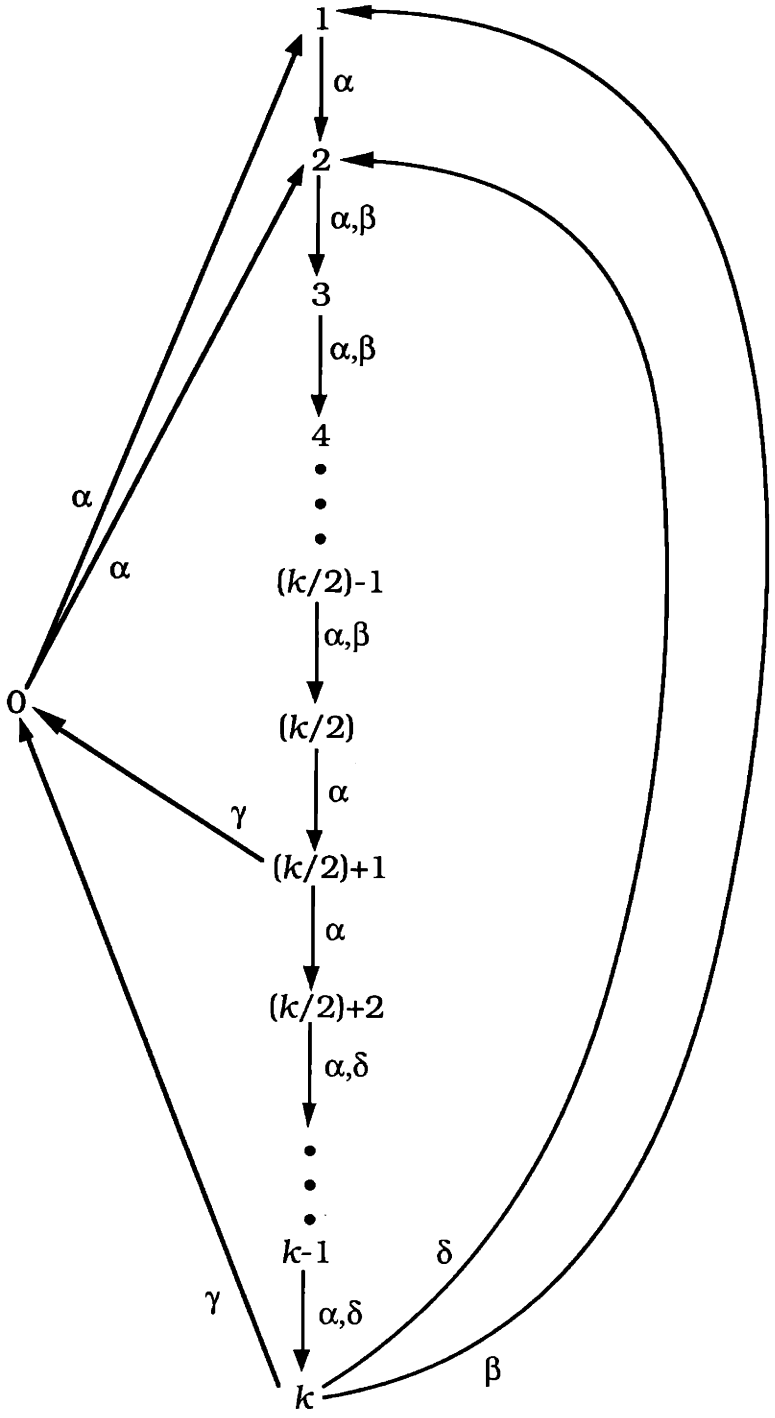

Figure 4.5: A Class of Systems: All events are observable.

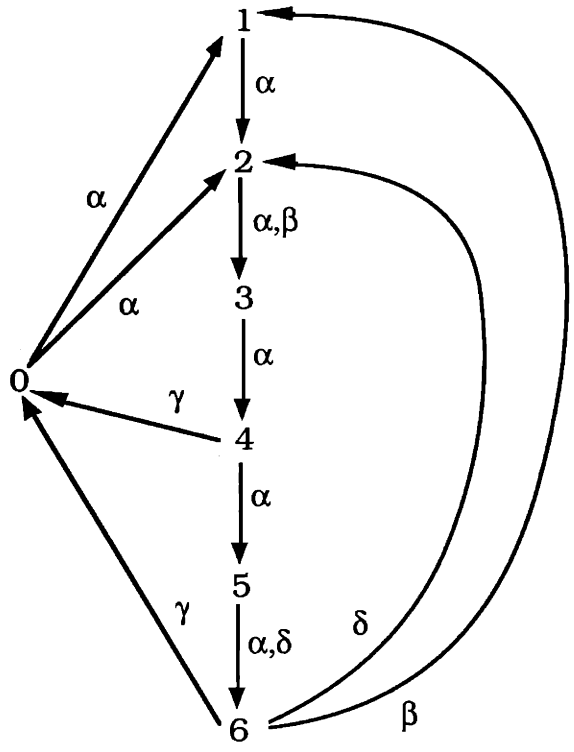

Figure 4.6: Special Case of Figure 4.5: $k = 6$.

4.2.2 Recurrent States and Always-Observability

In this section, we address a problem of finding a set of always-observable states, in the sense that, except perhaps for a finite number of transitions in the beginning, the observer has perfect knowledge of the current state every time the system goes through always-observable states. First of all, an always-observable state has to be a singleton state in the observer. Furthermore, it should not be an element of any other recurrent state of the observer which is not a singleton, where a recurrent state is one that may be visited after an arbitrarily long string of events. States that are on a cycle are certainly recurrent. The following definition also characterizes as recurrent those states that are in between cycles, since these states, although they may be visited at most once, may have this visit occur after an arbitrarily long sequence of transitions. For this reason, they must also be accounted for in characterizing always-observability:

Definition 4.6 A state $x \in X$ is a recurrent state if there exists some $y \in X$, $s \in L(A,y)$, $|s| \geq n$, such that $x \in f(y,s)$. A subset Q of X is termed a recurrent set if all $x \in Q$ are recurrent states. ❑

Clearly, the class of recurrent sets are closed under unions and intersections. Thus, a maximal recurrent set exists and let X_R denote this set. In order to compute X_R, we compute $\overline{X_R}$ which, by the following result, is the maximal set of states stable (in fact, just pre-stable, see Chapter 3) with respect to the dead states in the inverse automaton A^{-1} (see Chapter 2):

Proposition 4.7 $\overline{X_R}$ is the maximal D_i-stable set.

Proof: ($\subset$) Straightforward since all trajectories from $\overline{X_R}$ in A^{-1} are killed in a finite number of transitions.

($\supset$) Suppose x is D_i-stable, then all trajectories from x in A^{-1} are killed in a finite number of transitions. Therefore $x \in \overline{X_R}$. ❑

The following proposition provides a mathematical characterization of always-observability:

Proposition 4.8 A recurrent state $x \in X$ is an always-observable state iff

- x only has observable transitions defined to it, i.e., $d^{-1}(x) \subset \Gamma$, and
- for all $y \in X$, $s \in L_f(A, y)$ such that $|s| \geq nq^2$ and $x \in f(y, s)$,
  - y only goes to x with s, i.e., $f(y, s) = x$, and
  - any other string with the same output as s only goes to x, i.e., for all $z \in X$, $t \in L_f(A, z)$ such that $h(t) = h(s)$, $f(z, t) = x$. ❑

A subset Q of X is termed an always observable set if all $x \in Q$ are always-observable states. A system A is termed a-observable if all trajectories in A visit always-observable states infinitely often.

Clearly, the class of always-observable sets are closed under unions and intersections. Thus, a maximal always-observable set, X_A exists. As explained above, an always-observable state x should only have observable transitions defined to it, and the only recurrent state of the observer that x is in should be the singleton state $\{x\}$:

Corollary 4.9 A recurrent state x is always-observable iff $d^{-1}(x) \subset \Gamma$ and if $\hat{x}$ is a recurrent observer state and $x \in \hat{x}$ then $\hat{x}$ is the singleton state $\{x\}$.

Proof: ($\rightarrow$) The proof for the first statement is obvious. To prove the second statement just assume the contrary.

($\leftarrow$) Straightforward. ❑

As we did before, we can use O_P to check if a state is always observable:

Proposition 4.10 A recurrent state x is always-observable iff $d^{-1}(x) \subset \Gamma$ and if (x, y) for some y is a recurrent state of O_P, then $y = x$.

Proof: Straightforward by assuming the contrary in each direction. ❑

Thus, X_A can simply be computed by performing this $O(q^4)$ test for each recurrent state x such that $d^{-1}(x) \subset \Gamma$. Then, a test for a-observability is just a test for X_A-stability:

Proposition 4.11 A system A is a-observable iff it is X_A-stable.

Proof: Straightforward. ❑

4.2.3 Indistinguishability

Ramadge, in [37], introduces a notion of indistinguishability which he refers to as "possible indistinguishability". This turns out to be an extremely useful notion in our context as well. In this section, we reformulate his definition, present an algorithm for it in our framework, and use it, in Section 4.2.4 to study observability with delay and in Section 4.3 in analyzing the complexity of the observer O.

A pair of states (x, y) is termed to be an indistinguishable pair if they share an infinite length output sequence. Since the observer uses the states in Y, for notational simplicity, we will define indistinguishability for states in Y.

Definition 4.12 Given $x \in X$, let $L_\infty(A, x)$ denote the set of infinite length event trajectories generated from x, and $h(L_\infty(A, x))$ the corresponding set of output trajectories. The pair $(x, y) \in Y \times Y$ is an indistinguishable pair if $h(L_\infty(A, x)) \cap h(L_\infty(A, y)) \neq \emptyset$, i.e., if there is an infinite length output sequence that could have been generated starting from either x or y. ❑

As an example, note that in Figure 4.2, (0,2) is an indistinguishable pair since an infinite string of α's is a possible output sequence from either state. Since we have seen that this system is observable, we now see that the absence of indistinguishable pairs is not required for observability. [1]

The following lemma establishes a recursion for indistinguishable pairs:

Lemma 4.13 (x, y) is an indistinguishable pair iff there exists $s \in L_1(A, x)$, and $t \in L_1(A, y)$ such that $h(s) = h(t)$ and there exists an indistinguishable pair $(z, w) \in f(x, s) \times f(y, t)$.

Proof: ($\rightarrow$) Assume contrary, then for all $(z, w) \in f(x, s) \times f(y, t)$ all output sequences differ in a finite number of transitions. Therefore, (x, y) cannot be indistinguishable and we establish a contradiction.

($\leftarrow$) Straightforward. ❑

A subset I_P of $Y \times Y$ is called an indistinguishable pair set if every element (x, y) of I_P is an indistinguishable pair. Obviously, indistinguishable pair sets are closed under arbitrary unions and intersections. Thanks to the preceding lemma, we have the following for the computation of the maximal indistinguishable pair set:

Proposition 4.14 The following algorithm computes the maximal set of indistinguishable pairs, I_M, and it has complexity $O(q^4)$:

Algorithm Let $I_0 = Y \times Y$ and iterate:

$$I_{k+1} = \{(x, y) \in I_k | \exists s \in L_\gamma(A, x), t \in L_\gamma(A, y) \text{ for some } \gamma \text{ s.t. } f(x, s) \times f(y, t) \cap I_k \neq \emptyset\}$$

Terminate when $I_{k+1} = I_k$. Then $I_M = I_k$.

Proof: Straightforward by using Lemma 4.13. ❑

[1] In general, if there are indistinguishable states, we will not always be able to determine which of these states we were in at some point in the past, but this does not rule out the possibility that we may occasionally know the current state.

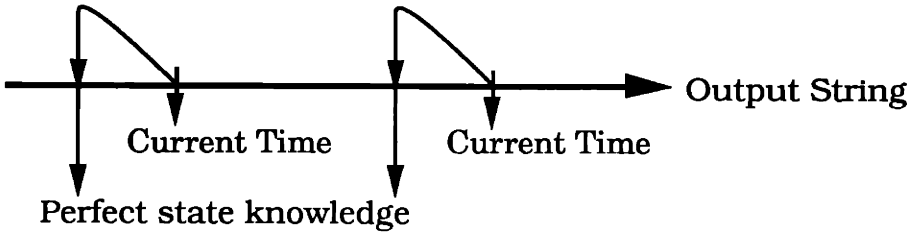

Figure 4.7: Observability with a Delay: The state, a finite number of transitions into the past, is known perfectly at intermittent (but not necessarily fixed) points in time.

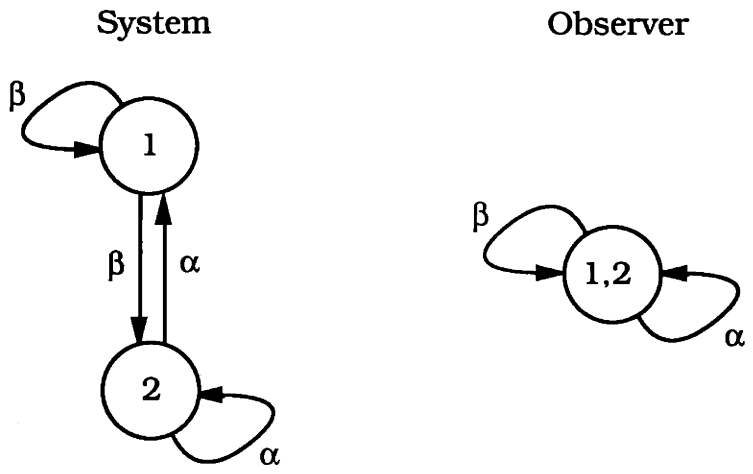

Figure 4.8: Example for WD Observability

4.2.4 Observability with a Delay

For observability with a delay, we require that we have perfect knowledge of the state some finite number of transitions into the past (as opposed to the current state) at intermittent (but not necessarily fixed) points in time (see Figure 4.7). For example, in Figure 4.8, where all events are assumed to be observable, we have a system

which is not observable. When α or β occurs, we do not have perfect knowledge of the current state but if α (respectively, β) occurs, we know that the previous state is state 2 (respectively, state 1). Our formulation of this notion of observability is based on Definition 4.1, in which the prefix p of s characterized the point at which the current state is known perfectly. In the following definition, we use a prefix p_1 of s and a prefix p_2 of p_1, where $h(p_1)$ characterizes the information required to have perfect knowledge of the state at the time in the past just after the occurrence of p_2. For example, in Figure 4.8, for a string $s = \alpha\beta\alpha\alpha$, $p_1 = s$ and $p_2 = \alpha\beta\alpha$. Perfect knowledge of the state is insured by the third item below which (similar to Definition 4.1) states that for all strings t_1 which produce the same output as p_1, the state after t_2 is the same as the state after p_2 where t_2 is the prefix of t_1 that produces the same output as p_2.

Definition 4.15 *A is observable with a delay (WD observable) if $\forall x \in X, s \in L(A,x)$ such that $|s| \geq nq^2$, there exists prefixes $p_1 \in L_f(A,x)$ of s and $p_2 \in L_f(A,x)$ of p_1 such that*

- $|s/p_2| \leq nq^2$,
- $f(x,p_2)$ *is single valued,*
- $\forall y \in X$ *and* $t_1 \in L_f(A,y)$: $h(t_1) = h(p_1) \Longrightarrow f(y,t_2) = f(x,p_2)$ *where* t_2 *is the prefix of* t_1 *such that* $h(t_2) = h(p_2)$. ❑

A test for WD observability can be constructed based on the following: If at any time the observer estimate, $\hat{x}$, is such that all pairs in $\hat{x}$ are distinguishable, then by using future outputs we can distinguish between the states in $\hat{x}$ in a finite number of transitions. For example, in Figure 4.8, since (1,2) is not an indistinguishable pair, in a finite number of transitions, just one transition in this case, we can dis-

tinguish between 1 and 2. In general, a necessary and sufficient condition for WD observability is that the observer is stable with respect to the states that only include distinguishable pairs:

Proposition 4.16 A is WD observable iff O is E_W-stable where

$$E_W = \{\hat{x} \in Z | \text{ there exists no } x, y \in \hat{x}, x \neq y \text{ such that } (x, y) \in I_M\}$$

Proof: ($\rightarrow$) Assume contrary, then there exists a cycle $\hat{x}_1 \cdots \hat{x}_k \hat{x}_1$ in O such that $\hat{x}_i \supset \{x_i, y_i\}$ where $x_i \neq y_i$ and (x_i, y_i) is an indistinguishable pair for all i. Let w be a string such that $x_1 \in f(x_1, w)$ and the event sequence $h(w)$ drives O precisely through the cycle $\hat{x}_1, \ldots, \hat{x}_k, \hat{x}_1$. Referring to Definition 4.15, let $x = x_1$, $s = w^l$ for some large enough l such that $|s| \geq nq^2$. Also pick $y = y_1$. For any prefix $p_1 \in L_f(A, x)$ of s, there exists some $t_1 \in L_f(A, y)$ such that $h(t_1) = h(p_1)$. On the other hand, for all prefixes p_2 of p_1 and corresponding prefix t_2 of t_1 such that $h(t_2) = h(p_2)$, we have that $x_i \in f(x, p_2)$ and $y_i \in f(y, t_2)$ for some i. Since $x_i \neq y_i$ for all i, $f(x, p_2) \neq f(y, t_2)$ and we establish a contradiction with the third item in Definition 4.15, and A cannot be WD observable. Therefore, O must be E_W-stable.
($\leftarrow$) Straightforward ❑

As we did with observability, we use the automaton O_P to construct a polynomial test for WD observability. It is necessary and sufficient to check stability of O_P with respect to the distinguishable pairs:

Proposition 4.17 A is WD observable iff O_P is E_{DP}-stable where $E_{DP} = \{(x, y) \notin I_M\}$.

Proof: Straightforward by assuming the contrary in each direction. ❑

Figure 4.8 is a very simple example that illustrates this result.

4.3 Observer Implementation and Complexity

Recall that the next state of the observer is expressed as a function of the current state and the next event as follows (Equation 4.5):

$$\hat{x}[k+1] = \bigcup_{x \in R(A|\overline{\Gamma}, \hat{x}[k])} f(x, \gamma[k+1]) \tag{4.9}$$

which can also be expressed as:

$$\hat{x}[k+1] = \bigcup_{x \in \hat{x}[k]} \hat{r}(x, \gamma[k+1]) \tag{4.10}$$

where

$$\hat{r}(x, \gamma) = f(R(A|\overline{\Gamma}, x), \gamma) \tag{4.11}$$

Clearly, $\hat{r}$ can be computed beforehand for all $x \in Y$ and $\gamma \in \Gamma$. This computation has $O(|\Gamma|q^2)$ complexity and the result occupies $O(|\Gamma|q^2)$ memory. Thus, computation of the next state of the observer simply becomes taking the union of $\hat{r}(x, \gamma[k+1])$ for all $x \in \hat{x}$, which has $O(q^2)$ complexity. Since also, observability can be tested in polynomial time, computational complexity associated with the observability problem by itself is polynomial.

While testing observability and the implementation of the observer do not require the complete enumeration of the observer state space, this enumeration is needed for other design and analysis problems. This is the case, for example, in the study of stabilization by output feedback which we will address in Chapter 5. Thus, it is of interest to characterize the cardinality of the observer. Unfortunately, even if A is observable (or, for the same matter, a-observable), the observer may have an exponential number of states. As an example, consider the following class of systems which is a slightly modified version of Figure 1 in [46]:

We index this class by an integer i. The system corresponding to $i = 3$ is illustrated in Figure 4.9, where all events are observable. The set of events for this class

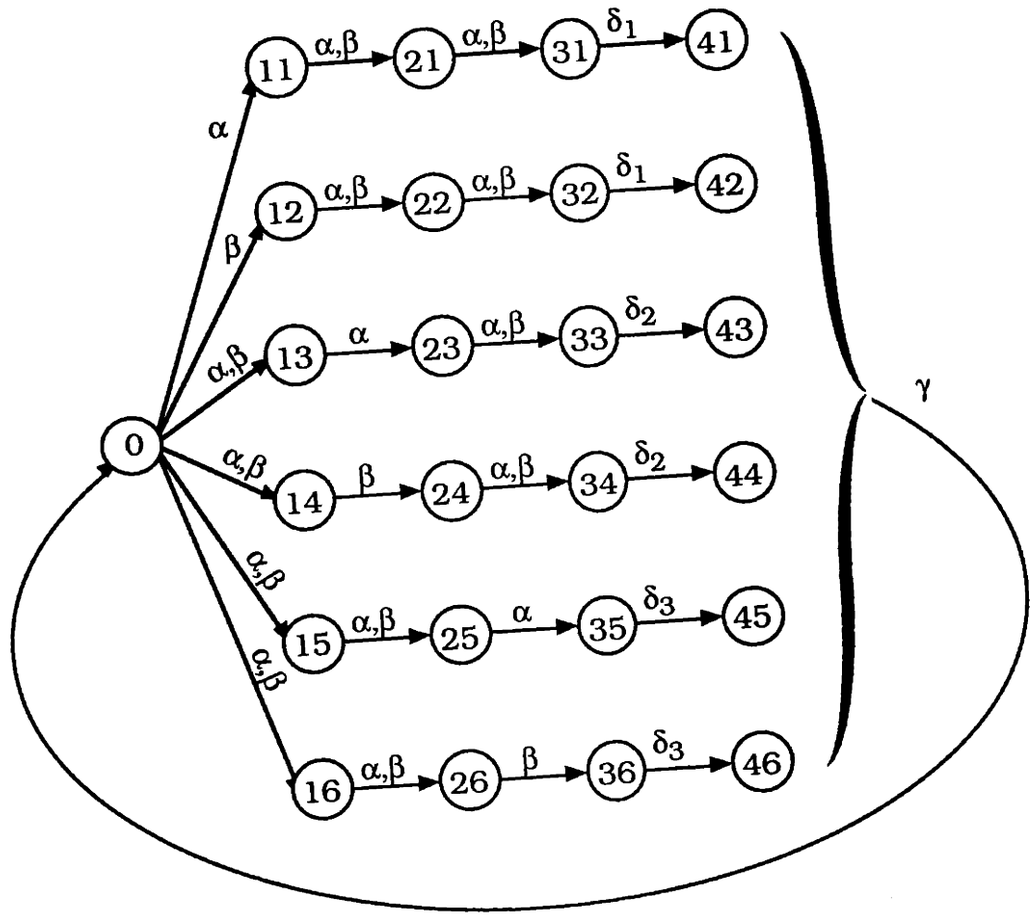

Figure 4.9: Example for Exponential Observer State Space

consists of α, β, γ, and δ_1 through δ_i. There are $2i(i+1)+1$ states and one of them is state 0, whereas the rest is indexed by pairs of integers (j,l) for j ranging from 1 to $i+1$ and l ranging from 1 to $2i$. It is not difficult to check that this system is observable and that 0 is an always-observable state. One can also show that the number of states in the observer is $O(2^i)$. To see why, suppose that the system is in state 0. If α (respectively, β) occurs, then the next state is in the set $\{11, 13, \ldots 16\}$ (respectively, $\{12, \ldots 16\}$). With the next event, the ambiguity in the current state is reduced to four states, then three states, etc. Furthermore, due to the particular way the transitions α and β are defined, the estimates corresponding to each sequence consisting of events α and β are different. It is this fact that leads to the exponential growth in the observer state space.

While the observer state space is exponential for the preceding example, there are many cases in which the cardinality of the state space is much smaller. Thus, it is of interest to characterize structure and characteristics of DEDS that may lead to significantly smaller observer state spaces. In the remainder of this section, we develop a bound on the size of the observer state space which, for certain DEDS, yields a much smaller number than 2^n. First of all, we restrict ourselves to put a bound on Z_R, the recurrent part of the observer state space Z. For any problem such as stabilization, focusing on long-term behavior such as stability, it is only Z_R that is of concern (for example, in output feedback design we can simply let the system evolve without active control during the start-up period—until O enters Z_R—and at that point we can begin to apply feedback).

We begin our analysis by noting that two states x and y are elements of the same recurrent observer state iff the pair (x,y) is indistinguishable in A^{-1}. For example, in Figure 4.9, states 32 and 35 are indistinguishable if we reverse all the transi-

tions in this automaton (since these two states then share the string, for example, $\alpha\alpha\beta(\gamma\delta_1\beta\beta\alpha)^*$). Therefore, the observer estimate after observing $(\alpha\beta\beta\delta_1\gamma)^*\beta\alpha\alpha$ is the set $\{32, 33, 35\}$ which includes the states 32 and 35. We use I_M^{-1} to denote the maximal set of indistinguishable pairs in A^{-1} and this set will play a central role in the computation of our bound.

Let Y_R denote the recurrent part of Y in our original automaton A (i.e., these are elements of Y that may be visited after arbitrarily long sequences of events). For any subset $S \subset Y_R$, we let $\eta(S)$ denote the number of recurrent observer states which include different subsets of S:

$$\eta(S) = |\{Q \subset S | S \cap \hat{x} = Q \text{ for some } \hat{x} \in Z_R\}| \tag{4.12}$$

Then, clearly $|Z_R| = \eta(Y_R)$. To compute a bound, we first find a collection of disjoint subsets of Y_R such that each recurrent observer state is a subset of exactly one element of this collection: First of all, we term a collection $\mathcal{B} = \{B_1, \ldots, B_k\}$ of disjoint subsets B_i of Y_R a Y_R-partition if $\bigcup_i B_i = Y_R$. A Y_R-partition $\mathcal{B}$ is termed a Y_R-distinguishability-partition if each pair indistinguishable in the inverse automaton is in some element of this partition, i.e., for all $(x, y) \in I_M^{-1}$, $\{x, y\} \subset B_i \in \mathcal{B}$. Since all pairs in an observer state are indistinguishable in the inverse automaton, they all must be in the same element of $\mathcal{B}$. For calculating a tight bound, we need to have the elements of $\mathcal{B}$ as small as possible. Thus, a Y_R-distinguishability-partition $\mathcal{B}$ is termed fine if for each $B_i \in \mathcal{B}$, the only B_i-distinguishability-partition is B_i itself. Clearly, there is only one Y_R-distinguishability-partition that is also fine, and we denote this partition by $\mathcal{B}^{\mathcal{F}}$. For Figure 4.9, $\mathcal{B}^{\mathcal{F}}$ consists of the sets $\{0\}$, $\{11, \ldots, 16\}$, $\{21, \ldots, 26\}$, $\{31, \ldots, 36\}$, $\{41\}, \ldots, \{46\}$. We then have the following result:

Proposition 4.18 For all $\hat{x} \in Z_R$, $\hat{x} \subset B_i \in \mathcal{B}^{\mathcal{F}}$ for some i.

Proof: Straightforward. ❑

The following result immediately follows from the above proposition:

Corollary 4.19 Given $S \subset Y_R$, and $\mathcal{B}^{\mathcal{F}} = \{B_i\}$, $\eta(S) = \sum_i \eta(B_i \cap S)$. Therefore,

$$|Z_R| = \sum_i \eta(B_i)$$ ❑

Corollary 4.20 We have the following first bound on the cardinality of the recurrent part of the observer state space:

$$|Z_R| \leq \sum_i (2^{|B_i|} - 1)$$ ❑

The "minus 1" in this equation corresponds to the fact that we can omit the empty set.

While this bound is exponential, it may be much tighter than $2^{|Y|} - 1$ if the partition $\mathcal{B}^{\mathcal{F}}$ is quite fine. Furthermore, if B_i is large, in many cases $\eta(B_i)$ will be much smaller than $2^{|B_i|} - 1$. Two remaining questions then are: (a) how can we compute $\mathcal{B}^{\mathcal{F}}$ efficiently; and (b) can we use the structure of the system to reduce the summands of the bound in Corollary 4.20 even further.

A straightforward algorithm for constructing $\mathcal{B}^{\mathcal{F}}$ is as follows: Let $\pi(x)$ denote the set of states that are indistinguishable with x in the inverse automaton:

$$\pi(x) = \{y | (x, y) \in I_M^{-1}\} \tag{4.13}$$

and construct an automaton, T, with state space Y_R and a single event, say α. Let the transition function $f_T(x, \alpha)$ define transitions from x to all states in $\pi(x)$, i.e., $f_T(x, \alpha) = \pi(x)$. The graph representing this automaton will, in general, consist of disconnected subgraphs. The set of states in each subgraph will in fact be an element of $\mathcal{B}^{\mathcal{F}}$:

Proposition 4.21 The following algorithm constructs $\mathcal{B}^{\mathcal{F}}$:

Algorithm First, pick some $x_1 \in Y_R$, then $B_1 = R(T, x_1)$ is an element of $\mathcal{B}^{\mathcal{F}}$. Second, pick some $x_2 \in Y_R \cap \overline{B_1}$, then $B_2 = R(T, x_2)$ is another element of $\mathcal{B}^{\mathcal{F}}$.

Proceed in this fashion until all the states are picked.

Proof: Clearly, B_1 is at least a subset of an element of $\mathcal{B}^{\mathcal{F}}$. Suppose there exists some $x \in Y_R$ such that $B_1 \cup \{x\}$ is also a subset of the same element of $\mathcal{B}^{\mathcal{F}}$. Then there exists some $y \in B_1$ such that (x, y) is indistinguishable and thus $x \in f_T(y, \alpha)$. But, by the construction of the algorithm, there exists a path from x_1 to y and thus there is also a path from x_1 to x. Hence, $x \in B_1$. Therefore, B_1 is an element of $\mathcal{B}^{\mathcal{F}}$ and similarly, so are all B_i constructed above. Since each state is in some B_i, the above algorithm constructs $\mathcal{B}^{\mathcal{F}}$. ❑

Now, we proceed with computing a possibly tighter bound for Z_R and we use Corollary 4.19 for this. For any $S \subset Y_R$, let $\phi(S, \alpha)$ be the set of states that can reach a state in S with a string that has α as its last and only observable event, i.e.,

$$\phi(S, \alpha) = R(A^{-1}|\overline{\Gamma}, f^{-1}(S, \alpha)) \tag{4.14}$$

Thus, given α, there are $\eta(\phi(S, \alpha))$ observer states that may make a transition, with α, to an observer state which is a subset of S. Thus, if we add these for all such events α, we get an upper bound for $\eta(S)$:

$$\eta(S) \leq \sum_{\alpha \in \Gamma} \eta(\phi(S, \alpha)) \tag{4.15}$$

But, by using Corollary 4.19, we can decompose $\phi(S, \alpha)$ using the partition $\mathcal{B}^{\mathcal{F}}$ and compute η for each part. We thus have the following result, where we assume that $S \subset B_i \in \mathcal{B}^{\mathcal{F}}$ since otherwise we can decompose S itself using the partition:

Proposition 4.22 Given $S \subset B_i \in \mathcal{B}^{\mathcal{F}}$,

$$\eta(S) \leq \min(2^{|S|} - 1, \sum_{\alpha \in \Gamma} \sum_{i} \eta(B_i \cap \phi(S, \alpha)))$$

Proof: Straightforward. ❑

We can apply this to Y_R and thus get the following:

Corollary 4.23 Given $\mathcal{B}^{\mathcal{F}}$,

$$|Z_R| = \eta(Y_R) = \sum_i \eta(B_i) \leq \sum_i \min(2^{|B_i|} - 1, \sum_{\alpha \in \Gamma} \sum_j \eta(B_j \cap \phi(B_i, \alpha))) \qquad \square$$

Now, a recursive application of Proposition 4.22 will give us a bound that gets progressively tighter with each application. If at any time $2^{|S|} - 1$ is a better bound for some set S, then clearly, there is no reason to apply the proposition further after that step. However, this algorithm may in general require an exponential amount of computation if iterated to the fullest. For example, this is the case for the example in Figure 4.9. On the other hand, the algorithm may be terminated at any step by using the bound $2^{|S|} - 1$. Alternatively, the following approximation can be used to compute a bound using less computation.

We now replace the summation over Γ in Proposition 4.22 by an approximation as follows: Given $S, Q \subset Y$, let $\rho(S, Q)$ denote the number of observable events that take states in $R(A|\overline{\Gamma}, Q)$ to states in S:

$$\rho(S, Q) = |\{\alpha \in d(R(A|\overline{\Gamma}, Q)) \cap \Gamma | f(R(A|\overline{\Gamma}, Q), \alpha) \cap S \neq \emptyset\}| \qquad (4.16)$$

First of all, note that

$$\sum_{\alpha \in \Gamma} \eta(B_i \cap \phi(S, \alpha)) \leq \rho(S, B_i \cap \phi(S, \Gamma)) \max_{\alpha \in \Gamma} \eta(B_i \cap \phi(S, \alpha)) \qquad (4.17)$$

Since computing the maximization requires computing $\eta(B_i \cap \phi(S, \alpha))$ for each α, we replace it with $\eta(B_i \cap \phi(S, \Gamma))$ instead. Then,

$$\sum_{\alpha \in \Gamma} \eta(B_i \cap \phi(S, \alpha)) \leq \rho(S, B_i \cap \phi(S, \Gamma)) \eta(B_i \cap \phi(S, \Gamma)) \qquad (4.18)$$

We thus have the following result:

Proposition 4.24 Given $S \subset B_i \in \mathcal{B}^{\mathcal{F}}$,

$$\eta(S) \leq \min(2^{|S|} - 1, \sum_i \rho(S, \tau_i(S))\eta(\tau_i(S)))$$

where

$$\tau_i(s) = B_i \cap \phi(S, \Gamma)$$

Proof: Straightforward. ❑

We can apply this result to Y_R and we get:

Corollary 4.25 Given $\mathcal{B}^{\mathcal{F}}$,

$$|Z_R| = \eta(Y_R) \leq \textstyle\sum_i \min(2^{|B_i|} - 1, \sum_j \rho(B_i, \tau_j(B_i))\eta(\tau_j(B_i))) \quad \text{❑}$$

As before, Proposition 4.24 can be applied recursively. Alternately, one can terminate this algorithm at any step by using the bound $2^{|S|} - 1$. It is not known in general if the full iteration of the algorithm requires a polynomial or exponential number of steps. However, as the following example shows, it requires a linear number of steps for the system of Figure 4.9 and in fact yields $|Z_R|$ exactly:

Example 4.26 For the system in Figure 4.9, $\mathcal{B}^{\mathcal{F}}$ consists of $B_1 = \{0\}$, $B_2 = \{11, \ldots, 16\}$, $B_3 = \{21, \ldots, 26\}$, $B_4 = \{31, \ldots, 36\}$, $B_5 = \{41\}$, $B_6 = \{42\}$, $B_7 = \{43\}$, $B_8 = \{44\}$, $B_9 = \{45\}$, and $B_{10} = \{46\}$. Let us use η_i as a shorthand for $\eta(B_i)$. Then, clearly, $\eta_1 = \eta_5 = \cdots = \eta_{10} = 1$. On the other hand, since $\tau_1(B_2) = \{0\}$ and $\rho(B_2, \tau_1(B_2)) = 2$, $\eta_2 \leq 2\eta_1 = 2$. Similarly, $\eta_3 \leq 2\eta_2 = 4$ and $\eta_4 \leq 2\eta_3 = 8$. Therefore, for this example,

$$|Z_R| \leq 1 + 2 + 4 + 8 + 1 + 1 + 1 + 1 + 1 + 1 = 21$$

and in fact, this is the exact value of $|Z_R|$. ❑

We conclude this section by presenting the following class of systems for which the cardinality of the observer state space is linear in n and our algorithm for computing a bound for $|Z_R|$ also yields $|Z_R|$ exactly:

Example 4.27 Consider the following class of systems, indexed by i (see Figure 4.10 for $i = 4$): The set of events for this class consists of α, β, δ and γ, where all of them are observable. There are $2(i+1)+1$ states and one of them is state 0. The event α (respectively, β) defines a transition from 0 to the odd numbered (respectively, even numbered) states. The event δ defines transitions from all other states to state 0. The event γ defines a transition from state 1 to 4, from 2 to 3, and for all other states j with $j \geq 3$, γ defines a transition from j to $j+2$. These systems are all observable (in fact a-observable), and Z_R is linear in i. For $i = 4$, $\mathcal{B}^{\mathcal{F}}$ consists of $B_1 = \{0\}$ and $B_2 = \{1, \ldots, 10\}$. Clearly, $\eta_1 = 1$. On the other hand, to calculate η_2, we need to know $\eta(\{1, \ldots, 8\})$, which we denote by η_3. Similarly, to calculate η_3, we need to know $\eta(\{1, \ldots, 6\})$, which we denote by η_4. Denoting $\eta(\{1, \ldots, 4\})$ by η_5, and $\eta(\{1, 2\})$ by η_6, and arguing as above, we see that we need to calculate η_6 first. Since $\eta_6 \leq \min(2^2, 2\eta_1) = 2$, $\eta_5 \leq \min(2^4, 2\eta_1 + \eta_6) = 4$. Similarly, $\eta_4 \leq 6$, etc., and thus $\eta_2 \leq 10$. Therefore, $|Z_R| \leq 1 + 10 = 11$, and in fact, this is the exact value of Z_R. ❑

4.4 Resilient Observers

In this section, we introduce the possibility of measurement error in our model and address a problem of resilient observability. Specifically, suppose that the output string that we observe contains errors. Then a major question is how this measurement error affects the behavior of the observer. In particular, does it lead to catastrophic

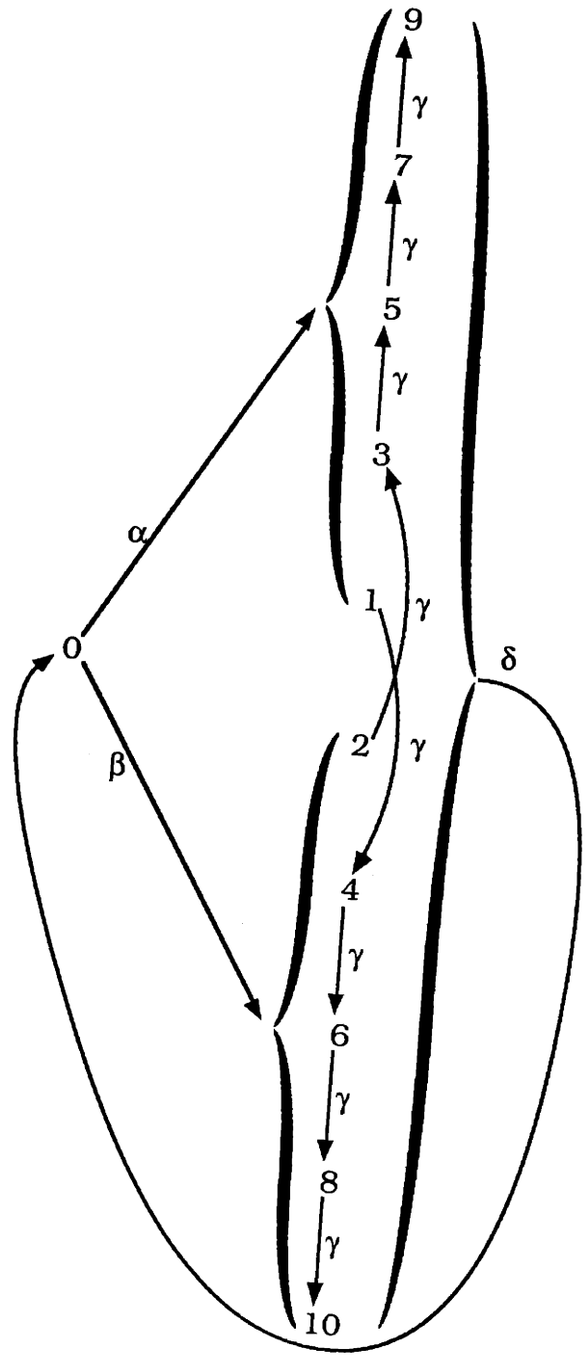

Figure 4.10: Example for Linear Observer State Space

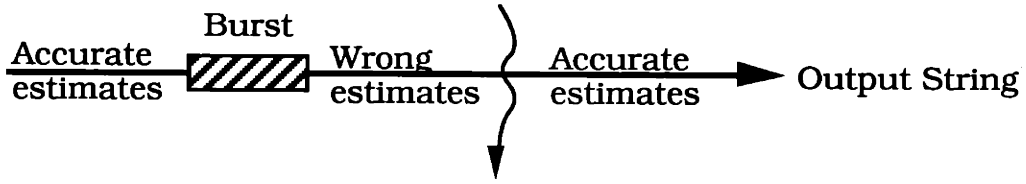

Figure 4.11: Resilient Observability: Following a burst of measurement errors, observer estimates can only be wrong for a finite number of transitions.

error propagation, or does the observer resume desired, correct behavior in a finite number of transitions. Let us consider three types of measurement errors:

- Although the system did not have any transitions, a transition has been mistakenly inserted.
- A transition has been mistaken for another.
- An observable transition has been totally missed in the output string.

An output corrupted with a burst of such measurement errors can be modelled by taking out a finite length string from the output string and replacing it with an arbitrary finite length string over Γ. Our goal here is to design resilient observers so that after a burst of measurement errors, the observer resumes correct behavior in a finite number of transitions, i.e., the actual state of the system is an element of the observer estimate. This is illustrated in Figure 4.11.

Since we allow the burst to be any string in Γ, the corrupted output is not necessarily an output string that can be generated by a state in X, and thus the response of O, as we have specified it so far, is undefined for this erroneous string. Thus, we must extend the observer so that it is defined for all such strings:

Definition 4.28 An observer is a map $\mathbf{B} : \Gamma^* \to 2^Y$ so that for those strings that can occur in A, $\mathbf{B}$ yields the same behavior as O, i.e., for any $x \in X$ and $s \in L_f(A, x)$, we require that

$$\mathbf{B}(h(s)) = \{y \in Y | \exists z \in Y, r \in L_f(A, z) \text{ such that } y \in f(z, r) \text{ and } h(r) = h(s)\} \quad \square$$

There is one special observer that will deserve particular attention. Specifically, not all events γ may be defined at certain states of O. For any such state and event, we then define a transition, back to the "know nothing" state $\{Y\}$—i.e., the observer is simply reset if an inconsistent event occurs. We denote this observer by $O_R = (F, w_R, v_R)$, and mathematically, it is obtained from O as follows:

$$w_R(\hat{x}, \gamma) = \begin{cases} w(\hat{x}, \gamma) & \text{if } \gamma \in v(\hat{x}) \\ \{Y\} & \text{otherwise} \end{cases} \tag{4.19}$$

$$v_R(\hat{x}) = \Gamma \tag{4.20}$$

As before, the initial state of O_R is the state $\{Y\}$. Note that O_R does define a map from Γ^* to 2^Y and thus, by a mild abuse of terminology, we refer to the system or the map as an observer. Note also that O_R is not stable with respect to its singleton states, but $A \parallel O_R$ is stable with respect to the composite states at which the observer is at a singleton state and the system is also at that state:

Proposition 4.29 $A \parallel O_R$ is stable with respect to $\{(x, \{x\}) | x \in Y\}$.

Proof: Straightforward. $\square$

In order to define what we mean by a resilient observer, we also need to define a notion to represent the discrepancy between two strings. There are many ways to define this, all of which depend on the reference point for comparing two strings. Since the actual point that the burst ends is important for our definition of resiliency, we compare two strings from their beginning and we represent their discrepancy by

how much they differ at the end. In particular, we say that the discrepancy between two strings s and t is of length at most i, denoted by

$$\xi(s,t) \leq i \tag{4.21}$$

if there exists a prefix, p, of both s and t such that $|s/p| \leq i$ and $|t/p| \leq i$. Now we can precisely define what we mean by a resilient observer **B**:

Definition 4.30 **B** is a resilient observer if for all strings s that can be generated by A, i.e.,

- $\forall x \in X$,
- $\forall s \in L_f(A,x)$,

for all possible ouput strings t which can be generated by corrupting $h(s)$ with a finite length burst, i.e.,

- $\forall$ positive integers i,
- $\forall t \in \Gamma^*$ such that $\xi(t,h(s)) \leq i$,

and for all possible completions r of s with a suffix of length at least nq^2 (so that the observer has enough time to recover), i.e.,

- $\forall r \in L_f(A,x)$ such that $|r| \geq |s| + nq^2$ and s is a prefix of r,

the observer estimate, in response to the corrupted output $th(r/s)$, includes the current state of the system:

$$f(x,r) \subset \mathbf{B}(th(r/s)) \qquad \square$$

Note that in case of a number of finite bursts that are spaced far enough apart, the estimates of a resilient observer are guaranteed to be correct starting from a finite

number of transitions following each burst, up to the occurrence of the next burst. On the other hand, if the number of correct measurements between each burst is less than q^2, then we cannot guarantee any correct state estimates.

Existence of a resilient observer does not necessarily imply that the system is observable. That is, all we require is that resilient observers resume correct estimates in a finite number of transitions following a burst.

Proposition 4.31 A resilient observer **B**, for A, exists iff $A \parallel O_R$ is E_1-stable, where

$$E_1 = \{(x, \hat{x}) | x \in \hat{x} \in Z\}$$

Proof: ($\rightarrow$) Straightforward by assuming the contrary.
($\leftarrow$) Obvious, since then O_R is a resilient observer. ❑

What this proposition implies is that we only need to look at O_R to check resiliency. The stability condition on O_R simply states that after a finite number of steps following an error, the composite $A \parallel O_R$ returns to a state so that the estimate provided by the state $\hat{x}$ of O does indeed include the true state, x, of A. In general, since the observer state space may be exponential in q, checking stability may be computationally difficult. However, if we have WD observability—which can be checked by a test of polynomial complexity—resiliency is guaranteed:

Lemma 4.32 If A is WD observable then $A \parallel O_R$ is E_1-stable.

Proof: Straightforward by assuming the contrary, since if $A \parallel O_R$ is not E_1-stable, there exists a cycle $(x_1, \hat{x}_1), \ldots, (x_k, \hat{x}_k), (x_1, \hat{x}_1)$ in $Y \times Z$ such that $x_i \notin \hat{x}_i$ for all i. Thus, there exists a cycle $(x_1, y_1), \ldots (x_k, y_k), (x_1, y_1)$ in $Y \times Y$ such that $y_i \in \hat{x}_i$ and (x_i, y_i) is an indistinguishable pair, for all i. By Proposition 4.17, A is <u>not</u> WD observable, and we establish a contradiction. Therefore, $A \parallel O_R$ is E_1-stable. ❑

When we have observability or WD observability, O_R actually has a much stronger property. We need the following definition:

Definition 4.33 A system is resiliently observable (respectively, resiliently WD observable) if the system is observable (respectively, WD observable) and a resilient observer exists. ❑

Consider the observer O_R and its composition, $A \parallel O_R$, with A. Let E_2 be the set of composite states where the observer makes the precise and correct estimate, i.e., $E_2 = \{(x, \{x\}) | x \in X\}$. Then, we have the following:

Proposition 4.34 A is resiliently observable iff $A \parallel O_R$ is E_2-stable.

Proof: Straightforward by using Lemma 4.32. ❑

Finally, the following result shows that we do not need any test for resilient observability, since observability itself is necessary and sufficient for resilient observability:

Proposition 4.35 A is resiliently observable (respectively resiliently WD observable) and O_R is a resilient observer iff A is observable (respectively WD observable).

Proof: ($\rightarrow$) Obvious.

($\leftarrow$) Straightforward using Lemma 4.32. ❑

4.5 Discussion

In this chapter, we have introduced notions of observability, and resiliency for discrete-event systems described by finite-state automata, and we have developed polynomial algorithms to test for observability, resiliency, and to construct resilient observers. We showed that a central element in these concepts is the notion of stability that we considered in Chapter 3. We have also shown that an observer may be implemented

in polynomial time, but the cardinality of its state space may be exponential. Although, this issue is not of practical importance for the problems discussed in this chapter, it is of central importance for problems of stabilization by output feedback that will be addressed in the following chapter.

As we have seen, if a system is observable, the canonic observer O_R is always resilient, i.e., catastrophic error propagation will never occur. In Chapter 6, we address the problem of invertibility, i.e., of deducing the entire event string from the output string, and we also introduce the notion of error recovery or resiliency in that context. In that case, invertibility is not enough to guarantee the existence of a resilient inverter, and further conditions are required to ensure resiliency and the absence of catastrophic error propagation. These notions would seem to be of value in trying to characterize the coordinated behavior of interconnections of DEDS and the ability of the composite to recover from a loss of coordination.

Chapter 5
Stabilizability by Output Feedback

5.1 Introduction and Background

In this chapter, we combine our work on stabilizability and observability to address a problem of stabilization by dynamic output feedback under partial observations. Specifically, we construct stabilizing compensators by cascading an observer and a stabilizing full-state feedback defined on the state space of the observer. While this is a well-established control-theoretic approach, there are several important distinguishing features of the DEDS compensation problem. First of all, in the context of linear systems, we know that observability together with stabilization by state feedback imply the existence of and provide the basis for designing stabilizing output compensators. Thanks to the intermittent nature of observations, the same is not true for the class of DEDS considered in this thesis. Secondly, since the observers we construct for DEDS keep track of all possible states in which the DEDS can be, it is possible to re-cast the output stabilization problem as the stabilization of the observer by state feedback. Finally, a critical issue of particular importance in the DEDS context is computational, and thus it is essential that one characterizes the complexity in designing and implementing a stabilizing compensator.

In this chapter we concentrate on the controlled system with partial knowledge

of the event trajectory. Thus, we use the model introduced in Chapter 2 except for the portion related to the tracking alphabet. Specifically, the systems we consider in this chapter are of the form $A = (G, f, d, e, h)$ where $G = (X, \Sigma, \Gamma, U)$. In Section 5.3, we will use this framework as described in Chapter 2. Up to that point however, we assume the slightly more restrictive framework of [39] in which there is an event subset $\Phi \subset \Sigma$ such that we have complete control over events in Φ and no control over events in $\overline{\Phi}$, the complement of Φ. In this case, we can take $U = 2^{\Phi}$ and

$$e(x) = d(x[k]) \cap \overline{\Phi} \tag{5.1}$$

Furthermore, we assume that $\Phi \subset \Gamma$. These assumptions simplify the presentation of our results, but it is possible to get similar results, at a cost of additional computational complexity, if our assumptions on controllable events are relaxed.

Recall that in Chapter 2 we defined an output compensator as a map $C : \Gamma^* \rightarrow U$. We use such compensators in this chapter. However, one constraint we wish to place on our compensators is that they preserve liveness. First, given an output string s, let $\hat{\mathbf{x}}(s)$ denote the observer estimate. Then, we must make sure that any x reachable from any element of $\hat{\mathbf{x}}(s)$ by unobservable events only is alive under the control input $C(s)$. That is, for all $x \in R(A|\overline{\Gamma}, \hat{\mathbf{x}}(s))$, $d_C(x, s)$ should not be empty. This leads to the following:

Definition 5.1 Given $Q \subset X$, $F \subset \Phi$, F is Q-compatible if for all $x \in R(A|\overline{\Gamma}, Q)$, $(d(x) \cap F) \cup (d(x) \cap \overline{\Phi}) \neq \emptyset$. A compensator C is A-compatible if for all $s \in h(\overline{L}(A))$, $C(s)$ is $\hat{\mathbf{x}}(s)$-compatible. ❑

Suppose that a compensator is such that for all output stings s and t such that the estimate of the current state given s is the same as the estimate given t, the compensator value given s is the same as the value given t. In this case, we can

represent C as a cascade of the observer and a map $K : Z \to U$, which can also be thought of as a state feedback for the observer:

Definition 5.2 A compensator C is O-compatible if for all $s, t \in h(\overline{L}(A))$, such that $\hat{\mathbf{x}}(s) = \hat{\mathbf{x}}(t)$, $C(s) = C(t)$. The corresponding map $K : Z \to U$ such that

$$C(s) = K(v(\{Y\}, s))$$

for $s \in h(\overline{L}(A))$, is termed the observer feedback for C. ❑

We will see in the following section that we can restrict attention to O-compatible compensators in order to address the stabilization problem.

5.2 Two Notions of Output Stabilizability

In this section, we present and analyze two notions of output stabilizability. While it certainly is possible for a system to be output stabilizable without being observable (for example, if it is stable), we will, for simplicity, assume observability. Also, while a system must be stabilizable in order to be output stabilizable, we will not explicitly assume stabilizability. Rather, checking stabilizability will be incorporated into our test for output stabilizability.

The obvious notion of output E-stabilizability is the existence of a compensator C so that the closed-loop system A_C is E-stable. Because of the intermittent nature of our observations, it is possible that such a stabilizing compensator may exist, so that we are sure that the state goes through E infinitely often, but so that we never know when the state is in E. For this reason, we define a stronger notion of output stabilizability that not only requires that the state pass through E infinitely often but that we regularly know when the state has moved into E. We begin with this latter notion which is easier to analyze.

5.2.1 Strong Output Stabilizability

The key to our analysis of strong output stabilizability is that we will know that the state is in E if and only if the observer state is a subset of E:

Definition 5.3 A is strongly output stabilizable if there exists a compensator C and an integer i such that A_C is alive and for all $p \in \overline{L}(A_C)$ such that $|p| \geq i$, there exists a prefix t of p such that* $|p/t| \leq i$ and $\hat{\mathbf{x}}(h(t)) \subset E$. We term such a compensator a strongly output stabilizing compensator. ❑

What this definition states is that in addition to keeping the system alive, the compensator C also forces the observer to a state corresponding to a subset of E at intervals of at most i observable transitions. The next result shows that we can restrict attention to observer feedback:

Proposition 5.4 A is strongly output stabilizable if there exists a state feedback $K : Z \rightarrow U$ for the observer such that X_I in $A \parallel O_K$ is E_{OC}-stable, where $X_I = \{(x, \{Y\}) | x \in X\}$ is the set of possible initial states in $A \parallel O_K$ and where $E_{OC} = \{(x, \hat{x}) \in Y \times Z | \hat{x} \subset E\}$ is the set of composite states for which the system is in E and we know that the current state is in E.

Proof: ($\leftarrow$) Obvious.

($\rightarrow$) If we can find a strongly output stabilizing compensator C that is O-compatible and construct the corresponding observer state feedback K, then X_I is certainly E_{OC}-stable in $A \parallel O_K$.

Let l_i be the set of length i elements of $h(\overline{L}(A))$. Given any strongly output stabilizing compensator C_1 for A, we construct the desired one as follows:

Let $Z_1 = \{\{Y\}\}$ be the set that consists of the initial state $\{Y\}$ of O and let $K(\{Y\}) = C_1(\epsilon)$. Let $S_{11}, \ldots S_{1k_1}$ be a collection of disjoint subsets of l_1 such that

(a) $\bigcup_i S_{1i} = l_1$; (b) for all $\sigma \in S_{1i}$, $v(\{Y\}, \sigma) = \hat{x}_i$ for some $\hat{x}_i \in Z$; and (c) for any S_{1i}, S_{1j}, $i \neq j$, $\hat{x}_i \neq \hat{x}_j$. Let us term such a collection of subsets an l_1-collection. For each $\hat{x}_i$ such that $\hat{x}_i \notin Z_1$, pick some $\alpha_i \in S_{1i}$ and let $K(\hat{x}_i) = C_1(\alpha_i)$. Construct a compensator C_2 such that for all output strings of the form σs, for some $\sigma \in S_{1i}$, $C_2(\sigma s) = C_1(\alpha_i s)$. Clearly, C_2 is a strongly output stabilizing compensator for A. Also, let $Z_2 = Z_1 \cup \bigcup_i \hat{x}_i$ which denotes the set of observer states for which we have defined K so far.

We repeat this construction for l_2, l_3, etc. After step $j-1$, C_j is a strongly output stabilizing compensator for A, and we will have defined K for observer states Z_j that can be reached by $\{Y\}$ with output strings of length at most $j-1$. At step j, let $S_{j1}, \ldots S_{jk_j}$ be the l_j-collection. For each $\hat{x}_i$ such that $v(\{Y\}, S_{ji}) = \hat{x}_i$ and $\hat{x}_i \notin Z_j$, pick some $a_i \in S_{ji}$ and let $K(\hat{x}_i) = C_j(a_i)$. Construct a compensator C_{j+1} such that for all output strings of the form ts, for some $t \in S_{ji}$, $C_{j+1}(ts) = C_j(r_i s)$. Clearly, C_{j+1} is a strongly output stabilizing compensator for A. Also, let $Z_{j+1} = Z_j \cup \bigcup_i \hat{x}_i$.

Proceed in this fashion until, at some step j, $Z_j = Z$, which implies that we have defined a feedback for all observer states. The reach of X_I in $A \parallel O_K$ is alive since by construction $K(\hat{x})$ is $\hat{x}$-compatible. Since also C_j is a strongly output stabilizing compensator for A, the compensator C defined by $C(s) = K(v(\{Y\}, s))$ is a strongly output stabilizing compensator for A. Therefore, X_I in $A \parallel O_K$ is E_{OC}-stable. ❑

Since O describes all the behavior that can be generated by A, we have the following which states that it is necessary and sufficient to check the stability of O with respect to the observer states that are subsets of E, while paying attention to keeping the system alive:

Proposition 5.5 *A is strongly output stabilizable iff there exists a state feedback $K : Z \to U$ for the observer such that O_K is stable with respect to $E_O = \{\hat{x} \in$*

$Z|\hat{x} \subset E\}$ and for all $\hat{x} \in Z$, $K(\hat{x})$ is $\hat{x}$-compatible. Furthermore, if A is strongly output stabilizable then the trajectories in the reach of X_I in $A \parallel O_K$ go through E_{OC} in at most nq^3 transitions.

Proof: A straightforward consequence of Proposition 5.4 and the fact that the radius of O is at most q^3. ❑

As an example, consider the system in Figure 5.1, where $E = \{1, 2\}$ and where all events are observable. Note that in this case, we need to check the stabilizability of the observer with respect to $E_O = \{2\}$. We achieve stability if α is disabled at the observer state $\{0, 2\}$. Proposition 5.5 essentially tells us that we can test strong output stabilizability by testing the observer for stabilizability. The following algorithm performs this test and constructs a feedback for strong output stabilization. It is very similar to our algorithm for pre-stabilizability in Chapter 4:

Proposition 5.6 The following algorithm is a test for strong output stabilizability. It has complexity $O(q^3|Z|)$:

Algorithm Let $Z_0 = E_O$ and iterate:

$$\begin{aligned} P_{k+1} &= \{\hat{x} \in Z | \{\gamma \in v(\hat{x}) | w(\hat{x}, \gamma) \in P_k\} \text{ is } \hat{x}\text{-compatible}\} \\ K(\hat{x}) &= \{\gamma \in v(\hat{x}) | w(\hat{x}, \gamma) \in P_k\} \text{ for } \hat{x} \in P_{k+1} \\ Z_{k+1} &= Z_k \cup P_{k+1} \end{aligned}$$

Terminate when $Z_{k+1} = Z_k = Z^*$. A is strongly output stabilizable iff $Z = Z^*$. The corresponding feedback is K as computed above.

Proof: The proof is straightforward and based on the proof of the algorithm for testing pre-stabilizability in Chapter 3. Computational complexity follows from the fact that the observer has $|Z|$ states and the algorithm terminates in at most q^3 steps. ❑

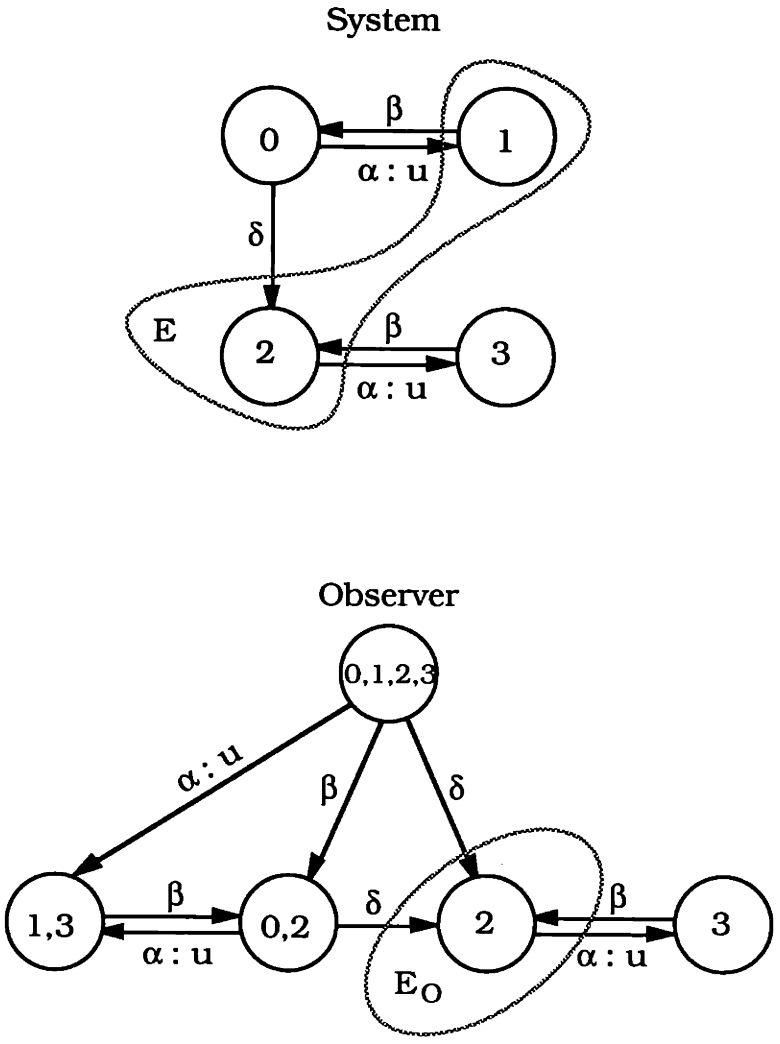

Figure 5.1: Example for Strong Output Stabilizability (all the events are observable)

5.2.2 Output Stabilizability

In this section, we study the following somewhat weaker notion:

Definition 5.7 A is output stabilizable (respectively, output pre-stabilizable) with respect to E if there exists a compensator C such that A_C is E-stable (respectively, E-pre-stable). We term such a compensator an output stabilizing (respectively, output pre-stabilizing) compensator. ❑

Note that this definition implicitly assumes that there exists an integer i such that the trajectories in A_C go through E in at most i transitions. Using this bound, we can show that output pre-stabilizability and liveness are necessary and sufficient for output stabilizability, as is the case for stabilizability and pre-stabilizability (see Chapter 3):

Proposition 5.8 A is output stabilizable iff A is output pre-stabilizable while preserving liveness (i.e., the closed loop system is pre-stable and alive).

Proof: ($\rightarrow$) Obvious.

($\leftarrow$) Let C be an output pre-stabilizing compensator that preserves liveness. Then, for each $x \in X$, there exists an integer i such that the trajectories from x in A_C go though E in at most i transitions. Thanks to our assumption that A cannot generate arbitrarily long sequences of unobservable events, for each $x \in X$, there exists an integer j such that the trajectories from x in A_C go through E in at most j observable transitions. Let j^* be the maximum over all j. Then, we know that the trajectories in A_C go through E in at most j^* observable transitions independently of the initial state. In order to prove our result, we will construct a stabilizing compensator C' using C and j^*. Specifically, given $s \in h(\overline{L}(A_C))$, let s^* denote the suffix of s for which $|s^*| = |s| \bmod j^*$, and let $C'(s) = C(s^*)$. Clearly, $A_{C'}$ is alive. Also, $A_{C'}$ is E-stable

since it is guaranteed to go through E at least once every j^* observable transitions. Therefore, A is output stabilizable. ❑

This result shows us that in order to design a stabilizing compensator, we only need to design a pre-stabilizing compensator. Our construction of a pre-stabilizing compensator involves (a) constructing a modified observer *which keeps track of the states the system can be in if the trajectory has not yet passed through* E, (b) formulating the problem of pre-stabilizing A by ouput feedback as a problem of stabilizing this observer by state feedback, and (c) constructing a pre-stabilizing compensator by using this observer and the state feedback constructed in (b).

To provide the motivation behind our approach, consider the system in Figure 5.1. For output stabilizability, we do not really need to disable α (as we had to for strong output stabilizability). Consider the loop in the observer that consists of the states $\{1,3\}$ and $\{0,2\}$. If the system is in state 1 (respectively, state 2), it is already in E. If the system is in state 3 (respectively, state 0), it makes a transition into E after the next event. Therefore, A is stable and thus is trivially output stabilizable (without disabling any event). This example illustrates the key idea in our analysis of output stabilizability: we must keep track of those state trajectories that have not yet passed through E; if that set becomes empty at some point, we will know that the system has passed through E, although we may not know the point in time at which it did.

The following construction allows us to perform this function: Delete all events in A that originate from the states in E and construct the corresponding observer. Let A_E denote this system and let $O_E = (F_E, w_E, v_E)$ denote its observer. For example, Figure 5.2 illustrates such an automaton and observer for the system in Figure 5.1. The observer O_E captures all the behavior of A until its trajectories enter E. When

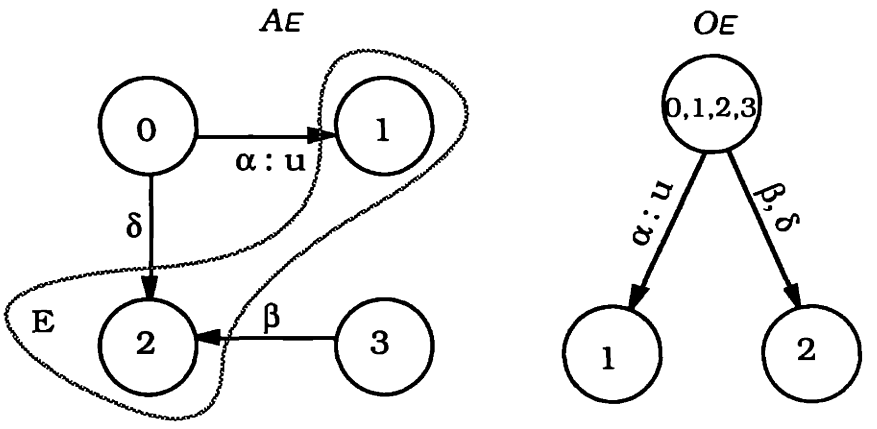

Figure 5.2: Example for A_E and O_E (all the events are observable)

we look at the states of O_E, we see that there are some "trapping" states, each of which is a subset of E and thus has no events defined. Let us consider an event trajectory s in A and the corresponding trajectory $h(s)$ in O_E that starts from the initial state $\{Y\}$. If the trajectory ever evolves to a "trapping" state in O_E, then we know that it has passed through E in A. Other states of O_E may have some elements in E and some elements that are not in E. Let $\hat{x}$ be such a state of O_E, then for a trajectory that evolves to $\hat{x}$, the system can be in one of the states in $\hat{x} \cap \overline{E}$ only if that trajectory has not passed through E yet. Even though O_E keeps track of trajectories that have not passed through E yet, it does not keep track of enough information to design a pre-stabilizing compensator, since, in order to preserve liveness, we also need to know all the states that the system can be in so that we can check if our control input keeps the system alive: The automaton

$$Q = (F_Q, w_Q, v_Q) = O_E \parallel O \tag{5.2}$$

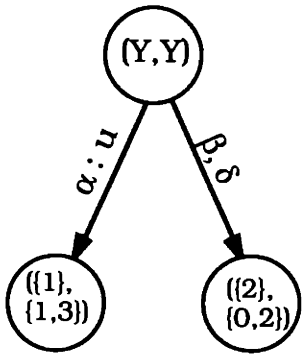

Figure 5.3: Example of the Automaton Q (all the events are observable)

together with the initial state (Y, Y) keeps track of all the information we need for designing an output stabilizing compensator. Note that

$$w_Q((y_1, y_2), \sigma) = (w_E(y_1, \sigma), w(y_2, \sigma)) \tag{5.3}$$

and $v_Q((y_1, y_2)) = v_E(y_1)$. The state space of Q, is $W = R(Q, (Y, Y))$. Figure 5.3 illustrates the automaton Q for the system in Figure 5.1. Note that the number of states of Q is the same as that of O_E. For each state of Q, the second component denotes the set of states that the system can be in, whereas the first component denotes the set of states that the system can be in if the trajectory has not gone through E yet.

The following lemma shows that the problem of output pre-stabilization can be formulated as a problem of pre-stabilization of Q. The key is to find a state feedback K for Q, which we can then adapt to a corresponding compensator for A, and which forces all trajectories in Q_K to have finite length. This in turn will force corresponding trajectories in A to go through E in a finite number of transitions. In doing this, however, we need to make sure that the compensator for A keeps A alive:

Lemma 5.9 A is output pre-stabilizable with respect to E while preserving liveness iff there exists a feedback $K : W \rightarrow U$ such that for all

$$(y_1, y_2) \in R(Q_K, (Y, Y))$$

$K((y_1, y_2))$ is y_2-compatible, and Q_K is pre-stable with respect to its dead states, i.e., with respect to the states y such that $v_{Q_K}(y) = \emptyset$.

Proof: ($\rightarrow$) Straightforward by assuming the contrary.

($\leftarrow$) We claim that the compensator defined by

$$C(s) = K(w_{Q_K}((Y, Y), s))$$

for $s \in L(Q_K, (Y, Y))$ and $C(s) = \Phi$ for all other s, pre-stabilizes A and we prove this as follows: Thanks to the compatibility condition, A_C is alive. Also,

$$h(\overline{L}(A_C)) \subset L(Q_K, (Y, Y))\Gamma^*$$

Given $s \in \overline{L}(A_C)$, if $s \in L(Q_K, (Y, Y))$ then the trajectory may not have passed through E yet. If $s \notin L(Q_K, (Y, Y))$, suppose that $s = p\sigma$ for some $p \in L(Q_K, (Y, Y))$ and $\sigma \in \Gamma$. Since σ is <u>not</u> defined at $w_{Q_K}((Y, Y), p)$, σ could have occurred only if the trajectory has already passed through E. Since also all strings in $L(Q_K, (Y, Y))$ are finite and C preserves liveness, A_C is E-pre-stable. ❑

In order to construct a compensator as proposed by the above lemma, let us first characterize the states in Q that we can "kill" while preserving liveness in A. In particular, let E_Q be the set of states $y = (y_1, y_2) \in W$ so that we can find a y_2-compatible set of events $F \subset \Phi$ which, if used as a control input at y, disables all events defined from y, i.e.,

$$E_Q = \{y = (y_1, y_2) \in W | \exists F \subset \Phi \text{ such that } v_{QF}(y) = \emptyset \text{ and } F \text{ is } y_2\text{-compatible}\} \tag{5.4}$$

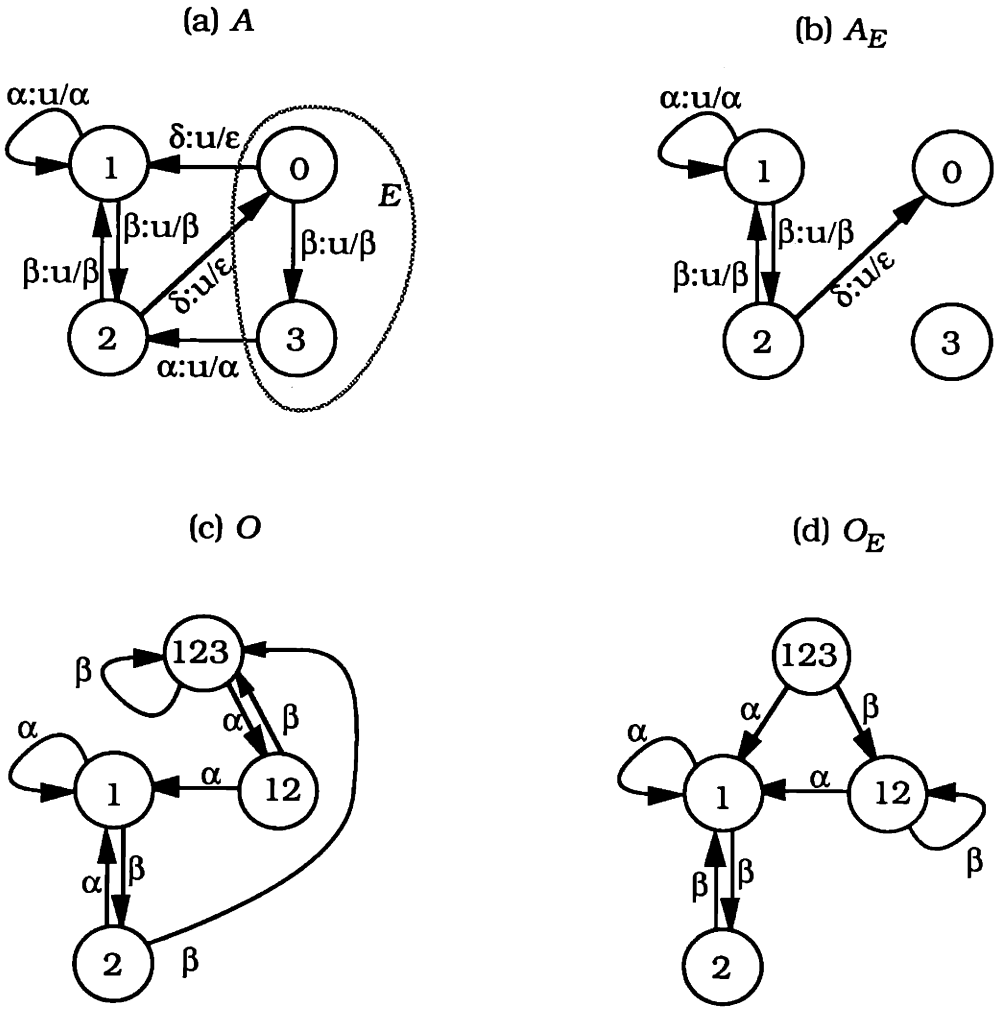

Figure 5.4: Output Stabilizability Example: (a) The system A, (b) A_E, (c) the observer O for A, and (d) the observer O_E for A_E.

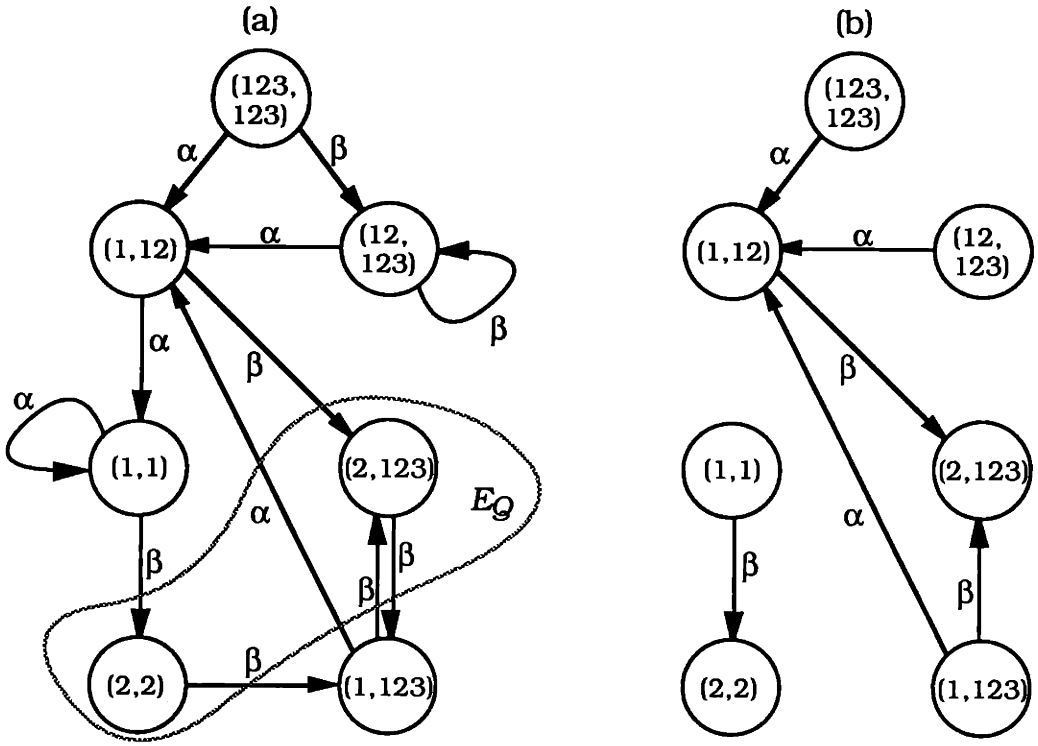

Figure 5.5: Output Pre-stabilization of Figure 5.4 (recall that α and β are both controllable and observable): (a) Automaton Q, and (b) Q_K as computed by the algorithm in Proposition 5.11.

where $v_{QF}(y) = (v_Q(y) \cap F) \cup (v_Q(y) \cap \overline{\Phi})$. For example, consider the system in Figure 5.4, where Figure 5.4(a) illustrates A, (b) illustrates A_E, (c) illustrates the observer O for A and (d) illustrates the observer O_E for A_E. The automaton Q for this example is illustrated in Figure 5.5(a). Note that we can disable β at both of the states (2,123) and (2,2) so that no transitions are enabled in Q at these states, but the states 1, 2, and 3 remain alive in A. Thus, $E_Q = \{(2,123),(2,2)\}$. Therefore, for this example, if we can find a feedback K so that Q_K is E_Q-pre-stable and alive, then, using Q and this feedback, we can construct a compensator that pre-stabilizes A, as we did in the

proof of Lemma 5.9:

Proposition 5.10 A is ouput pre-stabilizable while preserving liveness iff there exists a state feedback K_0 such that Q_{K_0} is E_Q-pre-stable and for all $(y_1, y_2) \in W$, $K((y_1, y_2))$ is y_2-compatible in A. Furthermore, the compensator defined by

$$C(s) = K(w_{Q_{K_0}}((Y,Y), s))$$

for $s \in L(Q_K, (Y,Y))$ and $C(s) = \Phi$ for all other s, pre-stabilizes A, where

$$K(y = (y_1, y_2)) = \begin{cases} F \subset \Phi | v_{QF}(y) = \emptyset \text{ and } F \text{ is } y_2\text{-compatible} & \text{if } y \in E_Q \\ K_0(y) & \text{otherwise} \end{cases}$$

Finally, the trajectories in A_C go through E in at most nq^3 transitions.

Proof: Straightforward using Lemma 5.9 and the fact that the radius of the observer is at most q^3. ❑

We now present an algorithm to test for output pre-stabilizability and to construct the corresponding feedback by appropriately modifying the algorithm in Proposition 5.6 for Q:

Proposition 5.11 The following algorithm is a test for output pre-stabilizability while preserving liveness. It has complexity $O(q^3|W|)$:

Algorithm Let $Z_0 = E_Q$ and for $y = (y_1, y_2) \in E_Q$, let $K(y) = F \subset \Phi$ where F is such that $v_{QF}(y) = \emptyset$ and F is y_2-compatible. Iterate:

$$\begin{aligned} P_{k+1} &= \{y \in W | \{\gamma \in v_Q(y) | w_Q(y,\gamma) \in P_k\} \text{ is } y_2\text{-compatible in } A\} \\ K(y) &= \{\gamma \in v_Q(y) | w_Q(y,\gamma) \in P_k\} \text{ for } y \in P_{k+1} \\ Z_{k+1} &= Z_k \cup P_{k+1} \end{aligned}$$

Terminate when $Z_{k+1} = Z_k = Z^*$. A is output pre-stabilizable iff $(Y,Y) \in Z^*$. The corresponding feedback is K as computed above. ❑

Figure 5.5(b) illustrates the closed loop system Q_K after this algorithm is applied to Q in Figure 5.5(a). In order to construct a compensator that pre-stabilizes the system in Figure 5.4(a), we use the range of (123,123) in Q_K as follows: Initially (i.e., before any observable events are seen so that we are in (123,123) of Q_K), we disable β. After α is observed (so that the state in Q_K is (1,12)), α is disabled, while β is enabled, and finally, after β is observed (corresponding to a transition to the state (2,123)), β is disabled while α is enabled. When α occurs again, we know that all the trajectories have passed through E, and thus we do not care about what the control input is after this point as long as it keeps the system alive.

In Chapter 3 we have termed a feedback to be maximally restrictive if we cannot disable any other event at any state while preserving liveness. We can generate such a feedback using the algorithm in Proposition 5.11 if we choose $K(y)$ such that removing any event from $K(y)$ violates compatibility. In Chapter 3, we have also defined a feedback to be minimally restrictive if, for each state, enabling any event, which is otherwise disabled, violates pre-stability. We have also shown that, a minimally restrictive feedback can be generated from a maximally restrictive one by arbitrarily enabling events (that are otherwise disabled) until pre-stability is violated. In the same manner, we can generate a minimally restrictive feedback from the feedback generated by the algorithm in Proposition 5.11.

We now turn our attention to output stabilizing compensators. Note that if, at some point, we are certain that the trajectory has passed through E then we can force the trajectory to go through E <u>again</u> by starting the compensator over, i.e., by ignoring all the observations to date and using the pre-stabilizing compensator on the new observations (see the proof of Proposition 5.8). In the proof of Proposition 5.8, we computed an integer j^* so that all the trajectories are guaranteed to go through E in

at most j^* transitions independently of the initial state of the system, and so that we can "reset" the output pre-stabilizing compensator after every set of j^* transitions. However, in some cases, it may not be necessary to wait for j^* transitions. In what follows, we present an approach which allows us to detect, as soon as possible, that the trajectory has passed though E.

Given an output pre-stabilizable A, suppose that C is the corresponding compensator and K is the corresponding Q-feedback for C. Recall that for Q_K, no events are defined at states $(y_1, y_2) \in E_Q$, and in general, given some $y = (y_1, y_2) \in R(Q_K, (Y, Y))$, not all events defined at y_2 are defined at y. Given an output trajectory of A_C, let us trace the corresponding trajectory in Q_K starting from the state (Y, Y). Suppose that we observe a transition which is <u>not</u> defined at the current state of Q_K. By the way we have constructed Q_K we know that the occurrence of such a transition implies that the trajectory has already passed through E. This is precisely the mechanism which we use to detect that the trajectory has passed through E. So, given $s \in h(\overline{L}(A_C) \cap L(Q_K, (Y, Y))$, let $y = w_{Q_K}((Y, Y), s)$ and suppose that the next observation is a transition $\sigma \notin v_{Q_K}(y)$, and thus we know that the trajectory has passed through E. At this point, we wish to force the trajectory to pass through E again, but in doing so, we can use our knowledge of the set of states that the system can be in at the time we have detected that the trajectory has passed through E, i.e., $w(y_2, \sigma)$. What we would then like to do is to have Q transition to the state $z = (w(y_2, \sigma), w(y_2, \sigma))$. However, as we have defined it so far, z may <u>not</u> be in W. What we must do in this case is to augment W with all such z's and any new subsequent states that might be visited starting from such a z and using an extension of the dynamics of Q. Specifically, the dynamics of Q given in (5.3) can be defined for arbitrary subsets $y_1, y_2 \subset Y$, as can its restriction w_{Q_K} by feedback.

We modify this definition as follows: if $w_{EK}(y_1, \sigma) = \emptyset$, then we set $w_{Q_K}((y_1, y_2), \sigma)$ to $(w(y_2, \sigma), w(y_2, \sigma))$. Let W^a be the union of the reaches of all states of the form (Y', Y') with $Y' \subset Y$ and define $Q^a = (F^a, w, v)$ where $F^a = (W^a, \Gamma, \Gamma)$. Note that $E_Q \subset W^a$ and $R(Q_K, (Y, Y)) \subset W^a$. If in fact any $z = (Y', Y')$ is pre-stabilizable with respect to $R(Q_K, (Y, Y))$ in Q^a, then we can force the trajectory to pass through E. The next result states that pre-stabilizability of Q is sufficient for being able to do this:

Proposition 5.12 If there exists a feedback K for Q such that Q_K is E_Q-pre-stable and $K(y)$ is y_2-compatible, then there exists a feedback K' such that for any $Y' \subset Y$, $z = (Y', Y')$ is pre-stable with respect to $R(Q_K, (Y, Y))$ in $Q^a_{K'}$ and $K'(y)$ is y_2-compatible for each $y = (y_1, y_2) \in R(Q^a_{K'}, z)$.

Proof: Straightforward by assuming the contrary. ❑

Note that K' can be chosen so that $K'(y) = K(y)$ for all $y \in R(Q_K, (Y, Y))$ and the algorithm in Proposition 5.11 can be used for constructing such a K'.

In order to construct an output stabilizing compensator, we use the above proposition recursively as follows: Let K_0 be a feedback that pre-stabilizes Q and preserves liveness, as can be constructed using the algorithm in Proposition 5.11. Let Z_0 represent the initial state of Q_{K_0} and let W_0 represent the range of Z_0, i.e., the states we may be in when we know that the trajectory has already passed through E:

$$Z_0 = (Y, Y) \tag{5.5}$$

$$W_0 = R(Q_{K_0}, Z_0) \tag{5.6}$$

We then augment Z_0 to include the states to which we may "reset" our compensator, i.e.,

$$Z_1 = Z_0 \cup \{(\hat{x}, \hat{x}) | \hat{x} = w(y_2, \sigma) \text{ for some } y = (y_1, y_2) \in W_0 \text{ and } \sigma \in \hat{v}(y_2, K_0(y))\} \tag{5.7}$$

where $\hat{v}(y_2, K_0(y)) = (v(y_2) \cap K_0(y)) \cup (v(y_2) \cap \overline{\Phi})$. Next, we find a feedback K_1 that satisfies Proposition 5.12 for each $(Y', Y') \in Z_1$. Finally, we let $W_1 = R(Q_{K_1}, Z_1)$. Proceeding in this fashion, we construct W_2, W_3, etc., until $W_{k+1} = W_k = W'$ for some k (note that k must necessarily be finite). Let K' be the corresponding feedback, then

- $Q_{K'}$ is E_Q-pre-stable,
- $K'(y)$ is y_2-compatible for all $y \in W'$, and
- for all $y \in E_Q \cap W'$ and $\sigma \in \hat{v}(y_2, K'(y))$, $(w(y_2, \sigma), w(y_2, \sigma)) \in W'$

Finally, we construct an automaton $Q' = (F', w', v')$ where $F' = (W', \Gamma, \Gamma)$ which includes the transitions to states in Z', i.e.,

$$w'(y, \sigma) = \begin{cases} w_Q(y, \sigma) & \text{if } \sigma \in v_{Q_{K'}}(y) \\ (w(y_2, \sigma), w(y_2, \sigma)) & \text{otherwise} \end{cases} \tag{5.8}$$

$$v'(y) = \hat{v}(y_2, K(y)) \tag{5.9}$$

Then, the compensator defined by

$$C(s) = K'(w'((Y, Y), s)) \tag{5.10}$$

for all $s \in L(Q', (Y, Y))$ stabilizes A. Thus the compensator consists of the automaton Q', started in (Y, Y) and the feedback $K' : W' \to 2^{\Phi}$ so that the desired compensator is given by the Equation (5.10). For example, for the system in Figure 5.4, we need to pre-stabilize the state (12,12) (see Figure 5.6(a)). The resulting automaton Q' that produces the desired compensator is shown in Figure 5.6(b).

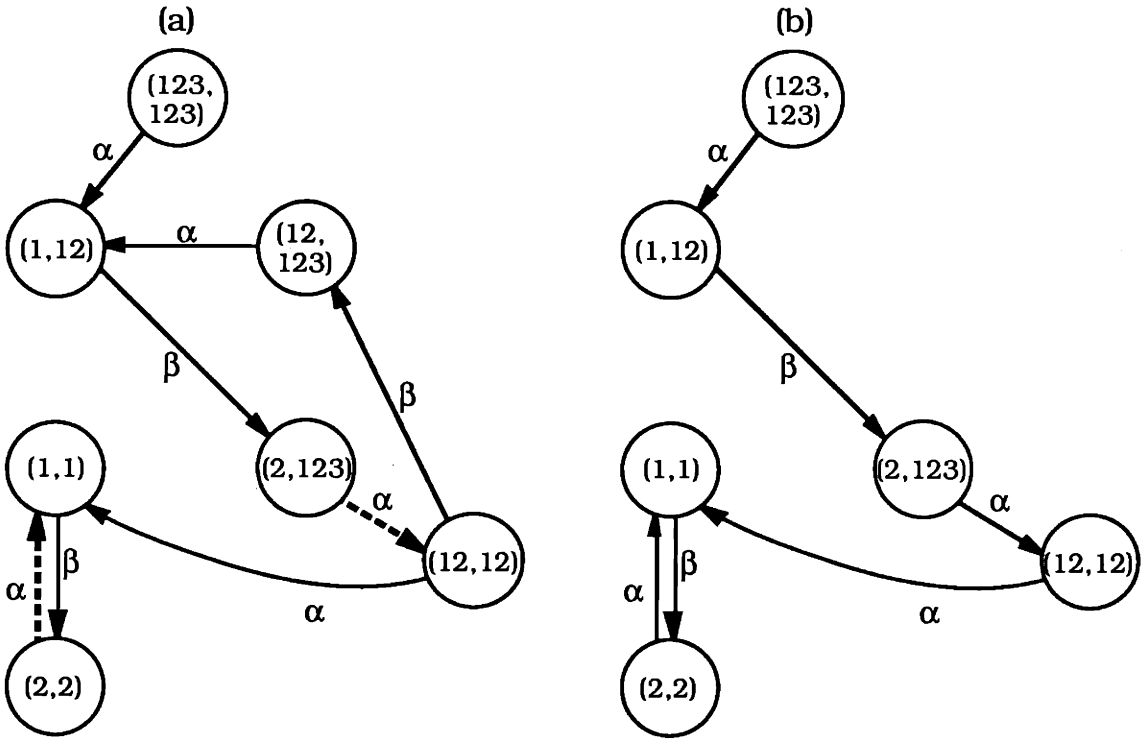

Figure 5.6: Output Stabilization of Figure 5.4 (recall that both α and β are controllable: (a) Adding the new states (through the dashed arcs), (b) Q'.

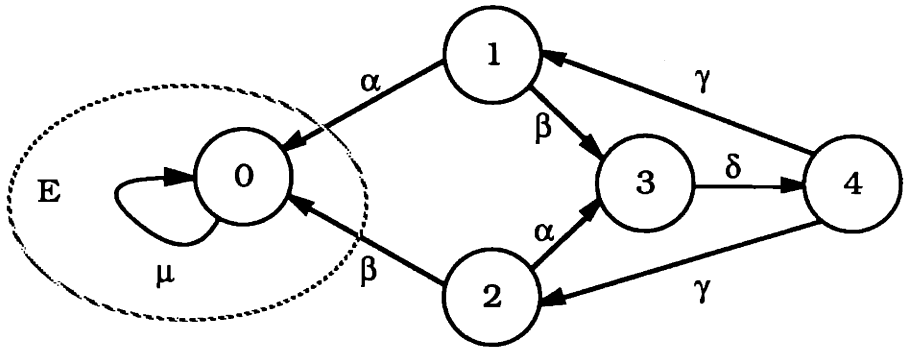

Figure 5.7: Stabilizable, Observable, But Not Output Stabilizable System (all the events are controllable and observable)

5.3 Sufficient Conditions Testable in Polynomial Time

The previous section presented necessary and sufficient conditions for output stabilizability that can be tested in polynomial time in the cardinality of the state space of the observer O (note that the cardinality of the state space of Q is polynomial in the cardinality of the state space of O). However, while in many cases the observer state space may be sufficiently compact, there are worst cases in which the cardinality of the state space of O is exponential in q (see Chapter 4). In this section, we present sufficient conditions that can always be tested in polynomial time in q.

It is well known in linear sytem theory that controllability and observability imply stabilizability using dynamic output feedback. Unfortunately, stabilizability and observability do not imply output stabilizability in our framework. For example, consider the system in Figure 5.7, where all the events are controllable and observable. This system is stabilizable by disabling β at state 1 and α at 2, and it is also observable. However, it is not output stabilizable, since we can never distinguish between states 1 and 2, and thus we cannot selectively disable α or β.

The reason for this phenomenon is that our notion of observability is much weaker than the corresponding system theory notion, since we only require that the state is known intermittently. We start this section by showing that a result similar to that in system theory can be achieved if we assume that after a finite number of transitions, and for each transition after that, we have perfect knowledge of the current state (this condition is equivalent to the notion of observability of Ramadge [37]). Later in this section, we also show how this condition may sometimes be satisfied by choice of feedback. Finally, we present a weaker sufficient condition based on a notion of always observability that we have defined in Chapter 4.

To formalize the first sufficient condition, we need the notion of transition-function-invariance that we have defined in Chapter 3: Let E_w be the maximal w-invariant subset of the set of singleton states of O. If $E_w = \emptyset$ and if O is E_w-stable, then at some finite point the observer state will enter E_w and never leave, so that the state will be known perfectly from that point on.

Proposition 5.13 Suppose that (i) $E \cap E_w = \emptyset$; (ii) A is $E \cap E_w$-stabilizable; (iii) O is E_w-stable, then A is output-stabilizable.

Proof: Let K be a state feedback such that A_K is $E \cap E_w$-stable. We then construct a feedback $\hat{K}$ on O by applying K only when the observer state has moved into E_w, i.e.,

$$\hat{K}(\hat{x}) = \begin{cases} K(x) & \text{if } \hat{x} = \{x\} \in E_w \\ \Phi & \text{otherwise} \end{cases}$$

This feedback clearly stabilizes A, and thus, A is output stabilizable. ❑

As an example, consider the system in Figure 5.8 where $E = \{0\}$. Note that $E_w = \{0, 2\}$, $E \cap E_w = \{0\}$, and the observer, illustrated in Figure 5.9 is E_w-stable. A $E \cap E_w$-stabilizing feedback is one that disables α at state 2. Thus, an output

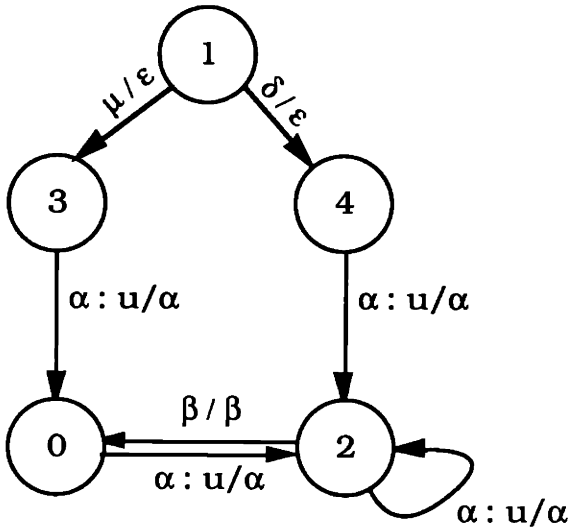

Figure 5.8: A Simple Example

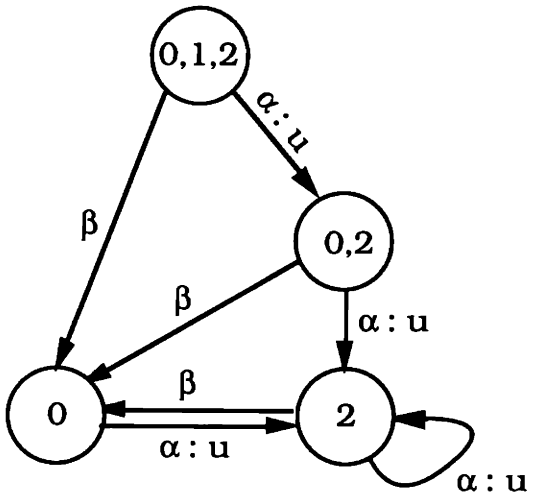

Figure 5.9: Observer for the system in Figure 5.8

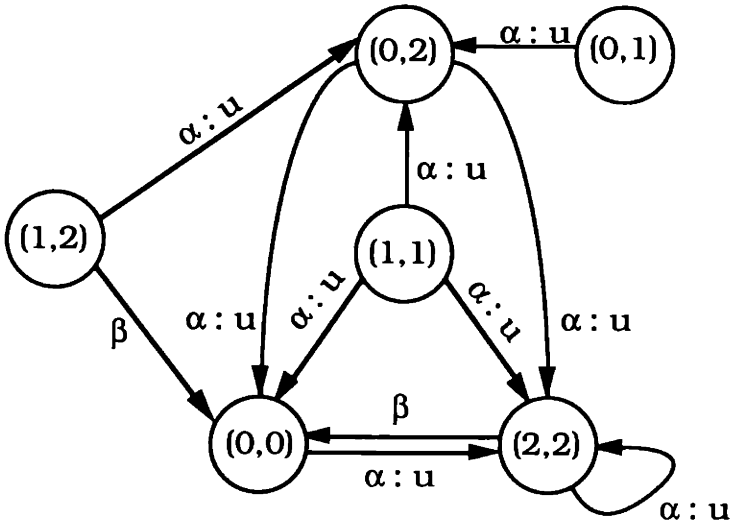

Figure 5.10: Example for the Automaton O_P

stabilizing feedback is one that disables α when the observer estimate is $\{2\}$.

To show that the computational complexity of testing Proposition 5.13 is polynomial in q, we proceed as we did in Chapter 4 for testing observability: Let $P = Y \times Y$ and construct the pair automaton O_P with state space P and event set Γ. For example, the corresponding automaton O_P for the system in Figure 5.8 is illustrated in Figure 5.10.

As developed in Chapter 4, although O_P is a nondeterministic automaton and therefore is certainly not an observer for A, O_P can be used to check the observability of A. Specifically, the dynamics of O_P have the following interpretation: Suppose that the system might be in either state x or state y, and suppose that the event γ occurs. Then, the next state of A' (the automaton defined over Y, see Section 4.2.1) could be any element of

$$S = f'(x, \gamma) \cup f'(y, \gamma) \tag{5.11}$$

The pair automaton dynamics captures this possible ambiguity by moving from (x, y) to any (x', y') with $x', y' \in S$. Also, there are some special states in O_P, namely those in $E_P = \{(x, x) | x \in Y\}$, corresponding to no ambiguity. It is not difficult to see that observability of A is then equivalent to the E_P-stability of O_P. Similarly, if a set of states of O_P of the form (x, x) is w_P-invariant, then the corresponding set of states in the observer is w-invariant. Thus, we can compute E_w using O_P: We first find V_P, the maximal w_P-invariant subset of E_P, which will be of the form $\{(x, x) | x \in Y'\}$ for some $Y' \subset Y$. It then follows that $E_w = \{\{x\} | x \in Y'\}$:

Proposition 5.14 E_w is the maximal w-invariant subset of the singleton states of O iff $\{(x, x) | \{x\} \in E_w\}$ is the maximal w_P-invariant subset of E_P in O_P.

Proof: Straightforward by assuming contrary in each direction. ❑

As an example, compare Figure 5.9 and Figure 5.10.

Furthermore, it follows from the work we did in Chapter 4 that O is E_w-stable iff O_P is $\{(x, x) | \{x\} \in E_w\}$-stable. Since testing a system for stability is equivalent to testing a system for pre-stability (see Chapter 3) which takes quadratic time in the number of states in the sytem, Proposition 5.13 can be tested in $O(q^4)$ time.

If the conditions of Proposition 5.13 are not satisfied, we can test a weaker sufficient condition for output stabilizability while keeping polynomial complexity. Instead of the maximal w-invariant subset of the singleton states, we can use a notion of achieving invariance using state feedback, that we have defined in Chapter 3: Given A and $Q \subset X$, Q is sustainably (f, u)-invariant in A if there exists a state feedback such that Q is alive and f-invariant in the closed loop system. In Chapter 3, we also show that a maximal sustainable (f, u)-invariant subset of a given set exists and we present an algorithm that computes it. Let E_u be the maximal sustainable (w, u)-invariant subset of the singleton states and let K_u be the associated state feed-

back. Note that K_u only needs to act on the singleton states, and thus it can also be thought of as a feedback for A. Note also that K_u needs to disable those events that take states in E_u outside of E_u, and it is unique provided that it only disables such events. As before, if A_{K_u} is $E \cap E_u$-stabilizable and O is E_u-stable, then A is output stabilizable:

Proposition 5.15 Suppose that (i) $E \cap E_u = \emptyset$; (ii) A is $E \cap E_u$-stabilizable; and (iii) O is E_u-stable. Then if $K_s(x)$ is a stabilizing feedback, the feedback

$$\hat{K}(\hat{x}) = \begin{cases} K_u(x) \cap K_s(x) & \text{if } \hat{x} = \{x\} \in E_u \\ \Phi & \text{otherwise} \end{cases} \tag{5.12}$$

is an output stabilizing feedback for A.

Proof: Straightforward. ❑

As an example, in Figure 5.11, where all events are observable, $E_w = \emptyset$, but $E_u = \{\{0\}, \{2\}\}$ and the associated feedback disables α when the observer is in state $\{0\}$. Furthermore, $E \cap E_u = \{0\}$ and if we disable α at state 2 then we can stabilize A with respect to state 0. Finally, note that O is E_u-stable. Thus, A is output stabilizable, and an output stabilizing feedback is one that disables α when the observer estimate is 0 or 2.

This sufficient condition can also be tested in polynomial time since, similar to Proposition 5.14, E_u is the maximal sustainable (w, u)-invariant subset of the singleton states of O iff $\{(x, x) | x \in E_u\}$ is the maximal sustainable (w_P, u)-invariant subset of E_P in O_P. Furthermore, O is E_u-stable iff O_P is $\{(x, x) | \{x\} \in E_u\}$-stable. Therefore, this sufficient condition for output stabilizability can also be tested in $O(q^4)$ time.

We conclude this section by presenting an even weaker sufficient condition that can also be tested in polynomial time. This condition is based on a notion of always

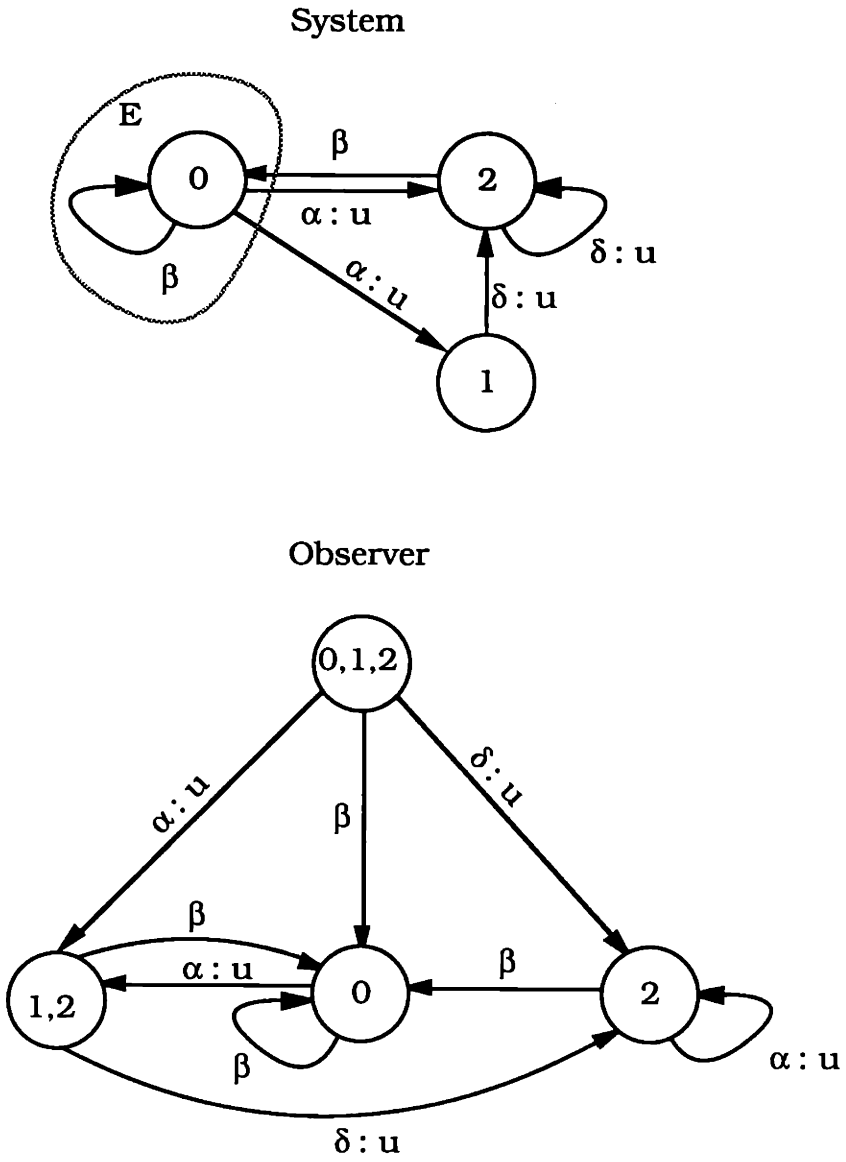

Figure 5.11: Simple Example for Using Control in Observer (all the events are observable)

observability that we defined in Chapter 4: We term a state x always observable if whenever the system is in x, the observer estimate is $\{x\}$. We term a system a-observable if it is stable with respect to its always observable states. Suppose that A is a-observable and let us construct the automaton A_a which is the same as A except that only events in always observable states can be controllable, i.e., $e_a(x) = d(x)$ for all states x that are not always observable. If A_a is stabilizable then A is also output stabilizable since whenever we need to exercise control, we have prefect knowledge of the state and thus we can simply use the feedback that stabilizes A_a on those singleton states of the observer that are always observable:

Proposition 5.16 Given an a-observable system A, if A_a is E-stabilizable then A is output stabilizable. ❑

As we show in Chapter 4, a-observability can be tested in $O(q^4)$ time, and thus this sufficient condition can also be tested in $O(q^4)$ time.

5.4 Resiliency

As we did with observability in Chapter 4, we can address a problem of robustness. Specifically, in this section we study the property of resilient output stabilizability in the sense that in spite of a burst of observation errors, the system stays alive and goes through E infinitely often.

In order to define what we mean by a resilient stabilizability, we use the function $\xi(s,t)$ to represent the discrepancy between two strings s and t as defined and used in Chapter 4.

Definition 5.17 Given a strongly output stabilizable A, A is resiliently, strongly output stabilizable if there exists a strongly output stabilizing compensator $C : \Gamma^* \to U$ and

an integer i such that for all strings s that can be generated by A_C, i.e.,

- $\forall x \in X$,
- $\forall s \in L_f(A_C, x)$,

for all possible ouput strings t which can be generated by corrupting $h(s)$ with a finite length burst, i.e.,

- $\forall$ positive integers i,
- $\forall t \in \Gamma^*$ such that $\xi(t, h(s)) \leq i$,

the compensator acting on such corrupted strings still strongly stabilizes the system after the error burst has ended. That is, for each such x, s, and t, the compensator $C'(h(s')) \triangleq C(th(s'))$, defined for $s' \in h(L(A, f(x,s)))$ is such that

- the range of $f(x,s)$ is alive in $A_{C'}$, i.e., for all $x \in R(A_{C'}, f(x,s))$, $d_{C'}(x) \neq \emptyset$, and
- for all $p \in L(A_{C'}, f(x,s))$ such that $|p| \geq i$, there exists a prefix p' of p such that $|p/p'| \leq i$ and $f(x, sp) \subset w_{CR}(\{Y\}, th(p')) \subset E$, where w_{CR} is the transition function of the resilient observer O_{CR} for A_C.

We say that C is a resiliently, strongly stabilizing compensator for A. ❑

In the above definition, the requirements on C' ensure that the compensator C acting on the corrupted output string (a) preserves liveness (as stated in the first bullet), and (b) stabilizes A following the burst (as stated in the second bullet).

Let us return to the characterization of strong output stabilizability in Proposition 5.5, but note that we can no longer use O as a basis for constructing a stabilizing compensator since the burst may be an arbitrary string in Γ^*. Therefore, as we did for

resilient observability in Chapter 4, we will use O_R. In particular, given the observer O and an observer feedback K, define $O_{KR} = (F_{KR}, w_{KR}, v_{KR})$ so that

$$w_{KR}(\hat{x}, \gamma) = \begin{cases} w_K(\hat{x}, \gamma) & \text{if } \gamma \in v_K(\hat{x}) \\ \{Y\} & \text{otherwise} \end{cases} \tag{5.13}$$

$$v_{KR}(\hat{x}) = \Gamma \tag{5.14}$$

We can then define a compensator $C(s) = K(w_{KR}(\{Y\}, s))$ for all $s \in \Gamma^*$. If an error burst now occurs, it may put the system and observer in arbitrary states not necessarily within the reach of the initial states X_I defined in Proposition 5.5. As the following result shows, we can characterize resilient output stabilizability as the stability of $A \parallel O_{KR}$ for some observer feedback K. In fact, since $A \parallel O_{KR} = A \parallel O_K$, we can use $A \parallel O_K$ instead:

Proposition 5.18 A is resiliently, strongly output stabilizable if there exists a state feedback $K : Z \to U$ for the observer such that $A \parallel O_K$ is E_{OC}-stable.

Proof: ($\rightarrow$) Straightforward by assuming the contrary.

($\leftarrow$) Straightforward since then $C(s) = K(w_{KR}(\{Y, \}, s))$ resiliently, strongly stabilizes A. ❑

Finally, we have the following companion of Proposition 5.4 which states that it is necessary and sufficient to test O for E_O-stability, but since the burst may put the system and the observer in arbitrary states, we need to use X-compatible feedback, in order to preserve liveness:

Proposition 5.19 A is resiliently, strongly output stabilizable with respect to E iff there exists a state feedback K for the observer such that O_K is E_O-stable and for all $\hat{x} \in Z$, $K(\hat{x})$ is X-compatible.

Proof: ($\leftarrow$) Straightforward.

($\rightarrow$) Assume contrary, then for each K such that O_K is E_O-stable, there exists some $\hat{x} \in Z$ and $x \in Y$ such that $(d(x) \cap K(\hat{x})) \cup e(x) = \emptyset$. Let s be a string such that $\hat{x} = w(\{Y\}, s)$. Suppose that the system started in state x and although no event has occurred, the observer observed a burst s. Then, while the system is still in x, the observer is in $\hat{x}$ and no other transition can occur. Therefore, A cannot be resiliently, strongly output stabilizable and we establish a contradiction. ❑

An algorithm for testing resilient, strong output stabilizability and constructing a feedback is identical to the algorithm in Proposition 5.6 except that when we search for a feedback, we search for one that is X-compatible, as opposed to $\hat{x}$-compatible, and the computational complexity is again $O(q^3|Z|)$. Thus, if we can find K that satisfies Proposition 5.19, then $C(s) = K(w_{KR}(\{Y,\}, s))$ is a resiliently, strongly stabilizing compensator for A.

We define resilient output stabilizability similarly:

Definition 5.20 Given output stabilizable A, A is resiliently output stabilizable if there exists an output stabilizing compensator C such that for all strings s that can be generated by A_C, i.e.,

- $\forall x \in X$,
- $\forall s \in L_f(A_C, x)$,

for all possible ouput strings t which can be generated by corrupting $h(s)$ with a finite length burst, i.e.,

- $\forall$ positive integers i,
- $\forall t \in \Gamma^*$ such that $\xi(t, h(s)) \leq i$,

the trajectories starting from $f(x,s)$ visit E infinitely often, i.e., $f(x,s)$ is E-stable in $A_{C'}$, where

$$C'(h(s')) = C(th(s'))$$

for all $s' \in h(L(A, f(x,s)))$. We say that C is a resiliently stabilizing compensator for A. ❑

The following result immediately follows from this definition:

Lemma 5.21 Given resiliently output stabilizable A, if some C is a resiliently stabilizing compensator for A then $C(s)$ is X-compatible for all $s \in h(\overline{L}(A))$. ❑

Similar to resilient strong output stabilizability, necessary and sufficient conditions for resilient output stabilizability parallel those of output stabilizability except that we need to use X-compatible feedback. Since, a resilient output stabilizing compensator needs to be defined for all strings in Γ^*, given a feedback K for the automaton Q defined in Section 5.2.2, we define $Q_{KR} = (G_{KR}, w_{KR}, v_{KR})$ so that

$$w_{KR}(y,\gamma) = \begin{cases} w_{Q_K}(y,\gamma) & \text{if } \gamma \in v_{Q_K}(y) \\ (Y,Y) & \text{otherwise} \end{cases} \tag{5.15}$$

$$v_{KR}(y) = \Gamma \tag{5.16}$$

We can then define a compensator $C(s) = K(w_{KR}((Y,Y),s))$ for all $s \in \Gamma^*$. We state the following companion of Proposition 5.10 where

$$E_{QR} = \{y = (y_1, y_2) \in W | \exists F \subset \Phi \text{ such that } v_{QF}(y) = \emptyset \text{ and } F \text{ is } X\text{-compatible}\} \tag{5.17}$$

Proposition 5.22 A is resiliently ouput stabilizable iff there exists a state feedback K such that Q_K is E_Q-pre-stable and for all $y \in W$, $K(y)$ is X-compatible in A.

Furthermore, the compensator defined by

$$C(s) = K(w_{KR}((Y,Y),s))$$

for all $s \in \Gamma^*$ resiliently stabilizes A.

Proof: ($\rightarrow$) Clearly, a feedback K which pre-stabilizes Q exists. By Lemma 5.21, the second condition is satisfied.

($\leftarrow$) Straightforward ❑

An algorithm for testing resilient output stabilizability and constructing a feedback can be generated from the algorithm in Proposition 5.6 in a straightforward fashion. In particular, we use E_{QR} in place of E_Q in Proposition 5.6 and we check X-compatibility, instead of y_2-compatibility.

For example, the feedback we computed for Q in order to stabilize the system in Figure 5.4 is also X-compatible (see Figure 5.6(b)), since, in this case, disabling either, but only one of, α or β does not disable all the events in any state of the system. A resilient output stabilizing compensator for the system in Figure 5.4 is illustrated in Figure 5.12 for which the initial state is (123,123).

5.5 Discussion

In this chapter, we have introduced notions of output stabilizability and resiliency for discrete-event systems described by finite-state automata, and we have developed algorithms to test for output stabilizability, resiliency, and to construct resilient output stabilizing compensators. These algorithms are polynomial in the cardinality of the state space of the observer. We have also presented sufficient conditions which can be tested in polynomial time in the cardinality of the state space of the system.

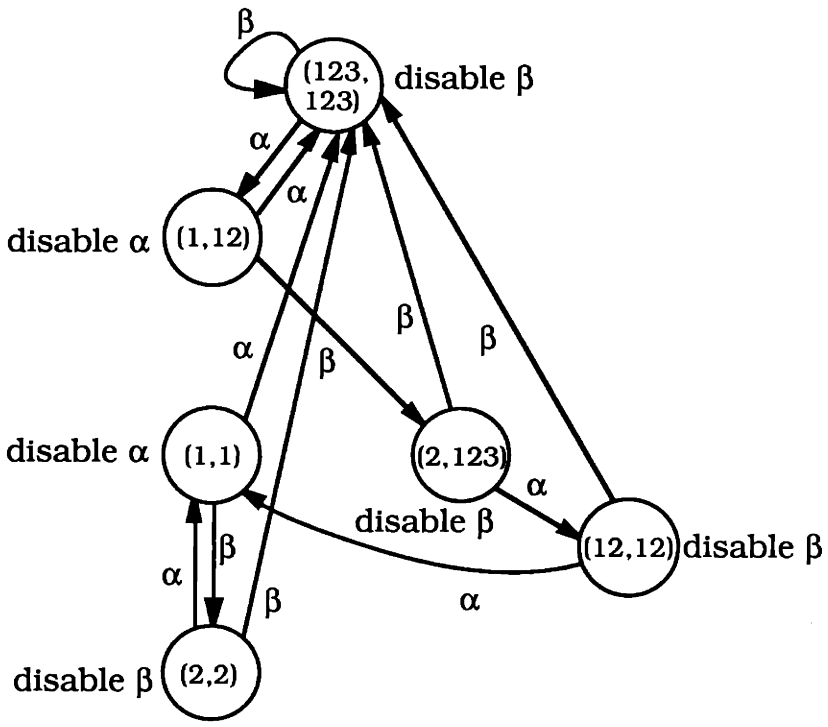

Figure 5.12: Resilient Output Stabilizing Compensator for Figure 5.4

The results presented in this chapter provide us with methods for stabilizing DEDS and for ensuring robustness to observation errors so that catastrophic error propagation is avoided. They also provide the basis for our work in controlling a DEDS so that particular sets of desired strings are tracked. In Chapter 7, we address this problem and formulate it as the stabilization of the composite of A and an automaton which generates the string or the set of strings that we wish the system to track. Using the results in this chapter, we can, in a straightforward way, also address tracking problems in the case of partial observations and observation errors.

Chapter 6
Invertibility

6.1 Motivation and Background

The focus of attention in this chapter is the system-theoretic problem of invertibility, i.e., the problem of reconstructing the full event sequence given observations only of output events. Such a problem may arise in the monitoring of a complex system or in post-mortem analysis or troubleshooting. Also, the dual of this problem, the generation of input sequences to achieve specified output sequences is of considerable importance in characterizing the tracking and regulation capabilities of a DEDS (see Chapter 7). In addition, as we will see, the error behavior of an inverter for a DEDS can be quite non-resilient. In particular, much as in catastrophic error propagation in sequential decoding [36], some DEDS inversion problems have the undesirable property that a finite burst of observation errors can lead to an unbounded sequence of inversion errors. We hope that our work here will help in the development of a theory and methodology for characterizing when large-scale DEDS can exhibit such behavior and for designing compensation that provides enhanced robustness to errors.

In this chapter, we concentrate on the uncontrolled system with partial knowledge of the event trajectory, as we did in Chapter 4. Specifically, the systems we consider in this chapter are of the form $A = (G, f, d, h)$ where $G = (X, \Sigma, \Gamma)$.

Finally, recall from Chapter 4 that I_M denotes the maximal set of indistinguish-

able pairs. In addition to the results on indistinguishability in Chapter 4, we state the following which plays an important role in the development of this chapter:

Proposition 6.1 Given $x, y \in Y$, $(x, y) \in I_M$ iff x and y share an output string of length greater than or equal to q^2.

Proof: Straightforward since any path of length q^2 in O_P has a cycle embedded in it. ❑

For future reference, let n_i denote the minimum number of observations required to distinguish between any pair of distinguishable states in A. What Proposition 6.1 states is that $n_i \leq q^2$. Of course, in many systems n_i can be much smaller than this bound.

6.2 Invertibility

In this section, we present and analyze two notions of invertibility: The first notion assumes that the initial state is known, while the second notion does not assume that the initial state is known. Since a reconstruction in the latter case involves estimating the current state first, our second notion allows for a bounded error in the beginning of the reconstructed string.

6.2.1 Invertibility with a Delay

We consider first the problem in which the initial state x_0 of A is known. We assume that A is a minimal automaton generating the event language $L = L(A, x_0)$, so that all states are reachable from x_0, and no two states generate the same language. Furthermore, we assume that A is deterministic. Neither of these assumptions is restrictive since we will be concerned with the estimation of elements in L, and we can

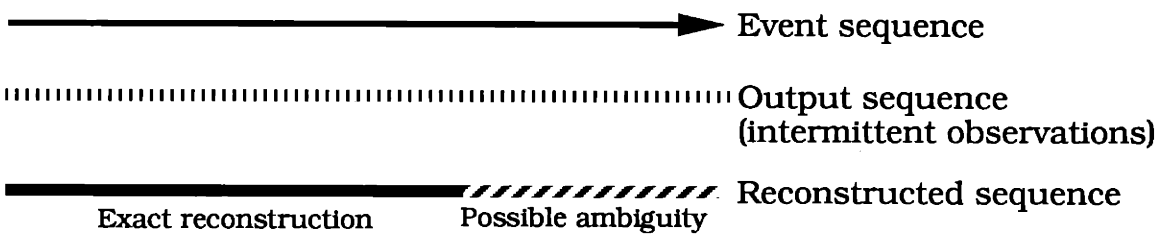

Figure 6.1: Invertibility with a Delay: Given the output sequence, the event sequence is reconstructed exactly but with some delay. The ambiguity at the end of the reconstructed string will be resolved using future observations.

always choose a minimal deterministic automaton (and initial state) that generates L. Specifically, given L, or equivalently, such an A and x_0, we are interested in whether or not we can reconstruct an event trajectory $s \in L$ when we only observe that part of the string that is in Γ, i.e., we observe $h(s)$ (see Figure 6.1).

To begin, let h_L be the restriction of the output function h to the domain L so that $h_L^{-1}(r)$ is the set of strings in L that generate the output string $r \in h(L)$. Let us now formally define invertibility with a delay:

Definition 6.2 L is invertible with a delay (or WD-invertible) if there exists an integer n_d such that $\forall s \in L$, there exists a prefix p of s such that $h_L^{-1}(h(s)) \subset p(\Sigma \cup \{\epsilon\})^{n_d}$, where $p(\Sigma \cup \{\epsilon\})^{n_d}$ is used to denote the set: p concatenated with arbitrary strings of length at most n_d. ❑

What this definition states is that for a WD-invertible language, we can, at any time, use knowledge of the output sequence up to that time to reconstruct the full event sequence up to a point at most n_d events into the past (that is, p is uniquely specified).

Consider the system in Figure 6.2, where state 0 is the initial state. In this case, L is WD-invertible with $n_d = 4$. Note that it is not invertible without delay (i.e., $n_d = 0$). For example, if we observe σ^2, the original input string could be $\alpha(\delta\sigma)^2$ or

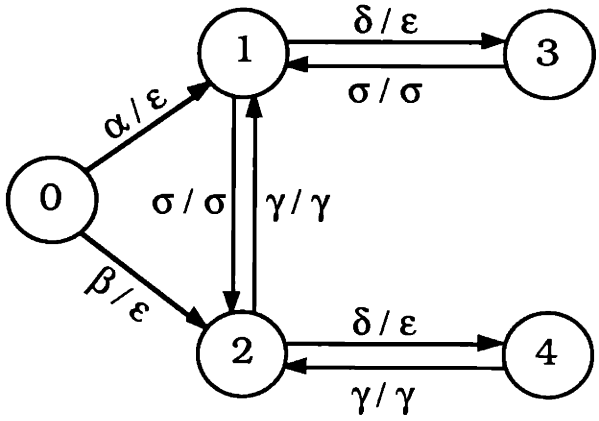

Figure 6.2: Example for WD-Invertibility: State 0 is the initial state.

$\alpha(\delta\sigma)^2\delta$ or $\alpha\delta\sigma\sigma$, etc., but we know the first three events with certainty.

To begin our investigation of WD-invertibility, recall from Chapter 4 that $L_f(A,x)$ consists of all strings in $L(A,x)$ with observable events as their last events. Let us define two subsets of $L_f(A,x)$. Specifically, let $L_1(A,x)$ (or L_1 where it is clear from the context) consist of those strings of $L_f(A,x)$ that have only one observable event, and $L_\sigma(A,x)$ (or L_σ) be the set of strings in L_1 that have $\sigma \in \Gamma$ as the observable event. We first need the following notion:

Definition 6.3 A is termed ambiguous if for some $x \in X$ and $\gamma \in \Gamma$, there exist distinct $s,t \in L_\gamma(A,x)$ such that $f(x,s) = f(x,t)$. ❑

The importance of this concept for invertibility is given by the following:

Proposition 6.4 If A is ambiguous, then L is not WD-invertible.

Proof: Let x, γ, s, and t be as in Definition 6.3. Since A is minimal and deterministic, find an event sequence p so that $f(x_0,p) = x$. Then, the distinct sequences ps and pt have identical outputs and drive x_0 to the same state. Thus, no future behavior will

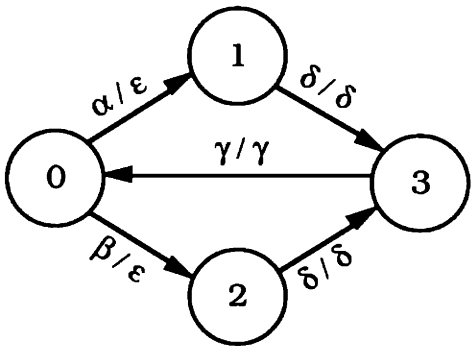

Figure 6.3: Example for an Ambiguous System

allow us to distinguish between these strings. ❑

As an example, the system in Figure 6.3 is ambiguous since both $\alpha\delta$ and $\beta\delta$, which produce the same output, take state 0 to state 3. Thus the language generated from 0 is not invertible.

Unambiguity alone is not sufficient for invertibility. For example, the automaton in Figure 6.4, where 0 is the initial state, is not ambiguous, but $L(A, x_0)$ is not invertible either. More specifically, event trajectories $(\beta\alpha)^*$ and $(\delta\alpha)^*$ both have the output α^*. In order to explore necessary and sufficient conditions for invertibility, let us first state the following recursive characterization of invertibility which follows from the fact that $R(A, x_0) = X$:

Proposition 6.5 L is WD-invertible iff $L(A, x)$ is WD-invertible for each $x \in X$. ❑

What this result suggests is the following: Suppose that we have perfect knowledge of some state x in the past, and that we have an algorithm for reconstructing the event trajectory s that corresponds to the next observed event γ from x, i.e., $s \in L(A, x)$ and $h(s) = \gamma$. Then, thanks to determinism, we also have perfect knowledge of the state that s takes x to. Since we also have perfect knowledge of the initial state,

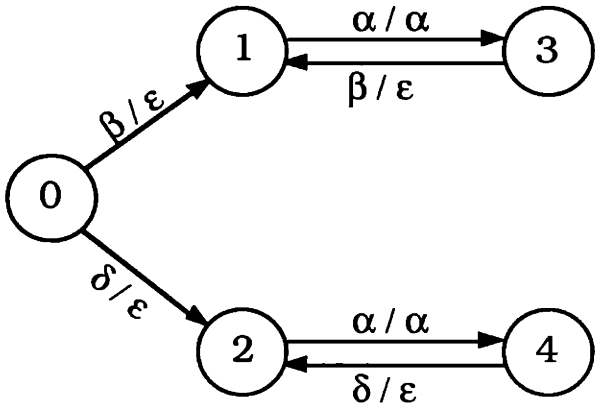

Figure 6.4: Example for an Unambiguous but not Invertible System: State 0 is the initial state.

we can reconstruct the entire event trajectory by applying this algorithm iteratively. We use such an approach below to present necessary and sufficient conditions for WD-invertibility which can be tested in polynomial time.

Consider an unambiguous A, any $x \in X$, and any $\sigma \in \Gamma$. Then, distinct elements of $L_\sigma(A, x)$ correspond to distinct elements of

$$Q_{x,\sigma} = f(x, L_\sigma(A, x)) \tag{6.1}$$

If L is WD-invertible then at some point we will be able to reconstruct the entire event sequence and hence the entire state trajectory through the transition into $Q_{x,\sigma}$ caused by the last observable event σ. That is, with the help of future event observations, we are able to distinguish between the elements of $Q_{x,\sigma}$. For example, in Figure 6.2, A is not ambiguous,

$$Q_{0,\sigma} = Q_{0,\gamma} = Q_{1,\sigma} = Q_{2,\gamma} = \{1, 2\}$$

and (1,2) is not an indistinguishable pair since state 1 produces only σ as output and state 2 produces only γ. Thus, we can distinguish between the states 1 and 2 using

the next observed event. Consequently, we have the following:

Proposition 6.6 L is WD-invertible iff A is not ambiguous, and for all $x \in Y \cup x_0$, $\sigma \in \Gamma$ and distinct $y, z \in Q_{x,\sigma}$, $(y, z) \notin I_M$. Furthermore, if L is WD-invertible then $n_d \leq nq^2$.

Proof: ($\rightarrow$) From Proposition 6.4, we only need to check the second condition. Assume the contrary, then for some $x \in Y$ and $\sigma \in \Gamma$, there exists distinct $s, t \in L_\sigma$ such that $(f(x, s), f(x, t)) \in I_M$. Thus, using the future outputs, we cannot distinguish between s and t. Therefore, L cannot be WD-invertible and we establish a contradiction.

($\leftarrow$) We will prove this inductively and this proof will also serve as a construction for an inverter. Clearly, in the beginning, we know that the system is in state x_0. Suppose that at some point in time, we have inverted the output sequence up through some point and therefore know that the system was in some particular state x a finite number of transitions into the past. Suppose that γ is the first observable event after the system is in x. Since all pairs in $Q_{x,\gamma}$ are distinguishable, we can, using at most q^2 future outputs (corresponding to nq^2 events since any chain of unobservable events can have at most n events), determine which state $y \in Q_{x,\gamma}$ the system was in. Given x, y, γ and since A is not ambiguous, we can exactly reconstruct that part of the input string which takes x to y and produces γ as the output. Therefore, the entire input sequence which takes x_0 to y can be reconstructed exactly, and by induction we have proven invertibility. ❑

To summarize, our WD-inverter does the following:

- Start with the initial state x_0.
- Given the next output γ, use future outputs (at most q^2 of them will be necessary and this corresponds to at most nq^2 events) to distinguish between the states

in $Q_{x_0,\sigma}$ (let y be the new state).

- Reconstruct that part of the state trajectory between x_0 and y.
- Repeat the process using the state y as the initial state and the output after y.

We conclude this section by providing a polynomial test for WD-invertibility. First of all, to construct a test for ambiguity, note that we only need to consider the states in Y, in addition to the initial state x_0, since all other states can be reached by states in $Y \cup \{x_0\}$ using only unobservable events. Furthermore, if there are no transitions defined to x_0, then $x_0 \in Y$, and otherwise x_0 can be reached by some state in Y using only unobservable events. Therefore, we only need to consider the states in Y. Let us pick some $x \in Y$ and consider the set of states $X_x = R(A|\overline{\Gamma}, x)$ which includes x itself. Recall that by assumption, there can be no loops that consist of only unobservable events. As the next result shows, ambiguity may then arise in three forms:

1. Two unobservable events define the same transition from some y to some z both in X_x.
2. For some distinct y and z in X_x there is an observable event defined at both y and z that takes both y and z to the same state (This is the case in Figure 6.3).
3. For some distinct y and z in X_x, there are unobservable events defined from y and z that take y and z to the same state.

Specifically, we have the following:

Proposition 6.7 *A is ambiguous iff there exists some $x \in Y$, such that any of the following conditions is satisfied*

1. There exist some $y \in X_x$, and $\sigma_1, \sigma_2 \in d(y) \cap \overline{\Gamma}$ such that $f(y, \sigma_1) = f(y, \sigma_2)$.
2. There exist distinct $y, z \in X_x$, and $\gamma \in d(y) \cap d(z) \cap \Gamma$ such that $f(y, \gamma) = f(z, \gamma)$.
3. There exist distinct $y, z \in X_x$, $\sigma_1 \in d(y) \cap \overline{\Gamma}, and \sigma_2 \in d(z) \cap \overline{\Gamma}$ such that $f(y, \sigma_1) = f(z, \sigma_2)$.

Proof: ($\leftarrow$) Obvious.

($\rightarrow$) Ambiguity implies that there exist some $w \in X$, $\gamma \in \Gamma$, and distinct $r, q \in L_\gamma(A, w)$ such that $f(w, r) = f(w, q)$. Since w can be reached by some $x \in Y$ using only unobservable events (i.e., $w \in X_x$), there exist distinct $s, t \in L_\gamma(A, x)$ such that $f(x, s) = f(x, t)$. Let $s = s'\gamma$ and $t = t'\gamma$. If $f(x, s') \neq f(x, t')$ then the second condition is satisfied. Suppose that $f(x, s') = f(x, t')$. Furthermore, suppose that there exist prefixes s'' of s' and t'' of t' such that $f(x, s'') \neq f(x, t'')$ but $f(x, s''\sigma_1) = f(x, t''\sigma_2)$ where σ_1 (respectively, σ_2) is the next event in s' after s'' (respectively, the next event in t' after t''). Then, the third condition must be satisfied. Finally, if we cannot find such s'' and t'', then the state trajectories corresponding to s and t must be the same. Since s and t are distinct they must differ in at least one event, and thus there must exist some state y in the state trajectory so that the first condition is satisfied. ❑

In order to simplify this test further let us pick some $x \in Y$. We first consider the second condition of the above proposition. For each $y \in f(X_x, \Gamma)$ and $\gamma \in d(X_x)$ let us define the following set:

$$F_{y,\gamma}^{-1} = \{z \in X_x | \gamma \in d(z) \text{ and } f(z, \gamma) = y\} \tag{6.2}$$

Note that this set can be empty for some y and γ. It is obvious that the second condition is satisfied iff there exists some y and γ such that $|F_{y,\gamma}^{-1}| \geq 2$. We next

consider a combined test for the first and third conditions. In this case, for each $y \in X_x$ let us define the following set:

$$D_y^{-1} = \{\sigma \in \overline{\Gamma} | \exists z \in X_x \text{ such that } \sigma \in d(z) \text{ and } f(z,\sigma) = y\} \tag{6.3}$$

It is straightforward to show that either the first or the third condition is satisfied iff $|D_y^{-1}| \geq 2$ for some y. We thus have the following result:

Corollary 6.8 *A is not ambiguous iff for all $x \in Y$,*

1. *for all $y \in f(X_x, \Gamma)$ and $\gamma \in d(X_x)$, $|F_{y,\gamma}^{-1}| \leq 1$, and*
2. *for all $y \in X_x$, $|D_y^{-1}| \leq 1$.* ❑

Let $\overline{n}_x$ be the maximum of $|R(A|\bar{\Gamma}, x)|$ over all $x \in Y$. Then, testing for ambiguity takes $O(\overline{n}_x q)$ time.

The following result shows that a necessary and sufficient test for the second condition of Proposition 6.6 is to test if (x_0, x_0) can only reach indistinguishable pairs in O_P:

Proposition 6.9 *Given A, we have that $(y,z) \notin I_M$ for all $x \in Y$, $\sigma \in \Gamma$ and distinct $y, z \in Q_{x,\sigma}$ iff the range of (x_0, x_0) in O_P is contained in $\{(x,y) \notin I_M\}$.*

Proof: Straightforward by assuming contrary. ❑

The condition of the above proposition can be tested in $O(q^2)$ time. Therefore, WD-invertibility can be tested in $O(\overline{n}_x q + q^2)$ time.

6.2.2 Invertibility with Unknown Initial State

Let us now consider a related notion of invertibility which will be of particular importance when we consider inversion in the presence of observation errors. In particular,

suppose that we do not have any information concerning the initial state. In this case, in general, we will not be able to reconstruct the entire event string because of some unresolvable ambiguity at the start. However, it may be possible for us to perform error-free reconstruction after the initial period of uncertainty. In this section, we investigate this property.

Let $h_{L(A)}$ be the restriction of the output function h to the domain $L(A)$ so that $h^{-1}_{L(A)}(r)$ is the set of strings in $L(A)$ that generate the output string $r \in h(L(A))$:

Definition 6.10 A system A is invertible with a delay with unknown initial state (WDX-invertible) if there exists an integer n_d such that for all $x \in X$, $s \in L(A,x)$, there exist p,q,r such that $s = pqr$, and $h^{-1}_{L(A)}(h(s)) \subset (\Sigma \cup \{\epsilon\})^{n_d} q (\Sigma \cup \{\epsilon\})^{n_d}$. ❑

In order to derive necessary and sufficient conditions for WDX-invertibility, we first enrich the state space of A by including the event trajectory as part of the state. Thanks to our assumption that there are no loops of unobservable events, the number of strings in $L_1(A,x)$ is finite for any x. Thus, we can enrich the state space of A to include such event trajectories, while keeping the state space finite, by using strings in L_1. In particular, let the new state space be as follows:

$$X_s = \{(s,y) | s = \epsilon \text{ or } s \in L_1(A,x) \text{ for some } x \in Y \text{ such that } y = f(x,s)\} \quad (6.4)$$

We then define a system $A_s = (G_s, f_s, d_s, 1)$, where $G_s = (X_s, \Gamma, \Gamma)$, and

$$f_s((s,y),\gamma) = \{(t,z) | t \in L_\gamma(A,y) \text{ such that } z = f(y,t)\} \quad (6.5)$$

$$d_s((s,y)) = \{\gamma \in \Gamma | \exists t \in L_1(A,y) \text{ such that } h(t) = \gamma\} \quad (6.6)$$

Note that A_s is not necessarily deterministic. For example, A_s corresponding to Figure 6.2 is illustrated in Figure 6.5. Note that for each state x, A_s has a state (ϵ, x). Since, for example, the string $\alpha\delta\sigma$ takes state 0 to state 1 in Figure 6.2, the

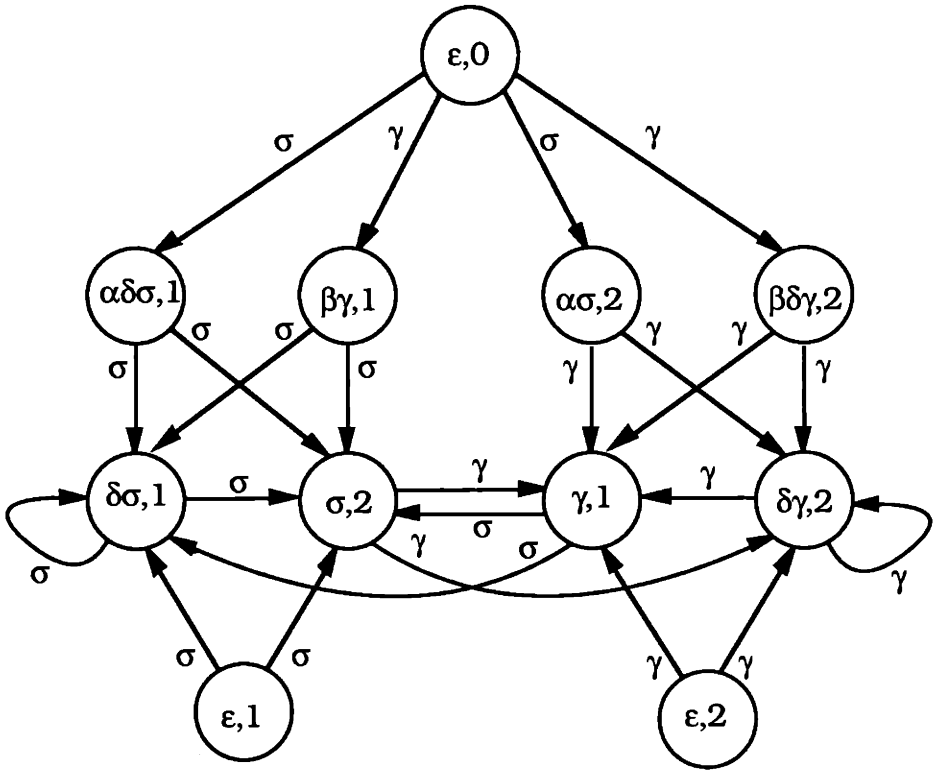

Figure 6.5: A_s for Figure 6.2

event σ takes state $(\epsilon, 0)$ to state $(\alpha\delta\sigma, 1)$ in Figure 6.5. Similarly, the event σ also takes $(\epsilon, 0)$ to $(\alpha\sigma, 2)$.

WDX-invertibility can be characterized more easily using A_s. First of all, note that reconstructing the event trajectory in A corresponds to reconstructing the state trajectory in A_s, but only with respect to the first component of the states, i.e., if two states in X_s have their first components equal, it is not important, for WDX-invertibility, to distinguish between these two states. However, if two states in X_s differ in their first components, then we need to be able to tell which state the trajectory has passed through in order to reconstruct the event trajectory in A, and thus, the second components of these two states need to be distinguishable in A. Let $O_s = (H_s, w_s, v_s)$ where $H_s = (Z_s, \Gamma, \Gamma)$ denotes the observer for A_s. We will characterize WDX-invertibility based on this observer. In particular, let E_s be those states $\hat{x}$ of O_s, such that for all pairs of states $(s, x), (t, y) \in \hat{x}$, either $s = t$ or x and y are distinguishable in A, i.e.,

$$E_s = \{\hat{x} \in Z_s | \text{ for all } (s_1, y_1), (s_2, y_2) \in \hat{x}, s_1 = s_2, \text{ or } (y_1, y_2) \notin I_M\} \tag{6.7}$$

In order to be able to reconstruct the event trajectory of A within a finite number of transitions, we need the observer to be E_s-pre-stable and indeed we would like the observer to stay in E_s. In the following result, let W_s be the maximal w_s-invariant subset of E_s in O_s:

Proposition 6.11 *A is WDX-invertible iff the observer O_s for A_s is W_s-pre-stable.*

Proof: ($\rightarrow$) Straightforward by assuming the contrary.

($\leftarrow$) We use the observer O_s as a basis for the inversion. Thanks to stability, the trajectory enters W_s in a finite number of transitions (Proposition 6.14 below provides a bound for the number of transitions it takes for the trajectory to enter O_s). The

trajectory stays in W_s once it enters W_s and we can then invert the event trajectory as follows: Let $\hat{x} \in W_s$ and partition $\hat{x}$ as

$$\hat{x} = \{(s_1, y_{11}), (s_1, y_{12}), \ldots\} \cup \{(s_2, y_{21}), (s_2, y_{22}), \ldots\} \cup \cdots$$

Note that $h(s_1) = h(s_2) = \cdots = \gamma$ where γ is the last observed event before O_s entered $\hat{x}$. In order to invert that portion of the event trajectory corresponding to the observation γ, we need to be able to distinguish between y_{ij} and y_{kl} for $i \neq k$. Since by definition of E_s, $(y_{ij}, y_{kl}) \notin I_M$, for $i \neq l$, A is WDX-invertible. ❑

The WDX-inverter motivated by the above proof can be outlined as follows:

- Trace the output trajectory of A in O_s until the trajectory in O_s enters W_s.
- Let $\hat{x}_0$ be the state that the trajectory in O_s enters when it enters W_s for the first time and let γ be the last observed event. Partition $\hat{x}_0$ as

  $$\hat{x}_0 = \{(s_1, y_{11}), (s_1, y_{12}), \ldots\} \cup \{(s_2, y_{21}), (s_2, y_{22}), \ldots\} \cup \cdots$$

  Let $Y_i = \{y_{i1}, y_{i2} \ldots\}$ for each i. Thanks to distinguishability, the states in Y_i and Y_j do not share any infinite length output sequences for $i \neq j$. Thus, using future observations, distinguish between Y_i, or equivalently, decide which s_i has occurred. Then, that s_i is the actual trajectory in A corresponding to the observation γ.
- Repeat the previous step for all the subsequent states of the trajectory in O_s, following $\hat{x}_0$.

To test WDX-invertibility, we use the pair automaton $O_{sP} = (Z_{sP}, w_{sP}, v_{sP})$ associated with A_s, in a manner similar to that used in Chapter 4 for testing observability. In what follows, let

$$E_{sP} = \{((s_1, y_1), (s_2, y_2)) \in P_s | s_1 = s_2 \text{ or } (y_1, y_2) \notin I_M\} \tag{6.8}$$

and let W_{sP} denote the maximal w_{sP}-invariant set in E_{sP}:

Proposition 6.12 *A is WDX-invertible iff O_{sP} is W_{sP}-stable.*

Proof: Follows from Proposition 6.11, and Proposition 4.17. ❑

Let $\bar{l}_x$ be the maximum of $|L_1(A,x)|$ over all $x \in Y$, then, WDX-invertibility can be tested in $O((\bar{l}_x q)^4)$ time since O_{sP} has at most $(\bar{l}_x q)^2$ states.

Let us now proceed with calculating a bound for n_d. Note that n_d is due to the ambiguity at the beginning and at the end of the inverted string. The ambiguity at the end depends on n_i, the minimum number of transitions it takes to distinguish between two states in A, and the ambiguity at the beginning depends on the number of transitions it takes a trajectory in O_s to enter W_s. Specifically, if we let n_w denote the length of the longest trajectory in O_s that starts from the initial state of O_s and ends as soon as the trajectory enters W_s, and if we let n_u denote the length of the longest chain of unobservable events in A,[1] we have the following:

Proposition 6.13 *If A is WDX-invertible, then $n_d \leq n_u \max(n_i, n_w)$.*

Proof: Straightforward by construction of the WDX-inverter and the fact that A can have at most n_u unobservable events between observable events. ❑

Recall that a bound on n_i, as stated in Proposition 6.1, is q^2, and we show below that the same is also a bound on n_w:

Proposition 6.14 *If A is WDX-invertible, then $n_w \leq q^2$.*

Proof: Assume contrary, then there exists a path of at least q^2+1 states (corresponding to q^2 transitions) in O_s:

$$\hat{x}_0, \ldots, \hat{x}_{q^2}$$

[1] Note that $n_u \leq n - q$.

such that there exists a path in O_{sP}:

$$p_1 = ((s_{01}, y_{01}), (s_{02}, y_{02})), \ldots, p_{q^2} = ((s_{q^2 1}, y_{q^2 1}), (s_{q^2 2}, y_{q^2 2}))$$

for which $(s_{i1}, y_{i1}), (s_{i2}, y_{i2}) \in \hat{x}_i$ for all i and such that $p_{q^2} \notin W_{sP}$. Then, for some integers j and k, say $j < k$, $\{y_{1j}, y_{2j}\} = \{y_{1k}, y_{2k}\}$. Let $\sigma \in v_{sP}(p_j)$ such that $p_{j+1} \in w_{sP}(p_j, \sigma)$. But then, $p_{j+1} \in w_{sP}(p_k, \sigma)$. Thus, there exists a cycle

$$p_{j+1}, \cdots, p_k, p_{j+1}$$

in O_{sP} which may reach p_{q^2}. Since $p_{q^2} \notin W_{sP}$ and W_{sP} is w_{sP}-invariant, no state in the above cycle can be in W_{sP} either. Thus, O_{sP} cannot be W_{sP}-stable, and we establish a contradiction. Therefore, $n_w \leq q^2$. ❑

We conclude this section by presenting a result on distinguishability, that plays an important role in the development of resilient WDX-invertibility in the following section. Before presenting this result, we need to introduce the following notion: We term a state recurrent if it can be reached from another state by an arbitrarily long string. We let A_r denote the recurrent part of A, which we construct as follows: Let D_0 denote the set of "dead" states in A^{-1}, i.e.,

$$D_0 = \{x \in X | d^{-1}(x) = \emptyset\} \tag{6.9}$$

Now let D_1 be the set of states that can only reach D_0, with at most one transition, in A^{-1}, i.e.,

$$D_1 = D_0 \cup \{x \in X | f^{-1}(x, d^{-1}(x)) \subset D_0\} \tag{6.10}$$

In general, we let

$$D_{i+1} = D_i \cup \{x \in X | f^{-1}(x, d^{-1}(x)) \subset D_i\} \tag{6.11}$$

and we let n_r be the smallest integer so that $D_{n_r+1} = D_{n_r}$. Then it is not difficult to check that $\overline{D}_{n_r}$ is the set of all recurrent states of A. We define A_r over the set $X_r = \overline{D}_{n_r}$ but with the same dynamics as A, and let A_{sr}, with state space X_{sr}, be the counterpart for A_r of A_s. The following result states a connection between WDX-invertibility and distinguishability in A_r^{-1}:

Proposition 6.15 If A is WDX-invertible then for all $p_1 = (s_1, x_1), p_2 = (s_2, x_2) \in X_{sr}$ such that $(p_1, p_2) \notin E_{sP}$ and $h(s_1) = h(s_2)$, and for all $y_1, y_2 \in X_r$ such that $x_1 = f(y_1, s_1)$ and $x_2 = f(y_2, s_2)$, y_1 and y_2 must be distinguishable in A_r^{-1}.

Proof: Let $\gamma = h(s_1) = h(s_2)$, and let us assume the contrary. Thus, there exists some p_1, p_2, y_1, y_2 that satisfy above conditions and such that y_1 and y_2 are indistinguishable in A_r^{-1}. Let O_{rP}^{-1} denote the pair automaton corresponding to A_r^{-1}. Since y_1 and y_2 are indistinguishable in A_r^{-1}, then, using similar reasoning to that in Proposition 6.1, we conclude that there exist a cycle

$$z_1, \ldots, z_k, z_1$$

in O_{rP}^{-1} that is reachable from (y_1, y_2). But then, the same cycle also exists (in reverse order) in O_P so that it may reach (y_1, y_2) in O_P. This in turn implies that if we represent z_i by (z_{i1}, z_{i2}), then there exists a cycle

$$((t_{11}, z_{11}), (t_{12}, z_{12})), \ldots, ((t_{11}, z_{11}), (t_{12}, z_{12}))$$

in O_{sP}, for some t_{ij}, which may reach $((r_1, y_1), (r_2, y_2))$, for some r_1 and r_2, in O_{sP}. But then, this cycle may also reach (p_1, p_2) which is <u>not</u> in E_{sP}. Therefore, none of the elements of the above cycle can be in W_{sP} (since it is invariant) and we establish a contradiction since A cannot be WDX-invertible. ❑

Finally, we let n_{ii} denote the minimum number of transitions it takes to distinguish between any two distinguishable states in A_r^{-1} and note that $n_{ii} \leq q^2$.

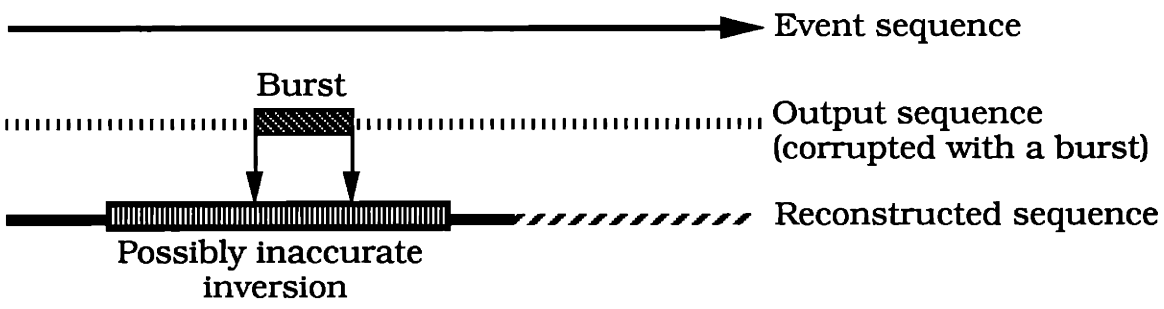

Figure 6.6: Resilient WD-invertibility: Inversion can only be wrong for a finite number of transitions before and after the burst.

6.3 Resilient Inverters

As with the observability problem, we are interested in resilient inverters. Specifically, we wish to construct inverters that invert correctly after a finite number of transitions following a burst. In addition, in contrast to the situation for resilient observability, we will allow the reconstructed string to be incorrect in a bounded window before and after the burst. We represent this notion of invertibility pictorially by Figure 6.6.

In contrast to the situation for observability and resilient observability, invertibility is not sufficent for resilient invertibility. For example, consider Figure 6.7 where 0 is the initial state. There is no resilient inverter for this system since an erroneous insertion of β, say after the kth output, may lead to an infinite number of errors. If in fact β never occurs, the actual string would be $(\alpha\delta)^*$ whereas the inverted string is $(\alpha\delta)^k\beta(\alpha\gamma)^*$.

In order to define what we mean by resilient invertibility, we use the function $\xi(s,t)$ to represent the discrepancy between two strings s and t as defined and used in Chapter 4. Also, as in the case of resilient observability, we allow the burst to

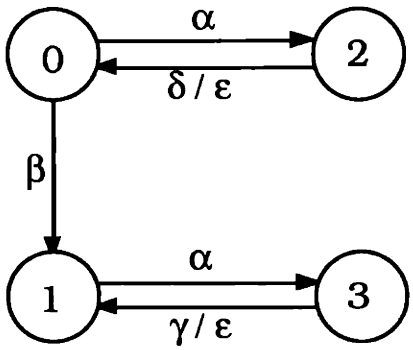

Figure 6.7: Example for Non-Resilient Inversion: State 0 is the initial state.

be any string in Γ. Then, the corrupted output is not necessarily an output string that can be generated by a state in X, and thus h_L^{-1} is undefined for this erroneous string. Therefore, we must define an inverter so that its response is defined for all such strings. We also require that the behavior of this inverter is equivalent to that of h_L^{-1} for uncorrupted strings:

Definition 6.16 A WD-inverter is a map $\mathsf{I} : \Gamma^* \rightarrow \Sigma^*$ so that for those strings that are in $L(A)$, I yields the same behavior as $h_{L(A)}^{-1}$, i.e., for all $s \in L(A)$, we require that $\mathsf{I}(h(s)) \subset h_{L(A)}^{-1}(h(s))$. Similarly, a WDX-inverter is a map $\mathsf{I} : \Gamma^* \rightarrow \Sigma^*$ so that for those strings that are in $L(A)$, I yields the same behavior as $h_{L(A)}^{-1}$, i.e., for all $s \in L(A)$, we require that $\mathsf{I}(h(s)) \subset h_{L(A)}^{-1}(h(s))$. ❑

Note that we require that the behavior of I is a subset of the inversion (as opposed to equivalent to the inversion), since if L is invertible, that portion of the inverted string that is of interest to us is unique and the inverter output for the ambiguous portion is not relevant. We term an inverter resilient if for all corrupted output strings, the inverter output, as compared to the actual event trajectory, is only incorrect within a bounded window around the burst. In the following formal definition of resiliency,

L_f denotes the set of strings in L that have an observable event as the last event:

Definition 6.17 A WD-inverter **I** is a resilient WD-inverter if there exists an integer n_b such that for all strings s in L, for all possible output strings t which can be generated by corrupting $h(s)$ with a finite length burst, i.e.,

- for all integers i,
- for all $t \in \Gamma^*$ such that $\xi(h(s), t) \leq i$ (let i' be the length of that part of s whose output is corrupted by the burst, i.e., let $s' \in L_f$ be the prefix of s such that $|h(s/s')| = i$ then $i' = |s/s'|$),

and for all possible completions r of s, i.e., for all $r \in L$ such that s is a prefix of r, there exists

- a prefix p_1 of s which is free of inversion errors in spite of the burst, i.e., p_1 is that prefix of s for which $|s/p_1| \leq n_b + i'$,
- a prefix p_2 of r so that the inversion error and the ambiguity can be confined to p_2/p_1, i.e., $|p_2/p_1| \leq 2n_b + i'$, and
- a prefix p_3 of r so that the inversion delay, as before, can be confined to r/p_3, i.e., $|r/p_3| \leq n_b$,

such that

$$\mathbf{I}(th(r/s)) \subset p_1(\Sigma \cup \epsilon)^{2n_b+i'}(p_3/p_2)(\Sigma \cup \{\epsilon\})^{n_b}$$

(Note that the observed string is $th(r/s)$, whereas r is what has actually occurred in the system.) L is resiliently WD-invertible if L is WD-invertible and a resilient WD-inverter exists. ❑

Similarly,

Definition 6.18 A WDX-inverter **I** is a resilient WDX-inverter if there exists an integer n_b such that for all strings s in $L(A)$, for all possible output strings t which can be generated by corrupting $h(s)$ with a finite length burst, i.e.,

- for all integers i,
- for all $t \in \Gamma^*$ such that $\xi(h(s), t) \leq i$ (let $s' \in L_f(A)$ be the prefix of s such that $|h(s/s')| = i$ and let $i' = |s/s'|$),

and for all possible completions r of s, i.e., for all $r \in L(A)$ such that s is a prefix of r, there exists

- a prefix p_0 of p_1, representing the ambiguity in the beginning, such that $p_1/p_0 \leq n_b$,
- a prefix p_1 of s which is free of inversion errors in spite of the burst, i.e., p_1 is that prefix of s for which $s/p_1 \leq n_b + i'$,
- a prefix p_2 of r so that the inversion error and the ambiguity can be confined to p_2/p_1, i.e., $|p_2/p_1| \leq 2n_b + i'$, and
- a prefix p_3 of r so that the inversion delay, as before, can be confined to r/p_3, i.e., $|r/p_3| \leq n_b$,

such that

$$\mathbf{I}(th(r/s)) \subset (\Sigma \cup \epsilon)^{n_b}(p_1/p_0)(\Sigma \cup \epsilon)^{2n_b+i'}(p_3/p_2)(\Sigma \cup \{\epsilon\})^{n_b}).$$

A is resiliently WDX-invertible if A is WDX-invertible and a resilient WDX-inverter exists. ❑

As the following result shows, a sufficient condition for resilient WD-invertibility is WD observability together with WD-invertibility, and we justify this as follows: If

A is WD observable, then a finite number of transitions after a burst, the observer estimate is guaranteed to include the actual state of the system. Moreover, using future outputs, we can exactly determine the actual state of the system. Since WD invertibility implies that the language generated by any state is WD invertible (see Proposition 6.5), we can invert correctly after a finite number of transitions:

Proposition 6.19 If L is WD-invertible and A is WD observable then L is resiliently WD-invertible.

Proof: Straightforward by the above reasoning using the fact that WD observers are resilient, as shown in Chapter 4, and using Proposition 6.5. ❑

The converse of Proposition 6.19 is not necessarily true. For example, consider the system illustrated in Figure 6.8, where all events are observable and 0 is the initial state. This system is clearly resiliently invertible since all events are observable. However, it is not WD observable since if only α occurs, we can never distinguish between the states 1 and 2.

To find necessary and sufficient conditions for resilient WD-invertibility, we first address the problem of resilient WDX-invertibility, since, as we have shown in Chapter 4, observation errors may lead to a complete loss of current state information. It turns out that WDX-invertibility is necessary and sufficient for resilient WDX-invertibility. Since WDX-invertibility is clearly necessary for resilient WDX-invertibility, we concentrate on showing sufficiency. In doing so, we assume WDX-invertibility and construct a resilient inverter. However, the construction for a resilient WDX-inverter is fairly complicated in this case. We start by using the WDX-inverter. If a burst never occurs, the inversion proceeds as before. In case of a burst, if the corrupted output trajectory is a feasible one, i.e., if it is in $h(L(A))$, the burst will never be detected. Later in this section, we show that our WDX-inverter works correctly in this

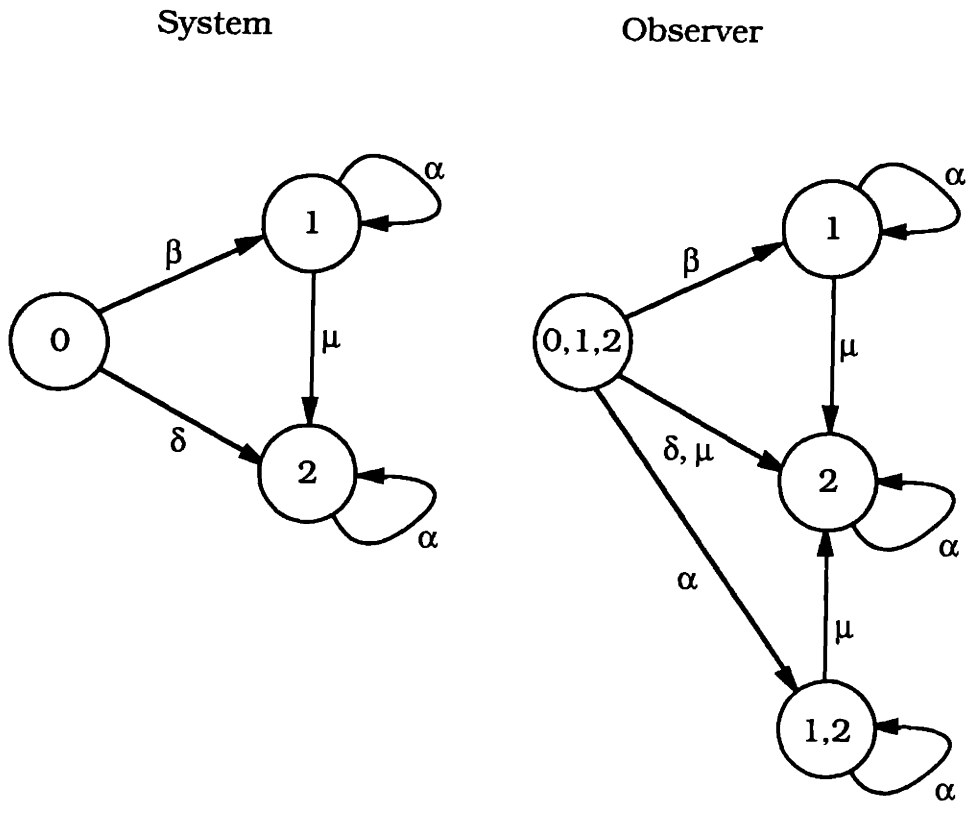

Figure 6.8: Counter Example to Necessity of WD Observability for Resilient Invertibility: All events are observable, 0 is the initial state.

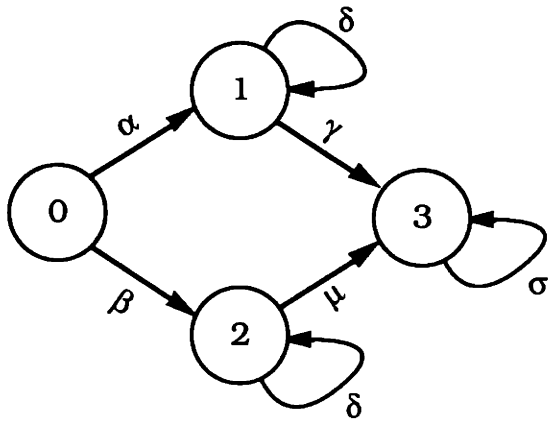

Figure 6.9: Example for Resilient Invertibility

case. However, if the corrupted string is not a feasible one then we need to modify our WDX-inverter so that we can incorporate this possibility. Since we have used the observer O_s as a basis for the WDX-inverter, our modifications will also involve modifications to how we use O_s.

Let us now examine the effect of an error burst on the inverter. Suppose that a burst occurs. As we noted above, it is possible that this burst is never detected. For example, consider the system in Figure 6.9, where all events are assumed to be observable so that inversion is trivial, i.e., the inverter is just the identity map. Suppose that $\alpha\delta^*$ occurs but $\beta\delta^*$ is observed. Then, the observation error is never detected, but inversion is resilient in that there is only a finite number of inversion errors (just one in this case) and they occur in a close vicinity of the error burst (in this case coincident with the burst). This example illustrates the general situation for a WDX-invertible system when an error burst occurs that is not detectable. In this case, thanks to the definition of WDX-invertibility, the WDX-inverter by itself will provide correct inversion after a finite period following the burst (since the WDX-

inverter is capable of working without knowledge of the "initial" state at the end of the burst).

Now, suppose that an inconsistent transition is observed, i.e., the current output γ is not defined at the current state $\hat{x}$ of O_s, or formally, $\gamma \notin v_s(\hat{x})$. This inconsistency may occur an arbitrary number of transitions after the burst. For example, in Figure 6.9, suppose that $\alpha\delta^i\gamma\sigma^*$ occurs, but $\beta\delta^i\gamma\sigma^*$ is observed. An inconsistency is detected when γ is observed and this point in time may be arbitrarily away from the burst since i may be arbitrarily large. When an inconsistency such as this occurs, it is not too difficult to see how to reset the inverter so that we can be certain that correct inversion will again begin at some point (t_c, see Figure 6.10) beyond the point at which we detected ambiguity (t_d, see Figure 6.10). Specifically, we can "reset" O_s to the state X_s, as we did when we considered resilient observability, and wait until the trajectory in O_s reaches W_s (as defined in Proposition 6.11) to start inverting again. Thanks to WDX-invertibility, the trajectory in O_s will reach W_s in a finite number of transitions (in fact, in q^2 transitions), and following this, the event trajectory can be inverted correctly as before.

However, as in the preceding example, the point t_d at which we detect the inconsistency may occur after an arbitrarily large time following the interval $[t_a, t_b]$ over which the actual measurement error burst occurred. What we would like to do, however, is to correct inversion errors (once we have detected a discrepancy) so that we can be sure that the inverted string is corrected except for a region around the actual error burst — i.e., as illustrated in Figure 6.10 we would like to find a number n_b and a backtracking procedure that guarantees that any inversion errors are confined to region B in Figure 6.10. Note that while the times t_d and t_c are known to us, t_a and t_b are not, so that our backtracking procedure must have a stopping rule that

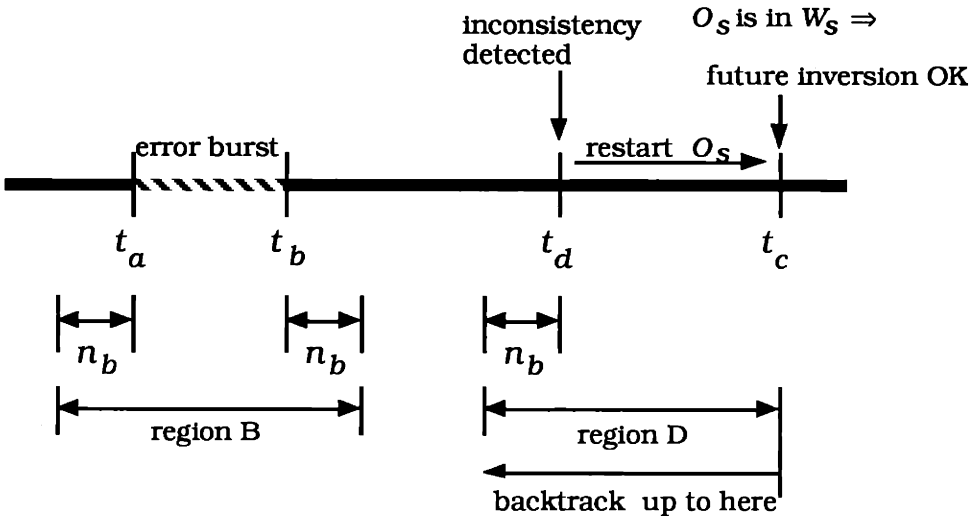

Figure 6.10: Illustration of Crucial Points in Time in Resilient Inversion: Particular choice of time indices illustrates the case when the inconsistency is detected arbitrarily away from the burst.

works without this knowledge. As illustrated in the figure our backtracking procedure starts from the point t_c, at which we are sure that processing in the future will be correct, and works backward in time. We will see that we need to backtrack at most n_b steps before t_d in order to guarantee that inversion errors are contained in region B.

Before we present our main result, let us provide a more complete picture of the possible situations that must be analyzed:

1. It is possible that an inconsistent transition never occurs, so that our WDX-inverter continues without change. In this case, region D in Figure 6.10 does not exist and we must show that any inversion errors are necessarily confined to B.

2. The point t_d lies sufficiently far from $[t_a, t_b]$ so that regions B and D do not

overlap. In this case what we must show is that the original inversion errors are confined to regions B and D and that our backtracking, which only covers region D, corrects the errors in D leaving any remaining errors confined to region B.

3. The point t_d lies outside $[t_a, t_b]$ but regions B and D overlap. In this case, it is obvious that the original inversion errors are confined to regions B and D (since their union covers everything from t_a through t_c). Note also that the backtracking step will cover the part of region B overlapping D. Thus, because of the original measurement errors, it is possible that we will encounter an inconsistency during the backtracking step. In this case we can be sure that we have backtracked into region B and can stop backtracking (so that our full backtracking step proceeds until n_b events before t_d or until an inconsistency is detected, whichever comes first). What we must show in this case is that backtracking corrects errors in that part of D lying outside of B.[2]

4. The point t_d lies inside $[t_a, t_b]$. The situation in this case is a bit more complex. Specifically, since there are still measurement errors following t_d, when we reset O_s at t_d and run it forward, we may observe another inconsistency before we reach t_c. In this case, we know that the error burst extends beyond our original t_d, and we simply reset O_s again using this latest point of inconsistency as our new t_d. This continues until we have a reset that leads to a successful achievement of the point t_c. In this case the associated t_d may lie outside $[t_a, t_b]$, in which case we are in one of the first three cases. The only remaining situation is one in which t_d is still within $[t_a, t_b]$. Note that in this case, since there are

[2]Note that in some cases, if n_b is sufficiently long and the error burst is short, we may backtrack to a point before t_a (but never outside B since $t_a \leq t_d$) so that the backtracking step may actually introduce new inversion errors, which, however, will be confined to B.

measurement errors following t_d, the correction procedure consists of restarting O_s and propagating it forward to t_c followed by backtracking may have errors in it. However, what we will see in this case is that region D is completely contained in region B so that inversion errors, even after our correction step, will be confined to B. Again, we may detect a subsequent inconsistency after this point, and this will then correspond once again to one of these four cases.

Proposition 6.20 *A is resiliently WDX-invertible iff A is WDX-invertible. Furthermore, if A is resiliently WDX-invertible then $n_b \leq \max[n_u \max(n_w, n_i, n_{ii}+1), n_r]$.*

Proof: ($\rightarrow$) Obvious. As shown in the inverter construction below, n_b depends on (a) the number of transitions it takes a trajectory from the initial state X_s of O_s to enter W_s, (b) the number of transitions it takes to distinguish between two distinguishable states in A, (c) one plus the number of transitions it takes to distinguish between two distinguishable states in A_r^{-1}, and (d) the minimum number of transitions it takes any state in A to reach a recurrent state. Thus, n_b is bounded by $\max[n_u \max(n_w, n_i, n_{ii}+1), n_r]$.

($\leftarrow$) This is a constructive proof in that it provides a construction for a resilient WDX-inverter. As described above, our resilient WDX inverter is the same as the WDX inverter if no inconsistencies are observed. When an inconsistency is observed, we restart O_s and wait until it enters W_s. If another inconsistency is observed after restarting O_s, we restart O_s again. After O_s enters W_s, we proceed in two directions: (1) We use the WDX inverter for future inversion. (2) We backtrack until an inconsistency is detected or until we have reached a point n_b steps before t_d. To prove the result we must pick n_b, specify the back tracking procedure, and analyze the four cases presented previously. We let $n_b = \max[n_u \max(n_w, n_i, n_{ii}+1), n_r]$ and we begin with case 1.

1. Suppose that no inconsistency is observed. From Proposition 6.13, we know that our WDX-inverter has made no errors any earlier than $n_u n_i$ events before t_a. What we will show is that it makes no errors any later than $n_u n_w$ events — or equivalently n_w observable events — after t_b. Suppose that at time t_b, A_s is in some state x and O_s is in some state $\hat{x}$. If $x \in \hat{x}$ then the inversion will be correct for all the transitions following the burst. Suppose that $x \notin \hat{x}$. Consider the states to which A_s and O_s move after n_w observed transitions, and call these y and $\hat{y}$. Then since O_s must enter W_s in at most n_w transitions (see the proof of Proposition 6.14), we must have that there is some $\hat{z} \in W_s$ so that $\hat{y} \cup \{y\} \subset \hat{z}$ (we can take $\hat{z}$ to be the state reached in these n_w observed transitions starting from some initial state in O_s that contains $\hat{x} \cup \{x\}$ — e.g., we can take the initial state to be <u>all</u> of X_s). If $y \in \hat{y}$, then subsequent inversions will clearly be correct. Otherwise, if we let $y = (s, w)$, then for all $(p, v) \in \hat{y}$ either $p = s$ or w and v are distinguishable. Also, since we observe no inconsistencies, the string we observe after this point is shared by $L(A_s, y)$ and $L(O_s, \hat{y})$. Let r be the next n_i observations. Recall that our WDX inverter, using r, eliminates the states in $\hat{y}$ that cannot generate r. Let $\hat{y}'$ be the set of states in $\hat{y}$ which can generate r. By WDX invertibility, for all $(p_0, v_0), (p_1, v_1) \in \hat{y}'$, $p_0 = p_1$. Also, thanks to Proposition 6.1, $p = s$ for all $(p, v) \in \hat{y}'$. Therefore, the inverter will produce the correct inversion at this point. Since these same conditions hold for all subsequent states in the trajectories of A_s and O_s, the inversion proceeds correctly.

Consider next the three other cases. What we first wish to do is to show that the inversion errors, before backtracking, are confined to regions B and D. In Cases 3 and 4 this is obvious, so we focus on Case 2. At an intermediate point between the regions B and D, let A_s be in state y and O_s be in state $\hat{y}$. Let r be the next n_i observations. Note that r ends before t_d and thus r can be generated by both y and

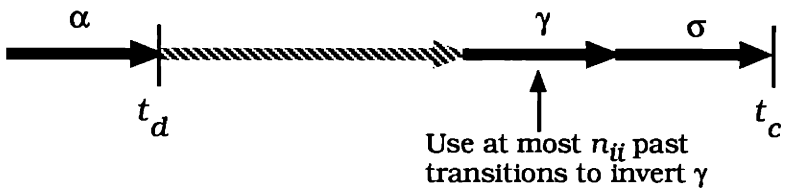

Figure 6.11: Proof of Resilient WD-invertibility (for the case 2(b)): Ordering of α, σ, and γ.

$\hat{y}$. Let $y = (s, w)$. As in the preceding proof of Case 1, we use Proposition 6.1 and conclude that the inversion at this point must be correct. Since this reasoning holds for any point between the regions B and D, the inversion error must be confined to these regions.

We must now specify the precise procedure to be used when an inconsistency is detected. Let α be the inconsistent transition at time t_d (see Figure 6.11). Since α is not defined at the current state of O_s, we start O_s from the state X_s after α occurs, and let it evolve. Let t_c be the point at which the state of O_s first enters W_s following the reset at t_d. Note first that, from Proposition 6.14, we know that there are at most $n_u n_w$ transitions between t_d and t_c. We can then conclude that in Case 4, i.e., when $t_a < t_d < t_b$, region D is completely contained in region B (see Figure 6.10). Then, we need only consider cases 2 and 3 in which $t_d > t_a$.

Let $\hat{x} \in W_s$ be the state of O_s at t_c and let σ be the event that caused this transition into W_s (see Figure 6.11). Then, for all $(s_1, y_1), (s_2, y_2) \in \hat{x}$, either $s_1 = s_2$ or y_1 and y_2 are distinguishable (see Figure 6.12 where the top ellipse denotes $\hat{x}$), and in addition, $h(s_1) = h(s_2) = \sigma$. Thus, thanks to distinguishability, using a finite number of future outputs (after σ), we can improve the estimate at t_c from $\hat{x}$, to, say $\hat{x}'$, such that for all $(s_1, y_1), (s_2, y_2) \in \hat{x}'$, $s_1 = s_2$, and therefore, we can reconstruct

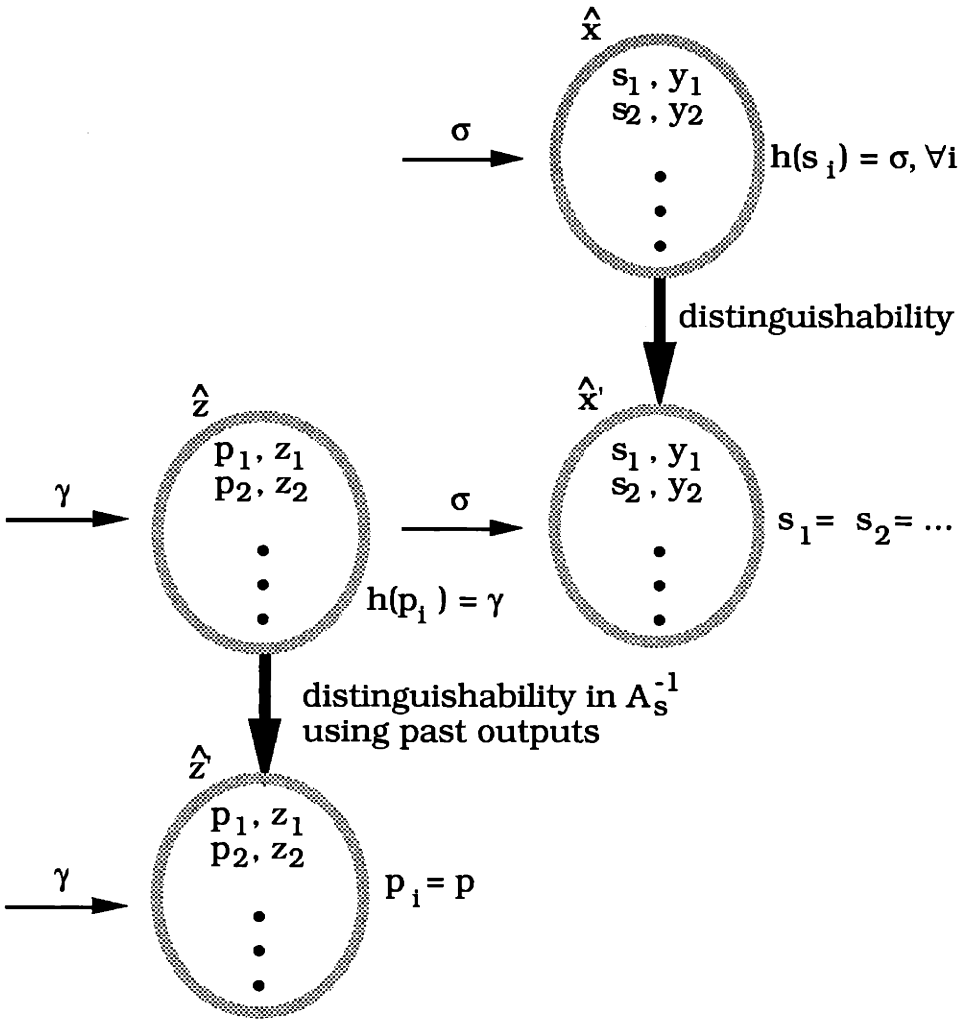

Figure 6.12: Proof of Resilient WD-invertibility (for the case 2(b)): Backtracking step.

the string corresponding to the output σ — i.e., the segment of the event trajectory in A ending with σ and preceded by unobservable events only (see Figure 6.12 where the second ellipse denotes $\hat{x}'$). We now begin the backtracking step: Let γ denote the last observable transition prior to σ (see Figure 6.11), and let us construct the set of states $\hat{z}$ in A_s that may reach a state in $\hat{x}'$ with one observable transition (σ) and such that the output corresponding to the first component of the state is γ, i.e., $\hat{z}$ consists of all $(r,x) \in X_s$ such that $h(r) = \gamma$ and $f_s((r,x),\sigma) \cap \hat{x}' \neq \emptyset$ (see Figure 6.12). Consider then any $(r_1,x_1),(r_2,x_2) \in \hat{z}$. If $(x_1,x_2) \notin I_M$ are distinguishable, we can use the n_i subsequent observations (γ, σ, and all but the last observation used to reconstruct σ) to decide between them. In this way we can reduce the size of $\hat{z}$ so that the remaining elements have $(x_1,x_2) \in I_M$.

Next, without loss of generality, we can eliminate all states in $\hat{z}$ that are <u>not</u> recurrent, for the following reason: If A_s is in a non-recurrent state, then at most n transitions could have occurred in the system, and thanks to that part of region B prior to the measurement errors, if we make any inversion errors because of eliminating non-recurrent states, then these errors will be in region B. So, we pick a pair of recurrent states $p_1 = (s_1,x_1), p_2 = (s_2,x_2) \in \hat{z}$ such that $s_1 \neq s_2$. Since also $h(s_1) = h(s_2)$, and $(x_1,x_2) \in I_M$, p_1 and p_2 satisfy the conditions of Proposition 6.15. Thus, by this proposition, for all $y_1, y_2 \in X_r$ such that $x_1 = f(y_1,s_1)$ and $x_2 = f(y_2,s_2)$, y_1 and y_2 must be distinguishable in A_r^{-1}. Then, we can also distinguish between x_1 and x_2 using n_{ii} observations prior to γ, or $n_{ii}+1$ observations including γ. Thus, by considering $n_{ii}+1$ outputs prior to $\hat{z}$, we can distinguish between different strings s_i in $\hat{z}$. Thus, using these prior outputs, we can construct a new set $\hat{z}'$ such that for all $(s_1,x_1),(s_2,x_2) \in \hat{z}'$, $s_1 = s_2$ (see Figure 6.12). Therefore, we can reconstruct the string corresponding to the output γ and can then repeat the process going back-

wards one observable event at a time. This backward reconstruction continues up to n_b transitions before t_d or until the observer estimate going backwards encounters an inconsistency (i.e., the $\hat{z}$ constructed above is empty), whichever comes first.

Finally, note that in order to correctly invert an observation γ in the backward inversion process that we have described, we need at most n_{ii} observations prior to γ. Since, by the definition of region B, the last n_b/n_u observations in region B are free of any measurement errors and $n_b/n_u \geq n_{ii} + 1$, the inversion for all points in region D that are outside of region B will be correct, proving our assertion for Cases 2 and 3. ❑

The following result immediately follows from this proposition and the fact that measurement errors may lead to the observation of inconsistent transitions:

Proposition 6.21 Given a WD-invertible L, L is resiliently WD-invertible iff A is WDX-invertible. Furthermore, if L is resiliently WD-invertible, then $n_b \leq \max[n_u \max(n_w, n_i, n_{ii} + 1), n_r]$. ❑

Note that the computational complexity of testing resilient WD-invertibility is the same as that of testing WDX-invertibility, and therefore is $O((\bar{l}_x q)^4)$.

6.4 Discussion

In this chapter, we have introduced notions of invertibility, and resiliency for discrete-event systems described by finite-state automata, and we have developed algorithms with polynomial complexity to test for invertibility, resiliency, and to construct resilient inverters. We have shown that the central element in these notions is the notion of stability that we considered in Chapter 3 and the notion of observability that we considered in Chapter 4.

The stability concepts that we introduced in Chapter 3 can be thought of as notions of error recovery or resiliency in that the system always returns to "good" states. In this chapter, we have carried this notion further by presenting an approach for reconstructing system behavior resiliently in spite of observation errors. Motivated by problems such as schedule-following in a flexible manufacturing system, one can also formulate regulator or tracking problems for DEDS in which a feedback system is sought so that the DEDS produces a particular desired sequence of output events. This tracking problem can also be thought of as a dual of the problem of invertibility that we have addressed. Analysis addressing these and related problems will be the subject of subsequent chapters.

Chapter 7
Tracking and Restrictability

7.1 Motivation and Background

In much of our work in the previous chapters, we have focused directly on what might be thought of as more standard control concepts of stability, observability, stabilization and output feedback. These chapters can be viewed as providing some of the elements required in order to develop a regulator theory for DEDS. In particular, to develop such a theory we need some notion of stability, and the one pursued in Chapters 3, 4, and 5, which can be thought of as an error recovery concept, appears to be a natural one in the discrete-event context.

In this chapter we develop another element which is needed for a regulator theory and which also is much closer to the linguistic concepts explored by others. In particular, we are concerned here with characterizing the tracking capabilities of a DEDS. In Chapter 6 we have developed and analyzed the concept of invertibility for DEDS, i.e., the ability to reconstruct the complete system event sequence given observations only of certain output events. Tracking is roughly the dual concept, as it is concerned with the construction of a compensator in order to force the output trajectory to follow a specified string. In this chapter, we provide a characterization of trackable languages, i.e., sets of strings that can be tracked at each state. We also formulate and analyze a second notion, restrictability, which is a slight generalization

of the notion of (language) controllability of Wonham and Ramadge in [39]. In particular, we define restrictability as the ability to control the system so that its output event behavior is in a given language. It is well-known that questions related to controllable languages can be NP-hard in the case of partial observations [46]. Drawing on our work in Chapters 4 and 5 on observability we are able to trace the source of this complexity to the cardinality of the observer state space of for a DEDS. That is, as we will see, if in fact the cardinality of the observer state space is polynomial in n, then the complexity of solving DEDS problems under partial observations will also be polynomial in n. This is of potential value as it makes it much easier to identify classes of systems for which analysis is far less complex than the worst case.

Our analysis of restrictability represents a relatively modest addition to the existing theory of controllable languages. However, we also consider two related notions which, we believe, are of some importance, and which are motivated by the desire to introduce notions of stability and error recovery in the theory of DEDS. The first of these concepts is that of eventual restrictability, i.e., the ability to restrict event behavior after a finite start-up period. This would appear to be a useful notion for capturing start-up or mode-switching behavior in DEDS. The second and more involved notion is that of reliable restrictability, i.e., the ability of the system to return to the desired, restricted behavior within a finite time following a burst of errors or failures. As we will see, eventual restrictability plays a key role in characterizing reliable restrictability.

In this chapter, we concentrate on the controlled system under perfect knowledge of the state and event trajectories. We will also include tracking events in our model. Specifically, the systems we consider in this chapter are of the form $A = (G, f, d, e, t)$ where $G = (X, \Sigma, U, \Xi)$. In parts of the next section, we will use this general frame-

work. In the rest of this chapter however, we assume the slightly more restrictive framework of [39] and Chapter 5 in which there is an event subset $\Phi \subset \Sigma$ such that we have complete control over events in Φ and no control over events in $\overline{\Phi}$, the complement of Φ. In this case, we can take $U = 2^{\Phi}$ and

$$e(x) = d(x[k]) \cap \overline{\Phi} \tag{7.1}$$

Also, in some cases in this paper, given $V \subset X$ which is f-invariant in A, it will be simpler to present our formulation using a restriction of A to V, denoted by $A|V$ and defined by the automaton (G_V, f, d, e, t), where $G_V = (V, \Sigma, U, \Xi)$. Note that $A|V$ is simply that part of A over the set of states V.

The tracking alphabet Ξ provides the flexibility to specify strings that we desire to track over an alphabet which may be much smaller than the entire event alphabet Σ. Note that if there exists a cycle in A that consists solely of events that are <u>not</u> in Ξ, then the system may stay in this cycle indefinitely, generating no event in Ξ. To avoid this possibility, we assume that it is <u>not</u> possible for our DEDS to generate arbitrarily long sequences of events in $\overline{\Xi}$. A necessary and sufficient condition for checking this is that if we remove the events in Ξ, the resulting automaton $A|\overline{\Xi}$ must be D_t-stable, where

$$D_t = \{x \in X | d(x) \subset \Xi\} \tag{7.2}$$

This is not difficult to check and will be assumed.

If this assumption is satisfied, then in some cases, it will be of value to consider an image automaton essentially defined over only those states in X that can be reached by the occurrence of some event in Ξ. We thus define the following sets:

$$Y_0^t = \{x \in X | \nexists y \in X, \sigma \in \Sigma, \text{ such that } x \in f(y, \sigma)\} \tag{7.3}$$

$$Y_1^t = \{x \in X | \exists y \in X, \tau \in \Xi, \text{ such that } x \in f(y, \tau)\} \tag{7.4}$$

$$Y^t = Y_0^t \cup Y_1^t \tag{7.5}$$

Thus, Y^t is the set of states x such that either there exists a tracking event defined from some state y to x (as captured in Y_1^t) or x has no transitions defined to it (as captured in Y_0^t; we include these transient states to capture possible start-up behavior). Let $r = |Y^t|$.

7.2 Tracking

In this section, we first present our notion of tracking, and present an algorithm for computing the supremal collection of strings that can be tracked. Later in this section, we present our notion of eventual tracking which is an extension of our notion of stability. Specifically, we consider the tracking of desired strings after a transient period of a finite number of transitions. For the system A, we will assume the more restrictive framework of Wonham and Ramadge, i.e., that an event controllable at some state is controllable at all the other states. However, various automata that we define in computing trackable strings will belong to the more general framework in Chapter 3. Furthermore, in order to simplify our presentation of these notions, we will assume that $\Phi \subset \Xi$, i.e., that all controllable events are also in the tracking alphabet.

7.2.1 Trackable Languages

We define tracking as being able to restrict the system behavior so that the automaton starting from current state must generate a desired string:

Definition 7.1 Given $x \in X$, a string $s \in \Xi^*$ is trackable from x if we can find a compensator $C : X \times \Sigma^* \to U$ such that A_C is alive and $t(L(A_C, x)) \subset s\Xi^*$. ❑

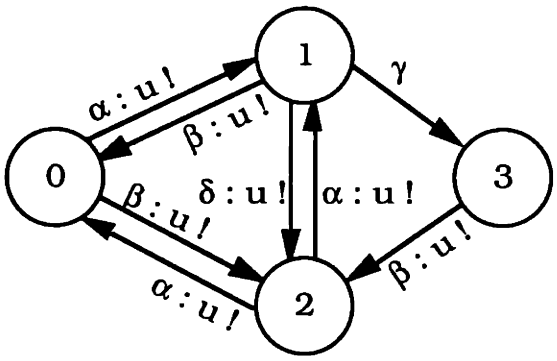

Figure 7.1: A Simple Example

As an example, consider the system in Figure 7.1. Any string in $(\alpha\beta)^*$ is trackable from 0, and a compensator for tracking all such strings can be defined by $C(0,\emptyset) = \{\alpha\}, C(1,\alpha) = \{\beta\}, C(3,\alpha\gamma) = \{\beta\}, C(0,\alpha\beta) = \{\alpha\}$, etc. As seen in this example, if a string is trackable then a compensator for tracking it can be constructed easily. Specifically, this compensator should only enable the next event in the string that we wish to track given the part of the string that has already been tracked.

Definition 7.2 A language L is a trackable language from x if it is complete and if each string in L is trackable from x. ❑

Note that the class of trackable languages is closed under arbitrary unions and let $L_T(A,x)$ denote the supremal language trackable from x. On the other hand, the class of trackable languages is not necessarily closed under intersections since the intersection of two complete languages L_1 and L_2 is not necessarily complete, even though this intersection is prefix closed. However, we can construct the supremal complete sublanguage of the intersection. Let the function $\chi : 2^{\Xi^*} \rightarrow 2^{\Xi^*}$ denote removing all the strings, in a given language L, that have no infinite extensions in

L, i.e.,

$$\chi(L) = \{s \in L |\ s \text{ has an infinite extension in } L\} \tag{7.6}$$

Then, $\chi(L_1 \cap L_2)$ is a trackable language.

Given some $x \in X$, let us examine the properties of $L_T(A, x)$: First of all, the first event of a string $s \in L_T(A, x)$ has to be defined at some state that is reachable from x by events in $\overline{\Xi}$, i.e., it has to be in the set

$$d^t(x) = d(R(A|\overline{\Xi}, x)) \cap \Xi \tag{7.7}$$

In Figure 7.1, the first event of a string in $L_T(A, 1)$ may be either β or δ.

Second, the first event of s, say τ, has to be trackable from x. We now characterize the set of such events. Let $l_T(x)$ be this set (i.e., the strings of length 1 in $L_T(A, x)$), and let $e^t(x)$ be the set of events in $d^t(x)$ that are either uncontrollable, or events such that if an event in this set is disabled, then some state in $R(A|\overline{\Xi}, x)$ is no longer alive, i.e.,

$$e^t(x) = (d^t(x) \cap \overline{\Phi}) \cup \{\tau \in d^t(x) | \exists y \in R(A|\overline{\Xi}, x) \text{ such that } d(y) = \{\tau\}\} \tag{7.8}$$

For example in Figure 7.1, $e^t(1) = \{\beta\}$, and e^t is the empty set for all other states. Note that if $e^t(x)$ contains more than one event, then we cannot track <u>any</u> event from x, and if it contains one event, then we can <u>only</u> track that event from x. Finally, if $e^t(x)$ is empty, then we can track all events in $d^t(x)$ from x. Thus,

Proposition 7.3 $l_T(x) = \{\tau \in d^t(x) | \{\tau\} \cup e^t(x) = \{\tau\}\}$ ❑

For example in Figure 7.1, $l_T(1) = \{\beta\}$.

After some $\tau \in l_T(x)$ is tracked, the automaton is in some state in $f^t(x, \tau) \triangleq f(R(A|\overline{\Xi}, x), \tau)$. Consequently, the remaining part of the string that can be tracked, with τ as prefix, must be trackable from <u>all</u> these states. Thus, we have the following implicit characterization of $L_T(A, x)$:

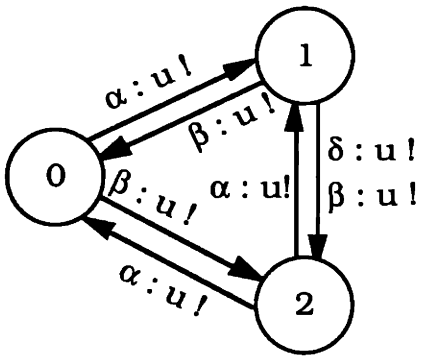

Figure 7.2: The Automaton A^t for Figure 7.1

Proposition 7.4 $L_T(A,x) = \bigcup_{\tau \in l_T(x)} \tau\chi(\bigcap_{y \in f^t(x,\tau)} L_T(A,y))$ ❑

In order to solve this equation, we first construct an automaton $A^t = (G^t, f^t, d^t, e^t, 1)$ where $G^t = (Y^t, \Xi, U, \Xi)$ and 1 is the identity map. For the system in Figure 7.1, A^t is illustrated in Figure 7.2.

Recall that if $e^t(x)$ contains one element, then we can only track that event from x. Thus, we might as well disable all the other events at such a state, i.e., let

$$K'(x) = \begin{cases} \emptyset & \text{if } e^t(x) \neq \emptyset \\ d^t(x) & \text{otherwise} \end{cases} \tag{7.9}$$

In Figure 7.2, $K'(1) = \emptyset$ since $e^t(1) = \beta$ (thus δ is disabled at state 1) and $K'(0) = \{\alpha, \beta\}, K'(2) = \{\alpha\}$. Also, recall that if $e^t(x)$ contains more than one event, then we cannot track any event from x. Let D_T represent such states, i.e.,

$$D_T = \{x \in Y^t | e^t(x) \geq 2\} \tag{7.10}$$

In addition, in order to be able to track complete languages, we need to keep other states away from D_T while preserving liveness. Thus, let V be the maximal sustainably (f,u)-invariant subset of $\overline{D}_T$ in $A^t_{K'}$ and let K_V be the associated minimally restrictive feedback. In Figure 7.2, V is all of the states. Finally, let $K(x) =$

$K_V(x) \cap K'(x)$. Note that for all $x \in V$, all the events in $d_K^t(x)$ are trackable from x, i.e., $l_T(x) = d_K^t(x)$, and in addition:

Lemma 7.5 If $x \in Y^t \cap \overline{V}$, then $L_T(A, x) = \emptyset$. If $x \in V$, then $L_T(A, x) \subset L(A_K^t, x)$.

Proof: Straightforward. ❑

In order to compute $L_T(A, x)$, let us first focus on the case in which $A_K^t|V$ is deterministic. In this case, since $l_T(x) = d_K^t(x)$ for all $x \in V$, then for any $x \in V$, the language generated from x in A_K^t is certainly trackable from x, and in fact it is the supremal such language:

Proposition 7.6 If $A_K^t|V$ is deterministic, then for all $x \in V$, $L_T(A, x) = L(A_K^t, x)$. Furthermore, for all $x \in Y^t \cap \overline{V}$, $L_T(A, x) = \emptyset$.

Proof: Straightforward using Lemma 7.5 and the fact that for all $x \in V$ and $\tau \in d_K^t(x)$, $f_K^t(x, \tau)$ is single valued. ❑

To complete the picture when $A_K^t|V$ is deterministic, we must construct $L_T(A, x)$ for the states x in $\overline{Y}^t$. Let us look first for any such x that is also in $R(A|\overline{\Xi}, V)$. That is, there exists $y \in V$ such that x can be reached from y without the occurrence of tracking events. Consider then any $\tau \in l_T(x)$. By definition $f^t(x, \tau) \subset Y^t$. Furthermore, $f^t(x, \tau) \subset f^t(y, \tau)$. Thus, there are two possibilities. Either $f^t(y, \tau)$ is contained in V or it is not. If it is, then, since K is a minimally restrictive feedback and since $A_K^t|V$ is deterministic $f^t(x, \tau) = f^t(y, \tau) =$ a single element of V. If $f^t(y, \tau)$ is not contained in V, then the feedback K must disable this event in order to achieve invariance for V. Thus we must also disable this event at x.[1] Thus, if we define

$$l_V(x) = \{\tau \in l_T(x) | f^t(x, \tau) \in V\} \tag{7.11}$$

[1]The existence of the feedback K in fact guarantees that τ can be disabled while preserving liveness.

then

$$L_T(A,x) = \bigcup_{\tau \in l_V(x)} \tau L_T(A, f^t(x,\tau)) \tag{7.12}$$

which allows us to compute $L_T(A,x)$ from $L_T(A,y)$, $y \in V$. Next suppose that $x \notin R(A|\overline{\Xi},V)$ and take any $\tau \in l_T(x)$. Again, there are two possibilities: either $f^t(x,\tau) \subset V$ or $f^t(x,\tau) \not\subset V$. Consider the second possibility in which we know that $\tau \notin L_T(A,x)$. There are two cases here: either $\tau \in e^t(x)$ or $\tau \notin e^t(x)$. In the first of these, we cannot disable τ and thus $L_T(A,x) = \emptyset$. In the latter, we simply disable τ. Consider next the possibility $f^t(x,\tau) \subset V$. There are two cases here as well: either $|f^t(x,\tau)| = 1$ or $|f^t(x,\tau)| > 1$. In the former case we know that

$$L_T(A,x) \supset \tau L_T(A, f^t(x,\tau)) \tag{7.13}$$

and indeed if only this case occurs, $L_T(A,x)$ is given as in (7.12). However, if $|f^t(x,\tau)| > 1$ for some τ, we have a situation exactly as in the nondeterministic case: essentially we must intersect the languages $\tau L_T(A,y)$ for all $y \in f^t(x,\tau)$. As this procedure is embedded in the fully nondeterministic case, we describe this case next.

If $A_K^t|V$ is nondeterministic, we first construct a deterministic automaton O^t over subsets of V such that for each state $\hat{x}$ of O^t, the events defined at $\hat{x}$ are given by the intersection of the events defined at each element $x \in \hat{x}$. In particular, we construct an automaton $O^t = (F^t, w^t, v^t)$ over the states 2^V with

$$w^t(\hat{x},\tau) = \bigcup_{x \in \hat{x}} f^t(x,\tau) \tag{7.14}$$

$$v^t(\hat{x}) = \bigcap_{x \in \hat{x}} (d^t(x) \cap K(x)) \cup e^t(x) \tag{7.15}$$

These dynamics can be defined with all of 2^V as the state space. Since we will only use particular initial states, we can restrict attention to the reach of these states

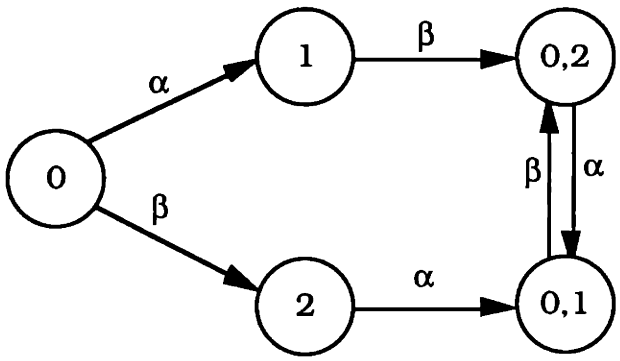

Figure 7.3: The Automaton O^t for Figure 7.2

under these dynamics. Specifically, we take the state space Z^t of O^t to be

$$Z^t = R(O^t, \{\hat{x} \in 2^V | \hat{x} = \{x\} \text{ and } x \in V, \text{ or } \hat{x} = f^t(x, \tau) \text{ for some } x \in \overline{Y}^t, \tau \in l_T(x)\}) \tag{7.16}$$

Figure 7.3, illustrates O^t for the automaton in Figure 7.2. Note that $l_T(3) = \{\beta\}$ and $f^t(3, \beta) = \{2\}$.

Let D_z be the set of dead states in Z^t, i.e.,

$$D_z = \{\hat{x} \in Z^t | v^t(\hat{x}) = \emptyset\} \tag{7.17}$$

let Z_V^t be the maximal sustainably (f, u)-invariant subset of $\overline{D}_z$; and let K^t be the associated minimally restrictive feedback. Then, we have the following result, where $Z_s = \{x | \{x\} \in Z_V^t\}$:

Proposition 7.7 Given $x \in Z_s$,

$$L_T(A, x) = L(O_{K^t}^t, \{x\})$$

Given $x \in Y^t \cap \overline{Z}_s$,

$$L_T(A, x) = \emptyset$$

Finally, given $x \in \overline{Y}^t$, let

$$l'_T(x) = \{\tau \in l_T(x) | f^t(x,\tau) \in Z_V^t\}$$

then

$$L_T(A,x) = \bigcup_{\tau \in l'_T(x)} \tau L(O^t_{K^t}, f^t(x,\tau))$$

Note that if $l'_T(x) = \emptyset$ then $L_T(A,x) = \emptyset$. ❑

The proof of this result is straightforward. Because of the nondeterminism we need to make sure that, for any prefix of a trackable string, the corresponding suffix is trackable from all states that can be reached by applying the prefix. The dynamics w^t defined via a union (7.14), and the allowable event function v^t, defined via an intersection (7.15), capture this exactly. A dead state $\hat{x} \in D_z$ then corresponds to a set of states such that no event is trackable from all elements of $\hat{x}$, and thus we must avoid these states and confine the dynamics to Z_V^t. For any singleton element of Z_V^t, i.e, any $\{x\} \in Z_V^t$, it is then easy to compute $L_T(A,x)$. For any other singleton that can be reached by a trackable event, i.e., $x \in Y^t \cap \overline{Z}_s$, we know that the trackable language is empty, since we have started outside of Z_V^t. Finally, for $x \in \overline{Y}^t$, the only trackable events are those which drive x completely within Z_V^t, i.e., $l'_T(x)$, and from there we can compute the suffixes of the trackable strings from x using the dynamics evolving within Z_V^t. Finally, as we have commented earlier, constructing a compensator for tracking any $s \in L_T(A,x)$ is easy: We just enable the next event that we wish to track given the string that has already been tracked.

The complexity of computing $L_T(A,x)$ for all x is quadratic in $|Z^t|$. However, as with the cardinality of the state space of an observer Chapter 4, $|Z^t|$ may be exponential in $|V|$, and thus computing L_T may have exponential complexity in $|V|$. In Chapter 4 we provided some bounds on observer state space size and gave examples

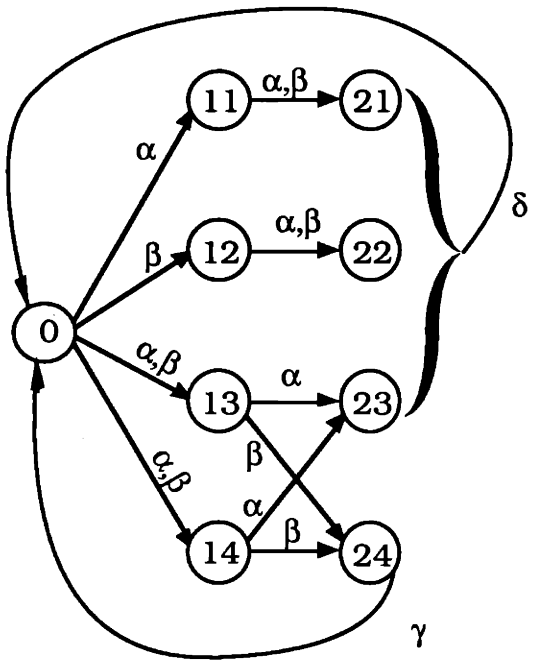

Figure 7.4: A Class of Systems $i = 2$

showing that in many cases the actual observer state space size may be considerably smaller than the worst case exponential bound. Similar analysis can be performed in the present context and indeed the bounding procedure of Chapter 4 can be used to compute a bound on the size of the recurrent part of Z^t. In the following example we illustrate both our procedure for computing $L_T(A, x)$ and the worst-case bound using an adaptation of the example used for analogous purposes in Chapter 4.

Example 7.8 Consider a family of systems indexed by an integer i. Each system has $2i^2 + 1$ states and $\Sigma = \Xi = \Phi = \{\alpha, \beta, \delta, \gamma\}$. The system for $i = 2$ is illustrated in Figure 7.4, and $i = 3$ is illustrated in Figure 7.5. The element of this class of systems for a given i can be constructed as follows: First, let us label one of the states by 0 and the rest by a pair of integers (j, k), $1 \leq j, k \leq i$, and arrange

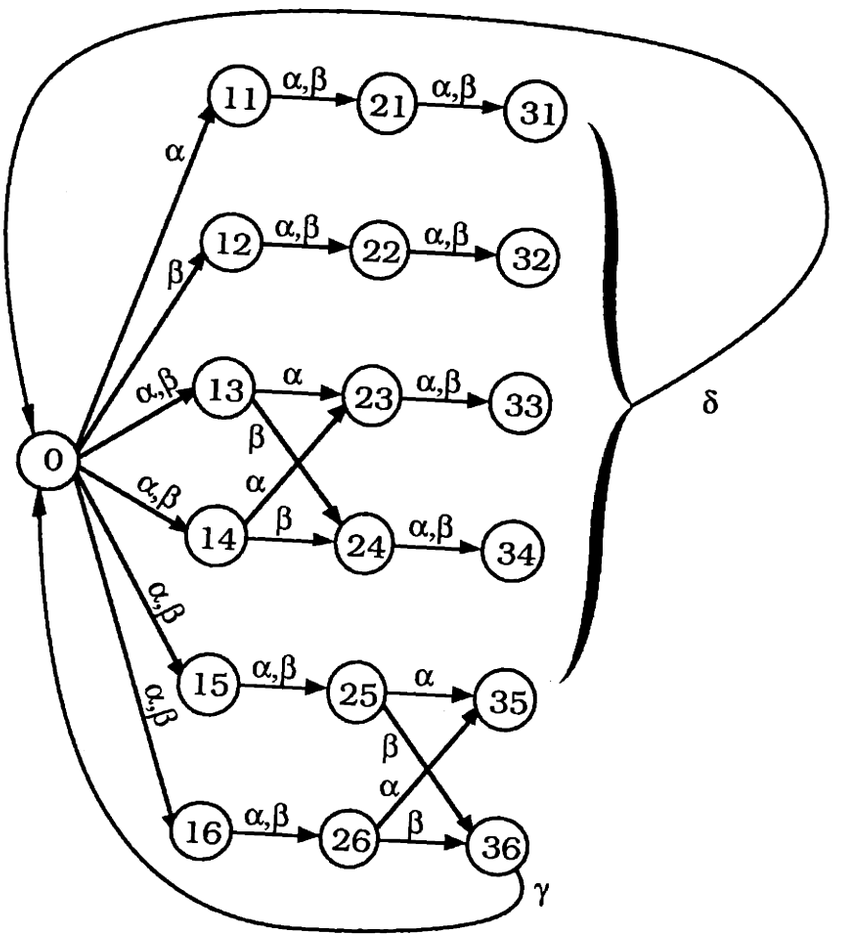

Figure 7.5: A Class of Systems $i = 3$

these states in a matrix so that the first (respectively, the second) index of a state refers to the column (respectively, the row) that the state is in. The event γ is only defined at state $(i, 2i)$ and it defines a transition from state $(i, 2i)$ to state 0. The event δ is defined at all the other states in the last column and it defines transitions from each of these states to state 0. Both α and β are defined at all the other states in A. The transition structure corresponding to α and β is chosen so that for each string of length $0 < j \leq i$, there is a different set of $2i - j$ states that can be reached from 0 by application of this string. In detail, at state 0, α defines transitions to state $(1,1)$ and states $(1,3)$ through $(1, 2i)$, and β defines transitions to states $(1,2)$ through $(1, 2i)$. Let us now consider the states that are in the first two rows but not in the last column. For each such state (j, k), both α and β define transitions to state $(j + 1, k)$. The transitions at the states in the third and fourth rows are defined as in the first two rows, except that for the two states that are also in the first column, α and β define transitions according to the cross pattern (from states 13 and 14 to states 23 and 24) in Figure 7.4. Fifth and sixth rows are also constructed similarly, except that the cross pattern is between columns two and three. In general, rows k and $k + 1$, for odd k, are constructed similarly, except that the cross pattern is between columns $k - 1$ and k.

Let A be the system in Figure 7.4. Note that $D_T = \emptyset$, $V = X$, and thus $A_K^t | V = A$. Note also that A is nondeterministic but only transitions at state 0 are nondeterministic. Therefore, if we construct $L_T(A, 0)$ then it is trivial to construct L_T for all the other states. For example,

$$L_T(A, (1,1)) = L_T(A, (1,2)) = (\alpha + \beta)\delta L_T(A, 0)$$

$$L_T(A, (1,3)) = (\alpha\delta + \beta\gamma) L_T(A, 0)$$

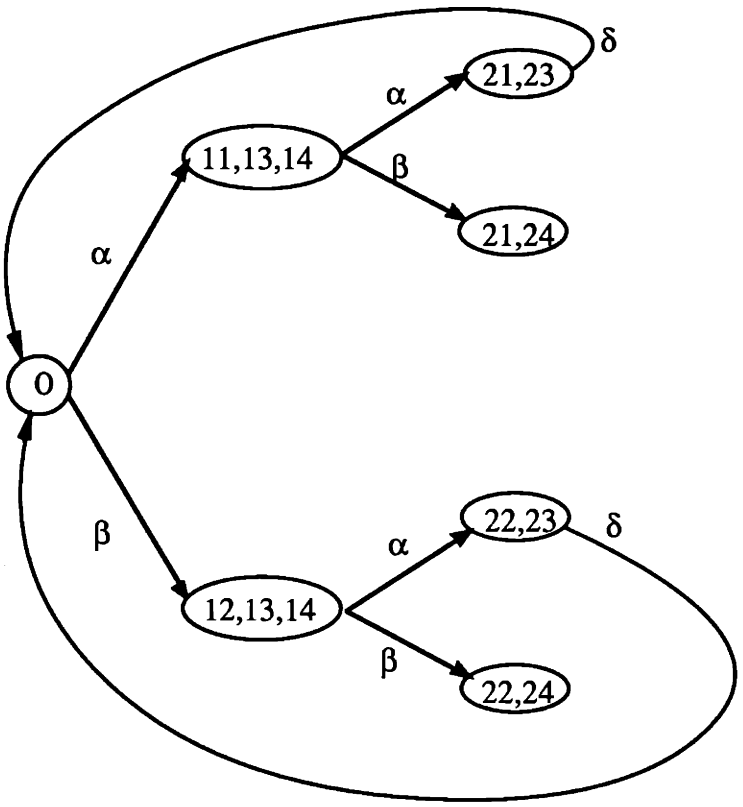

Figure 7.6: Range of $\{0\}$ in O^t for Figure 7.4

Let us now construct $L_T(A,0)$: The automaton O^t, for the range of $\{0\}$, is illustrated in Figure 7.6. In this case,

$$Z_V^t = \{\{0\},\{11,13,14\},\{12,13,14\},\{21,23\},\{22,23\}\}$$

Therefore, $\{0\} = Z_s$ and $L_T(A,0) = (\alpha\alpha\delta + \beta\alpha\delta)^*$.

It is straightforward to check that, in the general case, the cardinality of the range of $\{0\}$ in O^t is $2^{i+1}-1$. Thus, the cardinality of Z^t for such class of systems is at least $2^{i+1}-1$. ❑

7.2.2 Eventually Trackable Languages

A straightforward generalization of the notion of tracking is a notion of tracking a given string in a finite number of transitions. For example, in Figure 7.1, $(\beta\alpha)^*$ is trackable from state 2 in one transition, namely after the occurrence of α. We term this a notion of eventual trackability. In the following definition, $(\Xi \cup \{\epsilon\})^{n_t}$ denotes the set of all strings, over Ξ, of length at most n_t, where ϵ denotes the "null" string:

Definition 7.9 Given $x \in X$, a string $s \in \Xi^*$ is eventually trackable from x if there exists an integer n_t and a compensator $C : X \times \Sigma^* \to U$ such that A_C is alive and $t(L(A_C, x)) \subset (\Xi \cup \{\epsilon\})^{n_t} s \Xi^*$. A language L is eventually trackable from x if it is complete and if each string in L is eventually trackable from x. ❑

Similar to the class of trackable languages, the class of eventually trackable languages is closed under arbitrary unions and the supremal complete sublanguage of the intersection of two eventually trackable languages is also eventually trackable. Let $L_{ET}(A, x)$ denote the supremal language eventually trackable from x.

As stated by the following result, if a state x is E-pre-stabilizable for some $E \subset X$, then any string trackable from all states in E is eventually trackable from x. For the example in Figure 7.1, 2 is $\{0, 1\}$-pre-stabilizable and $(\beta\alpha)^*$ is trackable from both 0 and 1.

Lemma 7.10 Given $x \in X$ and $E \subset X$ such that x is E-pre-stabilizable,

$$L_{ET}(A, x) \supset \bigcap_{y \in E} L_T(A, y)$$

Proof: Straightforward. ❑

Conversely, suppose that some string s is eventually trackable from some state x, and let E_s be all the states from which s is trackable. Then, x must be E_s-pre-

stabilizable since otherwise, a trajectory from x may cycle arbitrarily through states from which s is not trackable:

Proposition 7.11 Given x, let $\mathbf{E}$ be the set of all sets $E \subset X$ such that x is E-pre-stabilizable. Then

$$L_{ET}(A,x) = \bigcup_{E\in\mathbf{E}} \bigcap_{y\in E} L_T(A,y)$$

Furthermore, for all $x \in X$ and $s \in L_{ET}(A,x)$, $n_t \leq r = Y^t$.

Proof: Straightforward. To prove the second statement, note that n_t can be chosen as the maximum number of tracking events on any trajectory from some $x \in X$ to E. Since r is the cardinality of Y^t, it is an upper bound on n_t. ❑

We can obtain a slightly tighter formula for $L_{ET}(A,x)$ as follows. Let $Y' \subset X$ be the set of states from which at least one tracking event is defined, i.e.,

$$Y' = \{x \in X | d(x) \cap \Xi \neq \emptyset\} \tag{7.18}$$

Let $r' = |Y'|$.

Corollary 7.12 Given x, let $\mathbf{E}'$ be the set of all sets $E' \subset Y'$ such that x is E'-pre-stabilizable. Then

$$L_{ET}(A,x) = \bigcup_{E'\in\mathbf{E}'} \bigcap_{y\in E'} L_T(A,y)$$

Proof: ($\supset$) Trivial by the above proposition.

($\subset$) Let $s \in L_{ET}(A,x)$, and let $E \subset X$ be a set so that x is E-pre-stabilizable and $s \in L_T(A,x')$ for all $x' \in E$. Next, let

$$E' = R(A|\overline{\Xi}, E) \cap Y'$$

Thanks to our assumption that it is not possible for A to generate arbitrarily long sequences of events in $\overline{\Xi}$, E is E'-pre-stable. Thus, x is E'-pre-stabilizable and $E' \in \mathbf{E}'$.

Also, since all events in $\overline{\overline{\Xi}}$ are uncontrollable, $s \in L_T(A, y)$ for all $y \in E'$. Therefore, $s \in \bigcup_{E' \in \mathbf{E}'} \bigcap_{y \in E'} L_T(A, y)$. ❑

Note that in general in order to compute $\mathbf{E}'$ we need to check, for each subset E' of Y', if x is E'-pre-stabilizable. Therefore, computing $L_{ET}(A, x)$ has complexity exponential in r'. However, testing if a string s is eventually trackable or not (from some state x) may or may not have exponential complexity, depending on the complexity of the state space of O^t since all we need to do is to compute the set of states in Y' from which s is trackable and test if x is pre-stabilizable with respect to this set. For example, $(\beta\alpha)^*$ is trackable from 0 and 1 in Figure 7.1. Since 2 is $\{0, 1\}$-pre-stabilizable, $(\beta\alpha)^*$ is eventually trackable from 2. In fact, both $(\alpha\beta)^*$ and $(\beta\alpha)^*$ are eventually trackable from all the states.

7.3 Restrictability

In this section, we first address the problem of restricting the output behavior of a system to a given language. This notion is a slight generalization of the notion of controllable languages in the Wonham and Ramadge framework as we also consider arbitrary initial states. Later in this section, we present a result precisely relating their notion to our framework. Next, we present the concept of eventual restrictability which allows us the flexibility of restricting the behavior after a finite number of transitions. Finally, we present and analyze "a notion" of reliability which allows us to model failure or error events and to test if the system can be made to recover its restriction capabilities following the occurrence of a burst of errors. Throughout this section we consider the general setting in which Φ need not be contained in Ξ.

7.3.1 Basic Notion

Given a complete language L over Ξ and a state x, our notion of restrictability is defined as the ability to control the system so that all the trajectories generated from x in the closed loop system are in L:

Definition 7.13 Given $x \in X$ and a complete language L over Ξ, x is L-restrictable if there exists a compensator $C : X \times \Sigma^* \to U$ such that the closed loop system A_C is alive and $t(L(A_C, x)) \subset L$. Given $Q \subset X$, Q is L-restrictable if all $x \in Q$ are L-restrictable. Finally, A is L-restrictable if X is L-restrictable. ❑

Clearly, the class of L-restrictable sets is closed under arbitrary unions and intersections. Let X_L denote the maximal L-restrictable set. In order to compute X_L, we first construct a recognizer for L and then formulate the problem of restrictability as a problem of stabilizability of the composite of this recognizer and A. In the rest of this section, we present this approach and establish connections to the work of Wonham and Ramadge.

Let (A_L, x_0) be a minimal recognizer for L and let Z_L denote its state space. Let A'_L be an automaton which is the same as A_L except that its state space is $Z'_L = Z_L \cup \{b\}$ where b is a state used to signify that the event trajectory is no longer in L. Also, we let $d'_L(x) = \Xi$ for all $x \in Z'_L$, and

$$f'_L(x,\sigma) = \begin{cases} f_L(x,\sigma) & \text{if } x \neq b \text{ and } \sigma \in d_L(x) \\ \{b\} & \text{otherwise} \end{cases} \tag{7.19}$$

As an example, consider the system illustrated in Figure 7.7(a) which is identical to an example in [50]. We have two simple automata, each of which can be thought of as a machine in a manufacturing system. Each of these machines has two states so that state 0 corresponds to being idle and 1 corresponds to working on a part.

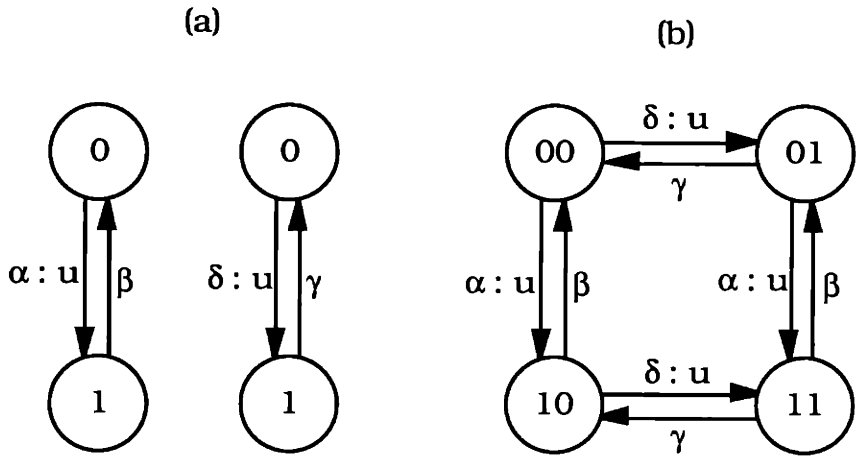

Figure 7.7: Example for Restrictability

Event α (respectively, δ) signifies that the first (respectively, second) machine started working, and event β (respectively, γ) signifies that the first (respectively, second) machine is finished with the part. Events α and δ are assumed to be controllable. Their composition, which models all the behavior that can be generated by the two machines is illustrated in Figure 7.7(b), and we let this composition be our automaton A. Suppose that the first machine feeds the second one, i.e., after the first machine is finished with a part, the second one starts working on it, and suppose that there is a buffer of size one between the two machines. Our goal is to design a compensator such that the buffer never overflows, i.e, at any given time, there can be at most one part in the buffer. This implies that the set of strings that we wish to allow needs to have β and δ alternate. A recognizer for this language, and in fact the automaton A'_L with the initial state 0, is illustrated in Figure 7.8, where we have taken $\Xi = \{\beta, \delta\}$ as the tracking alphabet.

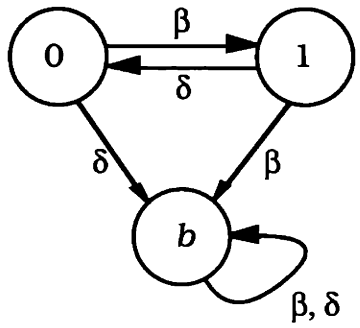

Figure 7.8: Automaton A'_L

Let A^{LA} denote the composite $A'_L \parallel A$ and let

$$E^{LA} = \{(x, y) \in X^{LA} | x \neq b\} \tag{7.20}$$

For example, Figure 7.9 denotes the composite of the automata in Figure 7.7(b) and Figure 7.8, where the first component of the labels of each state represent the state of A'_L and the last two represent the state of A, and transitions defined at states with the first component equal to b have been ignored for simplicity. Note that E^{LA} is the set of all states which do not have b as their first component.

Given $Q \subset X$, recall that $I(Q)$ denotes the maximal sustainably (f, u)-invariant subset of Q (see Chapter 3) and let K_I denote the associated minimally restrictive feedback. Then, we have the following result:

Proposition 7.14 A state $x \in X$ is L-restrictable iff $(x_0, x) \in I(R(A^{LA}, (x_0, x)) \cap E^{LA})$. Furthermore, a compensator for restricting the behavior of x to L can be constructed using the closed loop automaton $A^{LA}_{K_I} = (G', f', d')$ and the initial

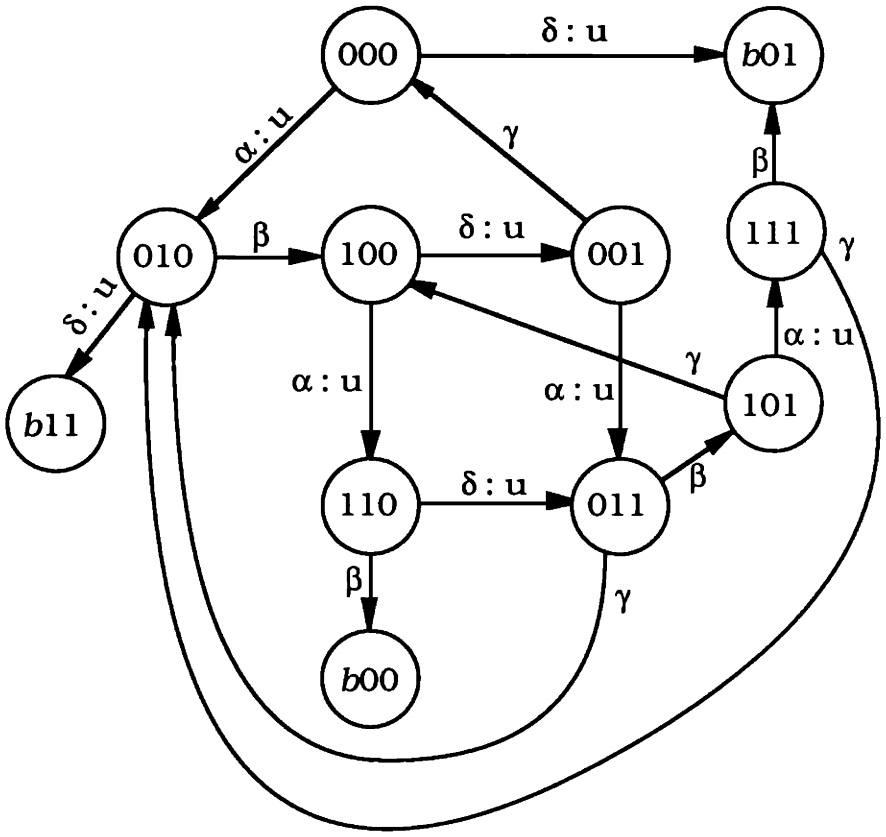

Figure 7.9: Composite of A and A'_L

state (x_0, x) as follows:

$$C(y,s) = \begin{cases} d'((x_0,x)) & \text{if } s = \epsilon \\ d'(f'((x_0,x),s)) & \text{if } s \in L(A^{LA}_{K_I},(x_0,x)) \\ \text{don't care} & \text{otherwise} \end{cases}$$

Proof: Straightforward by assuming the contrary in each direction. ❑

In Figure 7.9, if we disable δ at 000 and 010, and α at 100 and 101 then we see that all the states of A are L-restrictable.

Note that the compensator C is implemented in the following way: given the initial state, x, of A, we implement $A^{LA}_{K_I}$ starting from its initial state (x_0, x). Then, the compensator is simply the set-valued function of the present state of $A^{LA}_{K_I}$ given in Proposition 7.14.

Finally, the following result presents a straightforward construction for X_L, where $S_L = \{(x_0, x) \in X^{LA}\}$:

Proposition 7.15 $X_L = \{x \in X | (x_0, x) \in I(R(A^{LA}, S_L))\}$, and the complexity of this computation is $O(|X^{LA}|^2)$.

Proof: Straightforward. Since the complexity of computing $I(Q)$ is quadratic in the cardinality of the state space, the total complexity is $O(|X^{LA}|^2)$ (see Chapter 3). ❑

We can now relate our results to the notion of controllable languages of Wonham and Ramadge. We refer the reader to [39] for definitions. Specifically, let all events be tracking events, i.e., let $\Xi = \Sigma$, let L be the specified legal language, and let some $x \in X$ be the given initial state of A. Then:

Proposition 7.16 $L(A^{LA}_{K_I}, (x_0, x))$ is the supremal controllable sublanguage of the legal language L.

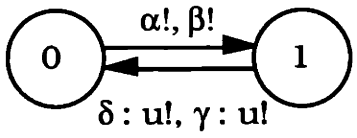

Figure 7.10: Example for Eventual Restrictability

Proof: This is straightforward to check from the definitions in [39] and the fact that K_I is minimally restrictive. ❑

As an example, if the initial state of the system in Figure 7.7(b) is 00, then the supremal controllable sublanguage of L is the language generated by state 000 in Figure 7.9 with δ disabled at 000 and 010, and α disabled at 100 and 101, as before. This compensator is also the same as the one computed in [50] for this example.

As a final comment, note that from the development in Section 7.2, one might expect that we would have presented results on "maximal" or "minimal" restrictable languages. These concepts, however, are trivial: the maximal language to which we can restrict behavior is obviously Ξ^*, while a number of minimal restrictable languages are possible: if $e(x) \neq \emptyset$, disable all controllable events at this state; if $e(x) = \emptyset$, disable all but one controllable event. Thus, in this context, it is more meaningful to fix L and look at the questions we have addressed here.

7.3.2 Eventual Restrictability

A natural generalization of our notion of restrictability is to consider restrictability in a finite number of transitions. For example, consider the system in Figure 7.10, where $\Xi = \Sigma$, and suppose that $L = (\alpha\gamma + \beta\delta)^*$. The automaton A'_L is illustrated in Figure 7.11, and the automaton A^{LA} is illustrated in Figure 7.12, where the transitions

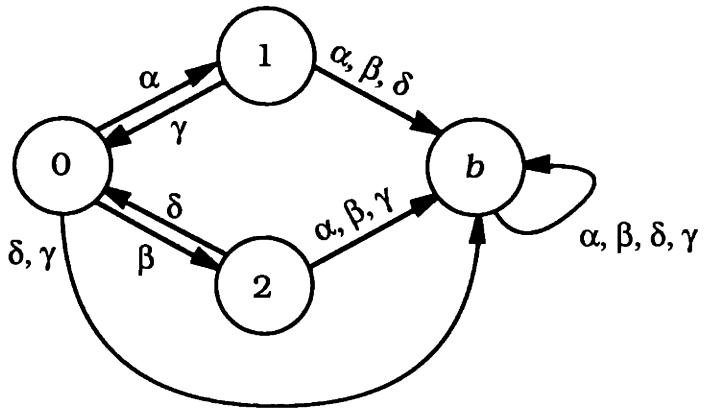

Figure 7.11: Automaton A'_L for $L = (\alpha\gamma + \beta\delta)^*$.

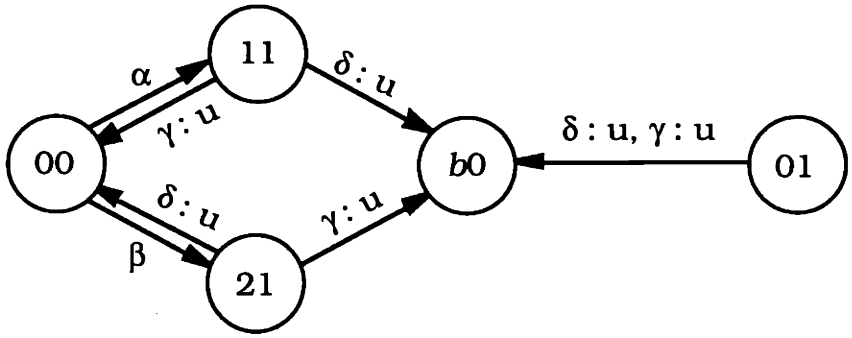

Figure 7.12: Composite of A and A'_L for the Eventual Restrictability Example

defined at state $b0$ have been ignored for simplicity. Note that 0 is L-restrictable, whereas 1 is not. However, if the system starts in state 1, the next transition takes state 1 to state 0 and the language generated from that point on can be restricted to L. We term this eventual restrictability:

Definition 7.17 Given $x \in X$ and a complete language L over Ξ, x is eventually L-restrictable if there exists an integer n_a and a compensator $C : X \times \Sigma^* \to U$ such that the closed loop system A_C is alive and $t(L(A_C, x)) \subset (\Xi \cup \{\epsilon\})^{n_a} L$. Given

$Q \subset X$, Q is eventually L-restrictable if all $x \in Q$ are eventually L-restrictable. Finally, A is eventually L-restrictable if X is eventually L-restrictable. ❑

Clearly, the class of eventually L-restrictable sets are closed under arbitrary unions and intersections. Let X_{EL} denote the maximal eventually L-restrictable set. Then, using our notion of stabilizability, X_{EL} is simply the maximal X_L-pre-stabilizable set. For the system in Figure 7.10, 1 is eventually L-restrictable since 0 is L-restrictable and 1 is pre-stable with respect to 0.

Proposition 7.18 X_{EL} is the maximal X_L-pre-stabilizable set and for all $x \in X_{EL}$, $n_a \leq r$. Furthermore, the complexity for this computation is $O(n^2)$.

Proof: The first part of this is straightforward. To show the bound on n_a, note that n_a is the maximum number of tracking events in any trajectory from an X_L -pre-stabilizable state to X_L. Since $|Y^t| = r$, if a trajectory from some state $x \in X$ has more than r tracking transitions before entering X_L, it has a cycle outside of X_L and therefore x is not pre-stabilizable. The complexity bound is direct from Chapter 3. ❑

A compensator for eventual restrictability can be constructed by using two compensators in tandem: The first one is a state feedback that pre-stabilizes A with respect to X_L. The second one is the compensator of Proposition 7.14 for restricting the language generated by x to L, where x is the element of X_L which the trajectory first visits.

7.3.3 Reliability

Our final generalization of restrictability is very similar to the notion of resiliency introduced in Chapter 6. Specifically, we allow a set of failure events and require that following a burst of failures, the system generates strings in L within a finite number

of transitions after the burst ends. In order to be consistent with our current framework, let us decompose Ξ into tracking events, Ξ_t, and failure events Ξ_f (instead of defining a new alphabet). A natural assumption is that no event in Ξ_f is controllable (since otherwise, we can just disable them). Given an integer $i \geq 1$, and $s \in L(A, x)$ for some $x \in X$, we say that s is a failure sequence with at most i failures if both the first and the last events of s are in Ξ_f, and at least one but at most i events of s are in Ξ_f. We define reliability as follows:

Definition 7.19 Given $x \in X$, a complete language L over Ξ_t, and an integer $i \geq 1$, x is i-reliably L-restrictable if x is eventually L-restrictable in $A|\overline{\Xi}_f$ and there exists a compensator $C : X \times \Sigma^* \to U$ such that the closed loop system $A_C|\overline{\Xi}_f$ is alive and for all failure sequences $s \in L(A_C, x)$ with at most i failures, $f(x, s)$ is eventually L-restrictable in $A_C|\overline{\Xi}_f$. Given $Q \subset X$, Q is i-reliably L-restrictable if all $x \in Q$ are i-reliably L-restrictable. A is i-reliably L-restrictable if X is i-reliably L-restrictable. ❑

Clearly, the class of i-reliably L-restrictable sets are closed under arbitrary unions and intersections. Let X_R^i denote the maximal i-reliably L-restrictable set, and let $X_R^\infty \triangleq \bigcap_{i=1}^\infty X_R^i$ denote the maximal set that is i-reliably L-restrictable for all integers i. Note that $X_R^0 = X_{EL}$, where X_{EL} is defined for $A|\overline{\Xi}_f$. The following result is immediate from the definitions:

Proposition 7.20 The sets X_R^i are nested, i.e.,

$$X_R^{i+1} \subset X_R^i$$

and if $X_R^{i+1} = X_R^i$, then $X_R^j = X_R^i$ for all $j \geq i$ including ∞. ❑

What remains is to describe a recursive procedure for computing X_R^i beginning from X_R^0. Let Y^i, for integers $i \geq 0$, denote the set of states x such that either no

failure events are defined at x or all the failure events take x to a state in X_R^i, i.e., if we let $d_f(x) = d(x) \cap \Xi_f$, then

$$Y^i = \{x \in X | d_f(x) = \emptyset \text{ or for all } \sigma \in d_f(y), f(y,\sigma) \subset X_R^i\} \tag{7.21}$$

Note that $Y^{i+1} \subset Y^i$.

Consider then what it means for a state $x \in X$ to be 1-reliably L-restrictable. First, we must have that $x \in X_R^0$. Secondly, what we must have is that any state that can be reached from x with one failure event must be eventually L-restrictable with failure events turned off. To be precise, define

$$L_1(A,x) = \{s \in L(A,x) | \text{ only the last element of } s \text{ in } \Xi_f\} \tag{7.22}$$

Thus, $L_1(A,x)$ are the possible event trajectories leading up to and including the first failure when we start in x. Then we must have for any $s \in L_1(A,x)$ that $f(x,s) \subset X_R^0$. Note that this implies that all of the states along any trajectory from x to $f(x,s)$ must lie completely in Y_0. Thus let X^0 denote the maximal sustainably (f,u)-invariant set in Y^0 and let K^0 denote the associated feedback. Then we have

Lemma 7.21 $X_R^1 = R^0$, the maximal eventually L-restrictable subset of X^0 in $A_{K^0}|\overline{\Xi}_f$.

Proof: That X_R^1 is contained in R^0 is clear from the preceding argument. To show the opposite inclusion, take any $x \in R^0$. Then, we can find a compensator such that with x as initial state only strings in L are allowed in the closed loop system (with failure events turned off) and the trajectories stay in Y^0. Then, following a failure event, the system can only make a transition to a state y that is eventually restrictable, and thus we can restrict the language generated from y to L within a finite number of transitions. Therefore, x is 1-reliably L-restrictable. ❑

Note that from the argument preceding the lemma statement, one might conclude that X_R^1 is simply $X_R^0 \cap X^0$. However, in making X^0 invariant, we have applied a feedback K^0, and this may then restrict what further feedback can be applied to achieve eventual restrictability. Thus, in general, X_R^1 may be smaller than $X_R^0 \cap X^0$.

Continuing with our construction, let X^i denote the maximal sustainable (f,u)-invariant subset in Y^i of $A|\overline{\Xi}_f$ and let K^i be the associated state feedback. Note that because of the nesting of the Y^i, K^i is compatible with K^{i-1} (i.e., any event disabled by K^{i-1} is also disabled by K^i). We then have the following:

Proposition 7.22 X_R^{i+1} is the maximal eventually L-restrictable subset of X^i in $A_{K^i}|\overline{\Xi}_f$.

Proof: Similar to the proof of Lemma 7.21. ❑

Thus the full recursive procedure is: (1) compute X_R^0 using the eventual restrictability results of the preceding subsection applied to $A|\overline{\Xi}_f$; (2) given X_R^i compute Y^i from (7.21); (3) compute X^i and K^i using the (f,u)-invariance results discussed in Chapter 3; (4) compute X_R^{i+1} using the eventual restrictability results applied to $A_{K^i}|\overline{\Xi}_f$. Also, as a byproduct of the above construction we obtain the following:

Corollary 7.23 $X_R^\infty = X_R^i$ for some $i \leq |Y'|$ where Y' is the set of states x such that $d_f(x) \neq \emptyset$. ❑

Thus, we can compute X_R^∞ in a finite number of steps, and in fact, the complexity of this computation is $O(|Y'||X^{LA}|^2)$.

As an example consider the system in Figure 7.13, where $\Xi_t = \{\alpha, \beta, \delta, \gamma\}$, $\Phi = \emptyset$, and $\Xi_f = \{\phi\}$. Let $L = \beta\alpha^*$, then $X_R^0 = \{0,2\}$ and $Y^0 = \{0,1,3\}$. Thus, $X^0 = Y^0$ and K^0 is a trivial feedback that enables all events. Finally, $X_R^1 = \{0\}$. Thus, state 0 can recover from a single failure, whereas state 2 cannot. Specifically, if the failure ϕ occurs at state 2, then the current state makes a transition to state 3 which is not

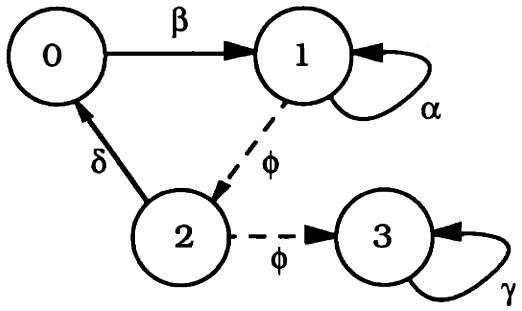

Figure 7.13: Reliable Restrictability Example: $\Xi_t = \{\alpha, \beta, \delta, \gamma\}$, $\Phi = \emptyset$, $\Xi_f = \{\phi\}$.

eventually L-restrictable. Continuing this procedure, we get $X_R^2 = X_R^3 = \cdots = \emptyset$. Thus, state 0 cannot recover from 2 or more failures.

7.4 Discussion

In this chapter, we have introduced notions of tracking, restrictability, and reliability for discrete-event systems described by finite-state automata. We have developed algorithms for constructing trackable languages, testing restrictability and reliability, and constructing compensators for reliable restriction of system behavior.

This work complements our stability analysis in Chapter 3 in that the notions of eventual trackability and eventual restrictability lead to particular choices for the set E that we use for stability. In the case of partial observations, our results in this chapter can be combined with our results on stabilization by output feedback in Chapter 5 to address problems of tracking and restrictability in the context of intermittent observations of events (see the next chapter). As we have shown in Chapters 4 and 5, problems of stabilization by output feedback have polynomial complexity if the observer state space is also polynomial. Since our conditions for restrictability

are also based on stabilizability and since we have seen how to place the problem of controllable languages of Wonham and Ramadge in our framework, we see that a source of the NP-completeness of this problem in the case of partial observations, is the cardinality of the observer state space. Thus, if in fact the observer has polynomial state space, then the problem of controllable languages for the case of partial observations can also be solved in polynomial time.

Another major problem of computational complexity in DEDS arises in the case of interacting automata. If, for example, we have m interconnected sub-systems each with n states then their composition may have n^m states. We approach this problem, in the next chapter, by formulating a notion of aggregation based on a given set of tasks, and thus reducing the state space of each sub-system (the number n) considerably.

Chapter 8
Multi-Level Control

8.1 Motivation and Background

A major issue of concern in the work of Wonham and Ramadge, as well as other work on DEDS is that of computational complexity, and the goal of this chapter is to address certain issues of complexity. In particular, in many applications the desired range of behavior of a DEDS is significantly smaller and more structured than its full range of possible behaviors. For example, a workstation in a flexible manufacturing system may have considerable flexibility in the sequence of operations it performs. However, only particular sequences correspond to useful tasks. This idea underlies the notion of a legal language introduced in [39]. In the analysis described in this chapter we use it as well to develop a method for higher-level modelling and control in which a sequence of events corresponding to a task is mapped to a single macro-event at the higher-level. Also, in DEDS described as interconnections of subsystems the overall state space for the entire system can be enormous. However, in many applications, such as a flexible manufacturing system consisting of interconnections of workstations, the desired coordination of the subsystems is at the task level, and thus we can consider the interactions of their individual aggregate models. These higher-level characterizations allow us to represent sets of states by a single state and sets of strings by a single event. We thus achieve both spatial and temporal

aggregation that can greatly reduce the apparent computational explosion arising in the analysis of extremely complex systems.

The work described in this chapter builds on all the previous chapters and can be viewed as the culmination of an effort to develop a regulator theory for DEDS. As we will see, our development will involve controlling the system so that its behavior is restricted to completing the desired tasks. To address this, we will rely on the notions of tracking and restrictability that we developed in Chapter 7. The latter of these is closely related to the notion of constraining behavior to a legal language. However, by describing the desired behavior in terms of primitive tasks, we achieve significant efficiencies, and through the use of the notion of eventual restrictability we are able to directly accommodate the phenomenon of set-up, i.e., the externally irrelevant transient behavior arising when one switches between tasks.

We will see that many of the components of our work are relevant here. In particular, our notion of stability in Chapter 3, i.e., of driving the system to a specified set of states is central to most of our constructions. Furthermore, since we assume a model in which only some events are observed, we will need to make use of our results in observability in Chapter 4 and output stabilization in Chapter 5. Finally, in order to derive an upper-level model, it will be necessary to be able to use our observations to reconstruct the sequence of tasks that has been performed. This is closely related to the problem of invertibility stated in Chapter 6.

In this chapter, we use the full model introduced in Chapter 2, but with the assumption (as in Chapter 5) that an event controllable at a state is controllable at all other states, i.e., we assume that $\Phi \subset \Sigma$ characterizes the controllable events. Specifically, the systems we consider in this chapter are of the form $A = (G, f, d, h, t)$ where $G = (X, \Sigma, \Phi, \Gamma, \Xi)$. We also assume that $\Phi \subset \Gamma$, as we did in Chapter 5.

As commented in the previous chapter, it is straightforward to extend the notions of the previous chapter in the context of partial observations of event trajectories. To illustrate this, we now turn our attention to a notion of eventual restrictability by output feedback, which plays an important role in the development of this chapter:

Definition 8.1 Given a complete language L over Ξ we say that A is eventually L-restrictable by output feedback if there exists an integer n_o and an output compensator $C : \Gamma^* \to U$ such that A_C is alive and for all $x \in X$, $t(L(A_C, x)) \subset (\Xi \cup \{\epsilon\})^{n_o} L$. Such a C is called an L-restrictability compensator. ❑

We construct a test for eventual restrictability by output feedback as follows: Given L, let (A_L, x_0^L) be a minimal recognizer for L and let Z_L denote its state space. Let A'_L be an automaton which is the same as A_L except that its state space is $Z'_L = Z_L \cup \{b\}$ where b is a state used to signify that the event trajectory is no longer in L. This is the state we wish to avoid. Also, we let $d'_L(x) = \Xi$ for all $x \in Z'_L$, and

$$f'_L(x,\sigma) = \begin{cases} f_L(x,\sigma) & \text{if } x \neq b \text{ and } \sigma \in d_L(x) \\ \{b\} & \text{otherwise} \end{cases} \tag{8.1}$$

Let O denote the observer for A, let $A(L) = A \parallel A'_L$, and let $O(L) = (G(L), w_L, v_L)$ denote the observer for $A(L)$; however, in this case, since we know that we will start A'_L in x_0^L, we take the state space of $O(L)$ as

$$Z(L) = R(O(L), \{\{x_0^L\} \times \hat{x} | \hat{x} \in Z\}) \tag{8.2}$$

Let

$$V_o = \{\hat{z} \in Z(L) | \text{ for all } (x_L, x_A) \in \hat{z}, x_L \neq b\} \tag{8.3}$$

Let $E(L)$ be the largest subset of V_o which is sustainably (f,u)-invariant in $O(L)$ and for which the associated unique minimally restrictive feedback K^{EL} has the property

that for any $\hat{z} \in Z(L)$, $K^{EL}(\hat{z})$ is $\hat{x}(\hat{z})$-compatible where

$$\hat{x}(\hat{z}) = \{x \in X | \exists x_L \in Z_L \text{ such that } (x_L, x) \in \hat{z}\} \tag{8.4}$$

The construction of $E(L)$ and K^{EL} is a slight variation of the algorithm in Chapter 3 for the construction of maximal sustainably (f,u)-invariant subsets. Specifically, we begin with any state $\hat{z} \in V_o$. If there are any uncontrollable events taking $\hat{z}$ outside V_o, we delete $\hat{z}$ and work with $V_1 = V_o \setminus \{\hat{z}\}$. If not, we disable only those controllable events which take $\hat{z}$ outside V_o. If the remaining set of events defined at $\hat{z}$ is not $\hat{x}(\hat{z})$-compatible, we delete $\hat{z}$ and work with $V_1 = V_o \setminus \{\hat{z}\}$. If not, we tentatively keep $\hat{z}$ and choose another element of V_o. In this way, we continue to cycle through the remaining elements of V_o. The algorithm converges in a finite number of steps (at most $|V_o|^2|$) to yield $E(L)$ and K^{EL} defined on $E(L)$. For $\hat{z} \in \overline{E(L)}$, we take $K^{EL}(\hat{z}) = \Sigma$ so that no events are disabled.

Consider next the following subset of $E(L)$:

$$E_o(L) = \{\hat{x} \in Z | x_0^L \times \hat{x} \in E(L)\} \tag{8.5}$$

Then,

Proposition 8.2 Given a complete language L over Ξ, A is eventually L-restrictable by output feedback iff there exists an A-compatible state feedback $K : Z \to U$ such that the closed loop system O_K is $E_o(L)$-pre-stable.

Proof: ($\rightarrow$) Straightforward by assuming the contrary.

($\leftarrow$) Let us prove this by constructing the desired compensator $C : \Gamma^* \to U$: Given an observation sequence s, we trace it in O starting from the initial state $\{Y\}$. Let $\hat{x}$ be the current state of O given s. There are two possibilities:

1. Suppose that the trajectory has not yet entered $E_o(L)$. Then we use O and the

$E_0(L)$-pre-stabilizing feedback K to compute $C(s)$. In particular,

$$C(s) = (v(\hat{x}) \cap K(\hat{x})) \cup (v(\hat{x}) \cap \overline{\Phi})$$

2. When the trajectory in O enters $E_o(L)$, we switch to using the expanded observer $O(L)$ and $K^{E(L)}$. In particular, let $\hat{x}'$ be the state the trajectory in O enters when it enters $E_o(L)$ for the first time, and let s' be that prefix of s which takes $\{Y\}$ to $\hat{x}'$ in O. Then, we start O_L at the state $x_0^L \times \hat{x}' \in E(L)$, and let it evolve. Suppose that s/s' takes $x_0^L \times \hat{x}'$ to $\hat{z}$ in $O(L)$, then

$$C(s) = (v_L(\hat{z}) \cap K^{E(L)}(\hat{z})) \cup (v_L(\hat{z}) \cap \overline{\Phi})$$

Since this feedback keeps the trajectory of O in $E(L)$ and $E(L) \subset V_o$, the behavior of A is restricted as desired. ❑

Note that since $E(L)$ is the maximal sustainably (f,u)-invariant subset of V_o and $K^{E(L)}$ is unique, the possible behavior of an L-restrictable state x in the closed loop system constructed in the proof is the maximal subset of L to which the behavior of x can be restricted. Note also that if $E_o = \emptyset$, then O cannot be $E_o(L)$-pre-stabilizable and thus A cannot be eventually L-restrictable by output feedback. Finally, if A is eventually L-restrictable by output feedback, then the number of observable transitions until the trajectory is restricted to L is at most n_s.

8.2 Characterizing Higher-Level Models

In this section, we present a notion of higher-level modelling of DEDS based on a given set of primitives, each of which consists of a finite set of tracking event strings. The idea here is that the occurrence of any string in this set corresponds to some macroscopic event, such as completion of a task, and it is only these macro-events

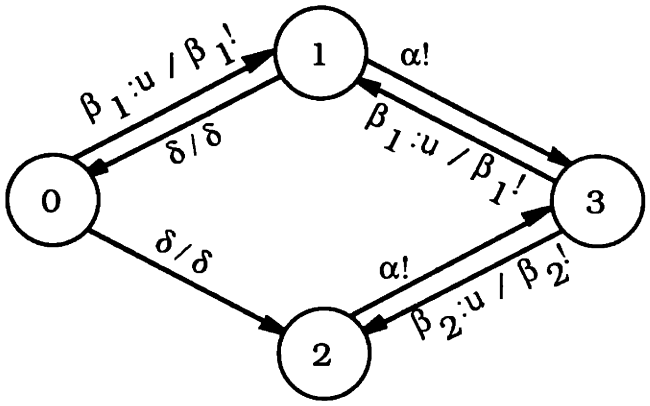

Figure 8.1: A Simple Example

that we wish to model at the higher level. Our modelling concept therefore must address the issues of controlling a DEDS such that its behavior is restricted to these primitives and of being able to observe the occurrences of each primitive. In this section we describe precisely what it means for one DEDS to serve as a higher-level model of another. In subsequent sections we explicitly consider the notion of tasks and the problems of controlling and observing them and the related concept of procedures, defined in terms of sequences of tasks, which allows us to describe higher-level models of interconnections of DEDS, each of which can perform its own set of tasks.

To illustrate our notion of modelling, consider the system in Figure 8.1 and suppose that we wish to restrict its behavior to $(\alpha\beta_1)^*$ by output feedback. In order to do this, we first specify $L_1 = \alpha\beta_1$ as a primitive. For this example it is possible, essentially by inspection, to construct an L_1^{*c}-restrictability compensator $C : \Gamma^* \to U$ that is simpler than the one given in the proof of Proposition 8.2. Specifically, we can take C to be a function of the state of an automaton, illustrated in Figure 8.2, constructed from the observer for the original system simply by deleting the events which the compensator disables. The initial state of this system, as in the observer,

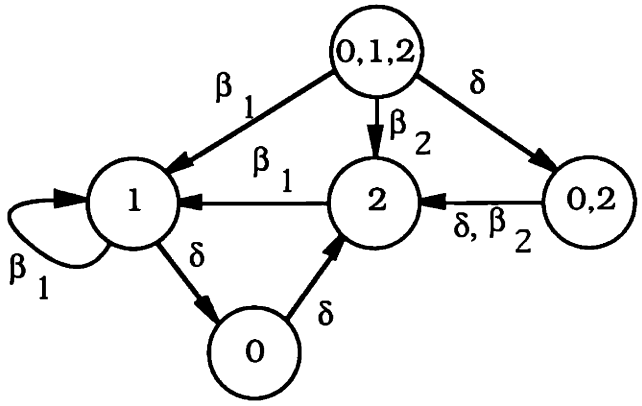

Figure 8.2: Illustrating the Compensator for Eventual L_1^{*c}-Restrictability by Output Feedback

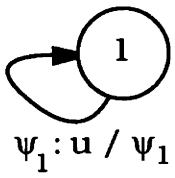

Figure 8.3: Model of Task 1

is (0,1,2), and for any string s the compensator value $C(s)$ is a function of the state of the modified observer. For example, if the first observed event is δ, the state of the system in Figure 8.2 is (1,2). In the original DEDS of Figure 8.1 the event β_1 would be possible from state 0; however, as illustrated in Figure 8.2, we disable β_1 so that the next observable event will either be δ (from 0) or β_2 (from 3 after the occurrence of α from 2). It is not difficult to check that the closed loop system A_C is eventually L_1^{*c}-restricted, and thus the language eventually generated by A_C can be modelled at a higher level by the automaton in Figure 8.3 for which ψ_1 represents an occurrence of $\alpha\beta_1$.

In order for this automaton, with ψ_1 observable, to truly model A_C, it must be true that we can in fact detect every occurrence of $\alpha\beta_1$ in A_C, perhaps with some initial uncertainty, given the string of observations of A_C. For example, by inspection of Figure 8.2, if we observe β_1, we cannot say if $\alpha\beta_1$ has occurred or not, but if we observe $\beta_1\beta_1$, we know that $\alpha\beta_1$ must have occurred at least once. Likewise, $\beta_2\beta_1$ corresponds to one occurrence of $\alpha\beta_1$, $\delta\beta_2\beta_1\delta\delta\beta_1$ corresponds to two occurrences of $\alpha\beta_1$, etc. In general, after perhaps the first occurrence of β_1, every occurrence of β_1 corresponds to an occurrence of $\alpha\beta_1$ and therefore, we can detect occurrences of ψ_1 from observations in A_C. The definition we give in this section will then allow us to conclude that the automaton in Figure 8.3 models the closed loop system A_C.

To begin our precise specification of higher-level models, let us first introduce a function that defines the set of strings that corresponds to a primitive: Given alphabets Σ' and Ξ, and a map $H_e : \Sigma' \to 2^{\Xi^*}$, if for all $\sigma \in \Sigma'$ $H_e(\sigma)$ is a collection of finite length strings, then we term H_e a primitive map. Here $\sigma \in \Sigma'$ is the macroscopic event corresponding to the set of tracking strings $H(\sigma)$ in the original model. We allow the possibility that several strings may correspond to one macroscopic primitive to capture the fact that there may be several ways to complete a desired task.

We will require the following property:

Definition 8.3 A primitive map H_e is termed minimal if for all, not necessarily distinct, $\sigma_1, \sigma_2 \in \Sigma'$ and for all $s \in H_e(\sigma_1)$, no proper suffix of s is in $H_e(\sigma_2)$. ❑

Given a primitive map, we extend it to act on strings over Σ' as follows: We let $H_e(\epsilon) = \epsilon$ and $H_e(s\sigma) = H_e(s)H_e(\sigma)$, where s is a string over Σ' and σ is an element of Σ'. Also, $H_e(s)H_e(\sigma)$ is the set of strings consisting of all possible concatenations of a string in $H_e(s)$ followed by a string in $H_e(\sigma)$. We use the same symbol to denote H_e and its extension to $(\Sigma')^*$.

Proposition 8.4 If H_e is minimal then for all distinct r_1, r_2 such that $r_1, r_2 \neq \epsilon$, $|r_1| \leq |r_2|$, and r_1 is not a suffix of r_2, $\Xi^* H_e(r_1) \cap \Xi^* H_e(r_2) = \emptyset$.

Proof: Assume the contrary, and let $s \in \Xi^* H_e(r_1) \cap \Xi^* H_e(r_2)$. Also let σ_1 (respectively, σ_2) be the last event in r_1 (respectively, r_2). There are two cases here: First, suppose that $\sigma_1 \neq \sigma_2$. Then, there exist distinct $p_1 \in H_e(\sigma_1)$ and $p_2 \in H_e(\sigma_2)$ such that both p_1 and p_2 are suffixes of s. Assume, without loss of generality, that $|p_1| \leq |p_2|$, then p_1 is also a suffix of p_2. But then, H_e cannot be minimal. Now, suppose that $\sigma_1 = \sigma_2$. Thanks to minimality, among all elements of $H_e(\sigma_1)$, only one string, say p can be a suffix of s. Let s' be that prefix of s such that $p = s/s'$. Then, repeat the previous steps using s', and all but the last elements of r_1 and r_2. Since r_1 and r_2 are distinct, and r_1 is not a suffix of r_2, σ_1 will be different from σ_2 at some step and then we will establish a contradiction. Therefore, $\Xi^* H_e(r_1) \cap \Xi^* H_e(r_2) = \emptyset$. ❑

The following result states that we can concatenate minimal primitive maps while preserving minimality:

Proposition 8.5 Given minimal $H_1 : \Sigma_2 \rightarrow 2^{\Sigma_1^*}$ and $H_2 : \Sigma_3 \rightarrow 2^{\Sigma_2^*}$, if we define $H_3 : \Sigma_3 \rightarrow 2^{\Sigma_1^*}$ so that $H_3(\sigma) = H_1(H_2(\sigma))$ for all $\sigma \in \Sigma_3$, then H_3 is a minimal primitive map. Here, since $H_2(\sigma)$ is a set of strings, $H_1(H_2(\sigma))$ is the set of strings resulting from applying H_1 to each string in $H_2(\sigma)$.

Proof: Assuming the contrary, there exists, $\sigma_1, \sigma_2 \in \Sigma_3$, $s \in H_3(\sigma_1)$, and a suffix r of s so that $r \in H_3(\sigma_2)$. Let $s' \in H_2(\sigma_1)$ and $r' \in H_2(\sigma_2)$ such that $s \in H_1(s')$ and $r \in H_1(r')$. Then, by minimality of H_2, r' cannot be a suffix of s' and s' cannot be a suffix of r' either. Also, since r is a suffix of s, $s \in \Sigma_1^* H_1(r')$. Then, thanks to Proposition 8.4, H_1 cannot be minimal, and we establish a contradiction. Therefore, H_3 must be minimal. ❑

Now, let us proceed with defining our notion of modelling. In particular, given

two automata $A = (G, f, d, h, t)$ and $A' = (G', f', d', h', t')$, we wish to specify when A' is an H_e-model of the system A, where $H_e : \Sigma' \to \Xi$ is a minimal primitive map. Two important properties that we will require of our models are the following:

1. **Restrictability:** We will require that if we can restrict the behavior of the macroscopic model to some complete language $L \subset \Sigma'^*$, then we can also restrict the original system to $H_e(L)^c$. Note that we have defined L over Σ', instead of Ξ'. Roughly speaking, we have done this since if all languages of interest are over the higher-level tracking alphabet Ξ', then we can perhaps choose a simpler macroscopic model completely over Ξ'. However, the alphabet Ξ' will still be useful in defining different levels of modelling (see Proposition 8.8).

2. **Detectability** We will also require that for any lower-level string s in $L(A)$ such that $t(s)$ is in $H_e(p)$ for some string p in the macroscopic system,

   (a) we can reconstruct p, after some delay, using the lower-level observation $h(s)$ of s, and

   (b) for any string r so that s is a suffix of r, the reconstruction acting on $h(r)$ results in a string that ends with the reconstruction of $h(s)$.

   Note that minimality implies the following: If we let $H_e^{-1}(t(s))$ denote the set of strings $p \in \Sigma'^*$ such that $t(s) \in H_e(p)$, then, thanks to minimality, $H_e^{-1}(t(s))$ is single valued. Thus, in order to satisfy the first condition of detectability, we need to be able to reconstruct $H_e^{-1}(t(s))$ from $h(s)$. The second condition deals with the issue of start-up. Specifically, in our framework of eventual restrictability, we allow for the possibility of a transient start-up period in which the lower-level may generate a short tracking event sequence that does not correspond to any primitive. What (b) requires is that the reconstruction can

recognize and "reject" such finite length start-up strings.

Definition 8.6 Given two DEDS A and A', and a minimal primitive map $H_e : \Sigma' \to 2^{\Xi^*}$, we say that A' is an H_e-model of A if there exists a map $H_o : \Gamma^* \to \Sigma'^*$ and an integer n_d such that[1]:

1. **Restrictability:** For all complete $L \subset \Sigma'^*$ such that A' is eventually L-restrictable, A is eventually $H_e(L)^c$-restrictable by output feedback.
2. **Detectability:** For all $s \in L(A)$, such that $t(s) \in H_e(p)$ for some $p \in L(A')$,
   (a) $p \in (\Sigma' \cup \{\epsilon\})^{n_d} H_o(h(s))$, and
   (b) for all $r \in \Sigma^* s$, $H_o(h(r)) \in \Sigma'^* H_o(h(s))$.

❑

Note that this concept of modelling provides a method of both spatial and temporal aggregation, as we will see, since A' may frequently be constructed to have many fewer states than A and sets of strings in A can be represented by a single event in A'. For example, all states in Figure 8.1 are represented by a single state in Figure 8.3, and $\alpha\beta_1$ is represented by ψ_1.

The following result, which immediately follows from Definition 8.6, states that the concept of modelling is invariant under compensation:

Proposition 8.7 If A' is an H_e-model of A then for any compensator $C' : \Gamma'^* \to U'$ for A', there exists a compensator $C : \Gamma^* \to U$ for A such that $A'_{C'}$ is an H_e-model of A_C with the same H_o. ❑

[1] We have chosen in our definition to look at the larger class of macroscopic languages to which A is eventually restrictable by full state feedback, rather than only with output feedback. All of our results carry over if we use this weaker notion of restrictability at the higher level. Similarly in our definition of detectability we have required the stronger condition that from lower level observations, we can reconstruct the entire upper-level event trajectory, not just the part in Γ'. Again, we can carry all of our development over to the weaker case. As we will see, this stronger definition suffices for our purposes

In general, we may be interested in several different levels of aggregation. Thus, we need the following result which states that a higher-level model also models automata at all lower levels:

Proposition 8.8 Given the automata $A = (G, f, d, h, t)$, $A' = (G', f', d', h', t')$, and $A'' = (G'', f'', d'', h'', t'')$, and minimal primitive maps $H'_e : \Sigma' \to 2^{\Xi^*}$ and $H''_e : \Sigma'' \to 2^{\Xi'^*}$, so that A' is an H'_e-model of A with H'_o and A'' is an H''_e-model of A' with H''_o, define $\pi : \Xi' \to 2^{\Sigma'^*}$ so that $\pi(\sigma) = \sigma(\overline{\Xi}' \cup \{\epsilon\})^{|X'|}$ for $\sigma \in \Xi'$ and define $H_e : \Sigma'' \to 2^{\Xi^*}$ as $H_e(\sigma) = H'_e(\pi(H''_e(\sigma)))$ for $\sigma \in \Sigma''$. Then,

1. H_e is a minimal primitive map.
2. A'' is an H_e-model of A with $H_o(s) = H''_o(h'(H'_o(s)))$ for all $s \in \Gamma^*$.

Proof: 1. Clearly, π is a minimal primitive map. Then, by Proposition 8.5, H_e is also a minimal primitive map.

2. Restrictability: If A'' is eventually L-restrictable, then A' is eventually $H''_e(L)^c$-restrictable by output feedback $\to$ A' is eventually $H''_e(L)^c$-restrictable $\to$ A' is eventually $\pi(H''_e(L)^c)$-restrictable $\to$ A is eventually $H_e(L)^c$-restrictable by output feedback.

Detectability: Straightforward. ❑

8.3 Aggregation

In this section, we use the concept of modelling of the previous section to present an approach for the aggregation of DEDS. What we have in mind is the following paradigm. Suppose that our system is capable of performing a set of tasks, each of which is a primitive as defined in the previous section. What we would like to do is

to design a compensator that accepts as inputs requests to perform particular tasks and then controls A so that the appropriate task is performed. Assuming that the completion of this task is detected, we can construct a higher level and extremely simple standard model for our controlled system: tasks are requested and completed. Such a model can then be used as a building block for more complex interconnections of task-oriented automata and as the basis for the closed loop following of a desired task schedule.

In the first subsection we define tasks and several critical properties of sets of tasks and their compensators. Roughly speaking we would like tasks to be uniquely identifiable segments of behavior of A that in addition do not happen "by accident" during task set-up. More precisely, we introduce the notion of independence of tasks which states that no task is a subtask of another (so that all tasks describe behavior at roughly the same level of granularity) and the notion of a consistent compensator, which, while setting up to perform a desired task, ensures that no other task is completed. With such a set of tasks and compensators we can be assured that a desired task sequence can be followed, with task completions separated by at most short set-up periods. In Section 8.3.2 we discuss the property of task observability, i.e., the ability to detect all occurrences of specified tasks. In Section 8.3.3 we then put these pieces together to construct a special higher-level model which we refer to as task standard form.

8.3.1 Reachable Tasks

Our model of a task is a finite set of finite length strings, where the generation of any string in the set corresponds to the completion of the task. Let T be the index set of a collection of tasks, i.e., for any $i \in \mathsf{T}$ there is a finite set L_i of finite length strings

over Ξ that represents task i. In our development we will need a similar but stronger notion than that of minimality used in the previous section. We let $L_T = \cup_{i \in \mathsf{T}} L_i$ and define the following:

Definition 8.9 Given T, we say that T is an independent task set if for all $s \in L_T$, no substring of s, except for itself, is in L_T. ❑

Then when we look at a tracking sequence there is no ambiguity concerning what tasks have been completed and which substring corresponds to which task. Note that if T is an independent set, then the minimal recognizer (A_T, x_0) for all of L_T has a single final state x_f, i.e., all strings in L_T take x_0 to x_f, and x_f has no events defined from it (since L_T is a finite set). Furthermore, for each $i \in \mathsf{T}$, the minimal recognizer $(A_{L_i}, x_0^{L_i})$ also has a single final state $x_f^{L_i}$ which has no events defined from it.

Let us define a second task $L_2 = \alpha\beta_2$ in addition to the task $L_1 = \alpha\beta_1$ for the example in Figure 8.1. Note that this system is in fact eventually L_1^{*c} and L_2^{*c}-restrictable. We term such tasks reachable:

Definition 8.10 A task $i \in \mathsf{T}$ is reachable if A is eventually L_i^{*c}-restrictable. T is a reachable set if each $i \in \mathsf{T}$ is reachable. ❑

Definition 8.11 Task $i \in \mathsf{T}$ is reachable by output feedback if A is eventually L_i^{*c}-restrictable by output feedback. T is reachable by output feedback if each $i \in \mathsf{T}$ is reachable by output feedback. ❑

Given a task $i \in \mathsf{T}$ that is reachable by output feedback, let $C_i : \Gamma^* \to U$ be an L_i^{*c}-restrictability compensator. Consider the possible behavior when we implement this compensator. Note that states in $E_o(L_i^{*c})$, as defined in Section 8.1, are guaranteed to generate a sublanguage of L_i^{*c} in the closed loop system. However, for any other

state $\hat{x} \in Z$, although we cannot guarantee that L_i^{*c} will be generated given the particular knowledge of the current state of the system (i.e., given that the system is in some state in $\hat{x}$), it may still be possible for such a string to occur. Furthermore, in general, a string in L_j, for some other j, may be generated from a state $x \in \hat{x}$ before the trajectory in O reaches $E_o(L_i^{*c})$. If in fact a string in L_j is generated from some $x \in \hat{x}$, then task j will have been completed while the compensator was trying to set-up the system for task i. Since this is a mismatch between what the compensator is trying to accomplish and what is actually happening in the system, we will require that it cannot happen. We define this property as follows (we state the definition for recurrent observer states, allowing for mismatch for a bounded number of transitions at the overall start-up of the system):

Definition 8.12 Given a reachable task $i \in \mathbf{T}$ and an L_i^{*c}-restrictability compensator C_i, C_i is consistent with $\mathbf{T}$ if for all $\hat{x} \in Z_r \cap \overline{E_o(L_i^{*c})}$, for all $x \in \hat{x}$, and for all $s \in L(A_{C_i}, x)$, $t(s) \notin L_T$. ❑

Now, let us consider testing the existence of and constructing consistent restrictability compensators. Note that we only need to worry about forcing the trajectory in O into $E_o(L_i^{*c})$ without completing any task along the way. Once that is done, restricting the behavior can be achieved by the compensator defined in the proof of Proposition 8.2. First, we need a mechanism to recognize that a task is completed. Thus, let (A_T, x_0) be a minimal recognizer for L_T with the final state x_f. Let X_T be the state space of A_T. Since not all events are defined at all states in X_T, we do the following: we add a new state, say state g to the state space of A_T, and for each event that is not previously defined at states in X_T we define a transition to state g. Thus if A_T enters state g, we know that the tracking event sequence generated starting from x_0 and ending in g is not the prefix of any task sequence. Also, to keep the automaton

alive, we define self-loops for all events in Ξ at states g and x_f. Let A'_T be this new automaton. Given a string s over Ξ, if s takes x_0 to g in A'_T then no prefix of s can be in L_T. If, on the other hand, the string takes x_0 to x_f then some prefix of this string must be in L_T. Now, let $O' = (G', w', v')$ be the observer for $A \parallel A'_T$. We let the state space Z' of O' be the range of initial states

$$Z'_0 = \{\hat{x} \times \{x_0\} | \hat{x} \subset Z_r\} \tag{8.6}$$

i.e., $Z' = R(O', Z_0)$. Let $p : Z' \rightarrow Z_r$ be the projection of Z' into Z_r, i.e., given $\hat{z} \in Z'$, $p(\hat{z}) = \bigcup_{(x_1,x_2)\in\hat{z}}\{x_1\}$. Also, let $E'_o = \{\hat{z} \in Z' | p(\hat{z}) \in E_o(L_i^{*c})\}$. Our goal is to reach E'_o from the initial states Z'_0 while avoiding the completion of any task. Once the trajectory arrives at E'_o further behavior can be restricted as desired. So, we remove all transitions from states in E'_o and instead create self loops in order to preserve liveness. Let $O'' = (G', w'', v'')$ represent the modified automaton. Let us now consider the set of states in which we need to keep the trajectory. These are the states that cannot correspond to a completion of any task. Thus, we need to keep the trajectories in the set

$$E'' = \{\hat{z} \in Z' | \forall (x_1, x_2) \in \hat{z}, x_2 \neq x_f\} \tag{8.7}$$

Let V' be the maximal (f,u)-invariant subset of E', and let $K^{V'}$ be the corresponding A-compatible and minimally restrictive feedback. In order for a consistent compensator to exist, Z'_0 must be a subset of V'. Assuming this to be the case, we need to steer the trajectories to E'_o while keeping them in V'. Therefore, we need to find a feedback $K'' : Z' \rightarrow U$ so that Z' is E'_o-pre-stable in $O''_{K^{V'}}$ and so that the combined feedback $K : Z' \rightarrow U$ defined by

$$K(\hat{z}) = K^{V'}(\hat{z}) \cap K''(\hat{z}) \tag{8.8}$$

for all $\hat{z} \in Z'$ is A-compatible. The construction of such a K, if it exists, proceeds much as in our previous construction in Section 8.1. Thanks to the uniqueness of $K^{V'}$, if we cannot find such a feedback, then a consistent restrictability compensator cannot exist. In the analysis in this chapter, we assume that consistent compensators exist. That is, we assume that for each task $Z'_0 \subset V'$ and K exists.

Finally, let us outline how we put the various pieces together to construct a consistent compensator C_i for task i: Given an observation sequence s, we trace it in O starting from the initial state $\{Y\}$. Let $\hat{x}$ be the current state of O given s. There are three possibilities:

1. Suppose that $\hat{x} \notin Z_r$ and the trajectory has not entered $E_o(L_i^{*c})$ yet. Then, we use O and an $E_o(L_i^{*c})$-pre-stabilizing feedback to construct $C_i(s)$ as explained in the proof of Proposition 8.2.

2. Suppose that $\hat{x} \in Z_r$ and the trajectory has not entered $E_o(L_i^{*c})$ yet. Then, we use the observer O'' and the feedback K defined above. In particular, let $\hat{x}'$ be the state in the observer O into which the trajectory moves when it enters Z_r for the first time, and let s' be that prefix of s which takes $\{Y\}$ to $\hat{x}'$ in O. Then, we start O'' at state $\hat{x}' \times x_o$ and let it evolve. Suppose that s/s' takes $\hat{x}' \times x_0$ to $\hat{z}$ in O'' then

$$C_i(s) = (v''(\hat{z}) \cap K(\hat{z})) \cup (v''(\hat{z}) \cap \overline{\Phi}) \tag{8.9}$$

3. When the trajectory enters $E_o(L_i^{*c})$, we switch to using $O(L_i^{*c})$ and the (f,u)-invariance feedback $K^{L_i^{*c}}$. $C_i(s)$ in this case can be constructed as explained in the proof of Proposition 8.2.

In order to develop a complete higher-level modelling methodology, we need to describe explicitly an overall compensator which responds to requests to perform

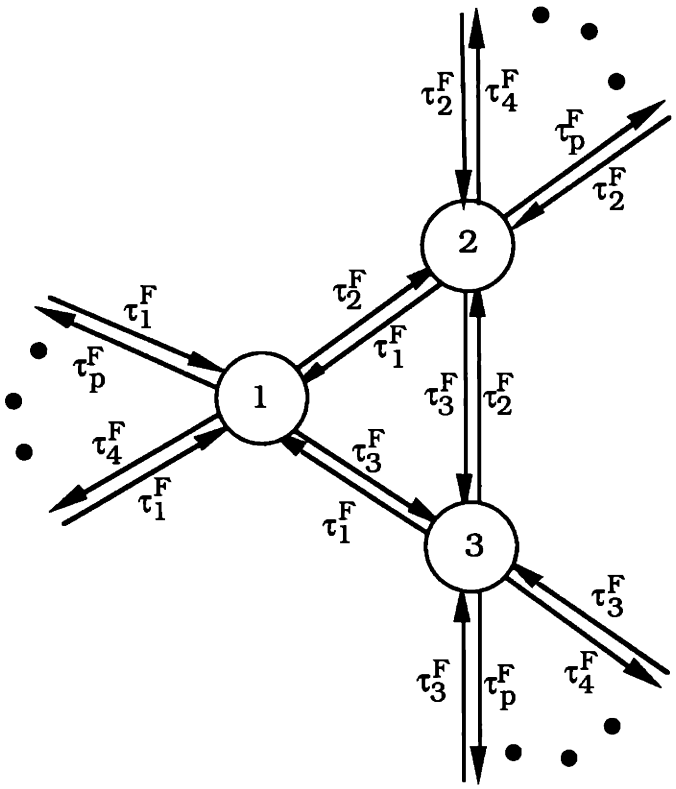

Figure 8.4: An Automaton to Construct C

particular tasks by enabling the appropriate compensator C_i. Given a set of p tasks $\mathbf{T}$, reachable by output feedback, and a task $i \in \mathbf{T}$, let $C_i : \Gamma^* \rightarrow U$ denote the compensator corresponding to task i. The compensator C that we construct admits events corresponding to requests for tasks as inputs and, depending on the inputs, C switches in an appropriate fashion between C_i. In order to model this, we use an automaton illustrated in Figure 8.4, which has p states, where state i corresponds to using the compensator C_i to control A. For each i, τ_i^F is a forced event, corresponding to switching to C_i. Let $\Phi_T = \{\tau_1^F, \ldots, \tau_p^F\}$ and $U_T = 2^{\Phi_T}$. The input to C is a subset of Φ_T, representing the set of tasks which are requested at present. The compensator

responds to this input as follows: Suppose that C is set-up to perform task i. There are three possibilities: (1) If the input is the empty set, then C disables all events in A, awaiting future task requests; (2) if the input contains τ_i^F, then C will not force any event but continue performing task i (thereby avoiding an unnecessary set-up transient); (3) Finally, if the input is not empty but it does not contain τ_i^F, then C will force one of the events in this set. At this level of modelling, we do not care which event C decides to force. Thus, we define $C : U_T \times \Gamma^* \rightarrow U \times \Phi_T$ so that given an input in U_T, C chooses the appropriate input ($\in U$) to A and generates the forced events ($\in \Phi_T$) as explained above. If the action of C corresponds to a switch from one task to another, the compensator C_i is initialized using the approach described previously. Specifically, suppose that the observer is in state $\hat{x}$ right before τ_i^F is forced. Consider the three cases described previously for C_i: If $\hat{x} \notin Z_r$ and $\hat{x} \notin E_o(L_i^{*c})$, then we use O starting from the initial state $\hat{x}$ and an $E_o(L_i^{*c})$-pre-stabilizing feedback. If $\hat{x} \in Z_r$ and $\hat{x} \notin E_o(L_i^{*c})$, then we start O'' at state $\hat{x} \times x_0$ and use the compensator described previously to drive the system to the desired set of states. Finally, if $\hat{x} \in E_o(L_i^{*c})$, then we start $O(L_i^{*c})$ at state $x_0^{L_i^{*c}} \times \hat{x}$, where $x_0^{L_i^{*c}}$ is the initial state of the minimal recognizer for L_i^{*c}, and we use the (f,u)-invariant feedback $K^{L_i^{*c}}$. A block diagram for A_C is illustrated in Figure 8.5.

8.3.2 Observable Tasks

In this section, we define a notion of observability for tasks which allows us to detect all occurrences of a given task. Consistent with our definition of detectability, we define task observability after an initial start-up transient. Specifically, we focus on detecting occurrences of tasks from that point in time at which the observer enters a recurrent state. One could, of course, consider the stricter condition of observability

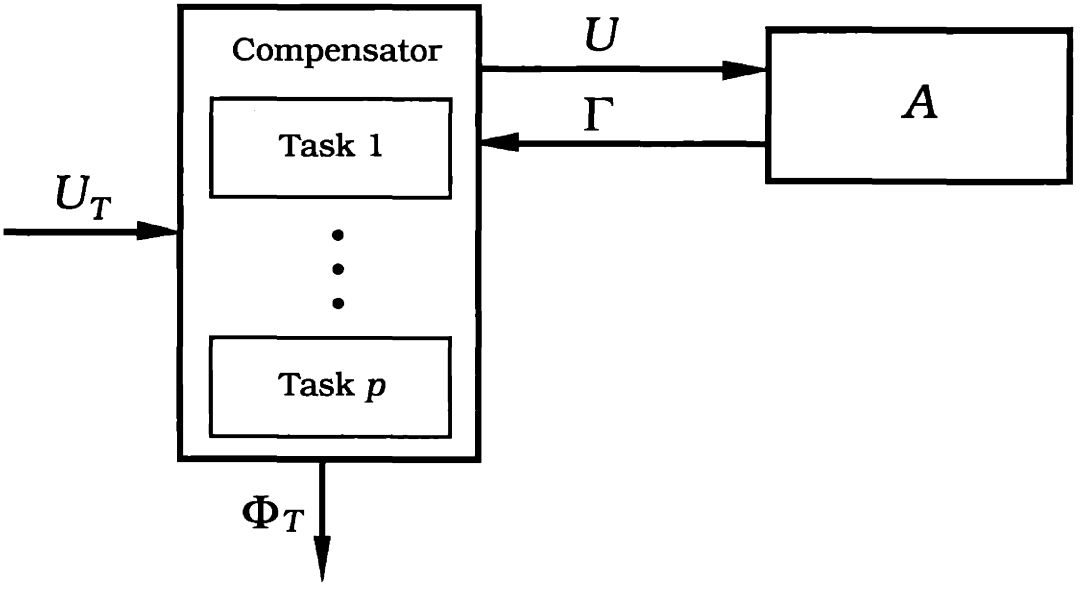

Figure 8.5: Block Diagram for A_C

without any knowledge of the initial state, but this would seem to be a rather strong condition. Rather, our definition can be viewed either as allowing a short start-up period or as specifying the level of initial state knowledge required in order for task detection to begin immediately (i.e., we need our initial state uncertainty to be confined to an element of Z_r).

Definition 8.13 A task $i \in \mathbf{T}$ is observable if there exists a function $\mathcal{I} : Z_r \times L(O, Z_r) \rightarrow \{\epsilon, \psi_i^F\}$ so that for all $\hat{x} \in Z_r$ and for all $x \in \hat{x}$, $\mathcal{I}$ satisfies

1. $\mathcal{I}(\hat{x}, h(s)) = \psi_i^F$ for all $s \in L(A, x)$ such that $s = p_1 p_2 p_3$ for some $p_1, p_2, p_3 \in \Sigma^*$ for which $t(p_2) \in L_i$, and
2. $\mathcal{I}(\hat{x}, h(s)) = \epsilon$ for all other $s \in L(A, x)$.

A set of tasks $\mathbf{T}$ is observable if each $i \in \mathbf{T}$ is observable. ❑

Since we assume that tasks are reachable throughout this chapter and will use task observability only in conjunction with task control, we will construct a test for the observability of task i assuming that it is reachable and that we are given an L_i^{*c}-restrictability compensator C_i which is consistent with T. Furthermore, thanks to consistency, we only need to construct $\mathcal{I}$ for $\hat{x} \in E_o(L_i^{*c})$ and for strings s such that $t(s) \in L_i^{*c}$. First, we let $A'_{L_i} = (G'_{L_i}, f'_{L_i}, d'_{L_i})$ be the same as the recognizer A_{L_i} but with a self-loop at the final state $x_f^{L_i}$ for each $\sigma \in \Xi$. Now, let $Q = (G_Q, f_Q, d_Q)$, with state space X_Q, denote the live part of $A'_{L_i} \parallel A$, i.e., X_Q is the set of states x in $X'_{L_i} \times X$ so that there exists an arbitrarily long string in $L(A'_{L_i} \parallel A, x)$. In fact, note that for each $x \in X$ such that $(x_0^{L_i}, x) \in X_Q$, there exists $s \in L(A, x)$ so that $t(s) \in L_i$. Finally, let $O_Q = (F_Q, w_Q, v_Q)$ be the observer for Q so that the state space Z_Q of O_Q is the reach of

$$Z_{Q0} = \bigcup_{\hat{x} \in E_o(L_i^{*c})} (\{x_0^{L_i}\} \times \hat{x}) \cap X_Q \tag{8.10}$$

in O_Q, i.e., $Z_Q = R(O_Q, Z_{Q0})$. Note that if i is observable, then the last event of each string in L_i must be an observable event. Assuming that this is the case, let

$$E_Q = \{\hat{z} \in Z_Q | \exists (x, y) \in \hat{z} \text{ such that } x = x_f^{L_i}\} \tag{8.11}$$

Given the observations on A_{C_i} let us first trace the trajectory in the observer O. At some point in time, O will enter some state $\hat{x} \in E_o(L_i^{*c})$. When this happens we know that the system starts tracking task i. At this point, let us start tracing the future observations in O_Q starting from the state $(\{x_0^{L_i}\} \times \hat{x}) \cap X_Q$. This trajectory will enter some $\hat{z} \in E_Q$ at some point in time. At this point, we know that task i may have been completed. However, for task observability, we need to be certain that task i is completed whenever it is actually, completed. Thus, for an observable task, it must be true that for all $\hat{z} \in E_Q$ and for all $(x, y) \in \hat{z}$, $x = x_f^{L_i}$. In this case we

can define $\mathcal{I}$ to be ϵ until the trajectory in O_Q enters E_Q and ψ_i^F from that point on Precisely stated, we have shown the following:

Proposition 8.14 Given a reachable task $i \in \mathbf{T}$ and an L_i^{*c}-restrictability compensator C_i so that i is consistent with C_i, if (1) the last event of each string in L_i is an observable event, and (2) for all $\hat{z} \in E_Q$ and for all $(x, y) \in \hat{z}$, $x = x_f^{L_i}$ then task i is observable in A_{C_i}. ❑

The procedure explained above allows us to detect the <u>first</u> completion of task i. Detecting other completions of task i is straightforward: Suppose that O enters the state $\hat{y}$ when O_Q enters E_Q. Note that $\hat{y} \in E_o(L_i^{*c})$. At this point we detect the first occurrence of task i and in order to detect the next occurrence of task i, we immediately re-start O_Q at state $x_0^{L_i} \times \hat{y} \cap X_Q$. The procedure continues with each entrance into E_Q signaling task completion and a re-start of O_Q. Note that the observer O runs continuously throughout the evolution of the system. Let $D_i^* : \Gamma^* \rightarrow \{\epsilon, \psi_i^F\}$ denote the complete task detector system (which, for simplicity, assumes an initial observer state of $\{Y\}$). We can think of D_i^* as a combination of three automata: the observer O, the system O_Q which is re-started when a task is detected, and a single one-state automaton which has a self-transition loop, with event ψ_i^F, which occurs whenever a task is detected. This event is the only observable event for this system. Note that both the O_Q re-start and the ψ_i^F transition can be implemented as forced transitions.

Finally, in the same way in which we constructed C from the C_i, we can also define a task detector D from the set of individual task detectors D_i. Specifically, if C is set at C_i initially, D is set at D_i. Using the output Φ_T of C, D switches between D_i. For example, if D is set at D_i and τ_j^F is forced by C, then D switches to D_j. The output of D takes values in $\Gamma_T = \{\psi_1^F, \ldots, \psi_p^F\}$. A block diagram for D is illustrated

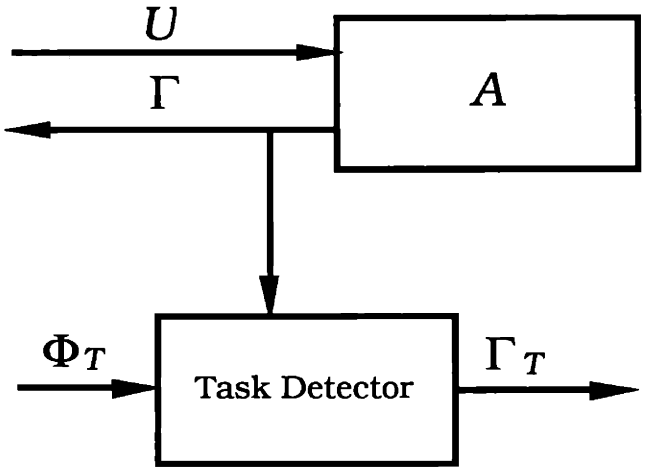

Figure 8.6: Task Detector Block Diagram

in Figure 8.6.

8.3.3 Task-Level Closed Loop Systems and Task Standard Form

Using the pieces developed in the preceding subsections we can now construct a task-level closed-loop system as pictured in Figure 8.7. The overall system is $A_{CD} = (G_{CD}, f_{CD}, d_{CD}, t_{CD}, h_{CD})$ where

$$G_{CD} = (X_{CD}, \Sigma \cup \Phi_T \cup \Gamma_T, \Phi \cup \Phi_T, \Gamma \cup \Phi_T \cup \Gamma_T, \Xi \cup \Phi_T) \tag{8.12}$$

Note that Φ_T and Γ_T are both observable and Φ_T is observable. Also, we include Φ_T in the tracking events to mark the fact that the system has switched compensators. This is important since following the switch, we will allow a finite length set-up. Also, since it does not make much sense in practice to force a switch to another compensator while the system is in the middle of completing a task, we impose the restriction that events in Φ_T can only be forced right after a task is completed. Since we require that all the tasks are observable (see Proposition 8.15), we can easily implement this

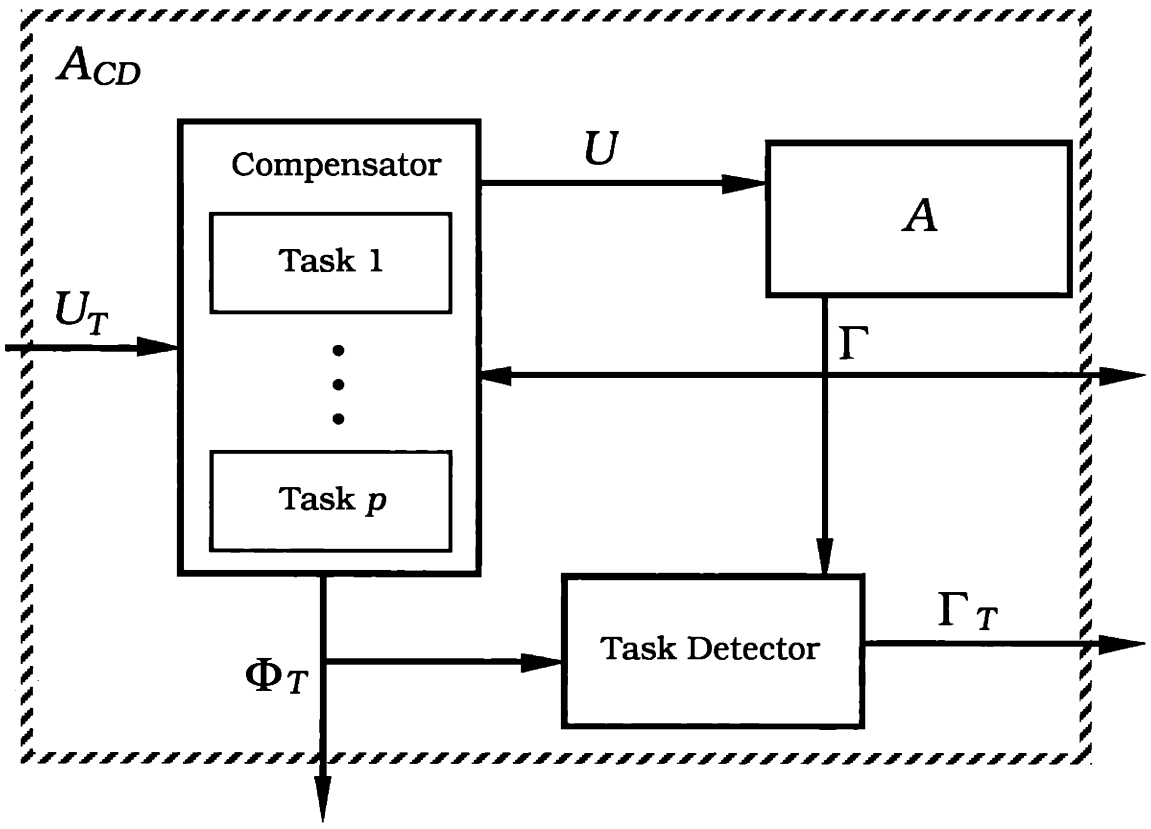

Figure 8.7: The Task-Level Closed-Loop System

restriction. Then, A_{CD} can only generate strings s such that

$$t(s) \in (\Xi \cup \{\epsilon\})^{n_t}(L_1^* \cup \cdots \cup L_p^*)(H_e(\tau_1)L_1^* \cup \cdots \cup H_e(\tau_p)L_p^*)^* \tag{8.13}$$

where n_t is the maximum number of tracking transitions needed until O enters the set of recurrent states in $E_o(L_i^{*c})$ for each $i \in \mathbf{I}$.

The higher-level operation of this system consists of the task initiation commands, Φ_T and the task completion acknowledgements, Γ_T. The input U_T indicating what subset of tasks can be enabled can be thought of as an external command containing the choices of subsets of Φ_T to be enabled. The use and control of this command involves higher-level modelling or scheduling issues beyond the purely task-level concept. What we show in this section is that the task-level behavior of A_{CD} can in fact be modelled, in the precise sense introduced in Section 8.2, by a much simpler automaton $A_{TSF} = (G_{TSF}, f_{TSF}, d_{TSF})$ illustrated in Figure 8.8 where all the events are controllable and observable, i.e.,

$$G_{TSF} = (X_{TSF}, \Sigma_{TSF}, \Phi_{TSF} = \Sigma_{TSF}, \Gamma_{TSF} = \Sigma_{TSF}) \tag{8.14}$$

We are not concerned with defining the tracking events of A_{TSF} since this alphabet is are not of concern in our main result below. We term A_{TSF} the task standard form.

Let us first define H_e. We first define $H_e(\epsilon) = \epsilon$ and $H_e(\psi_i) = L_i$. Note that, thanks to the independence of $\mathbf{I}$, for any pair of not necessarily distinct tasks i and j, no suffix of string in $H_e(\psi_i)$ can be in $H_e(\psi_j)$. Defining $H_e(\tau_i)$ is more tricky. There are two issues:

1. We need to take into account the fact that the closed loop system does not generate strings in L_i immediately after C switches to C_i. In particular, if we assume that O is in a recurrent state when C switches to C_i and if we let n_e

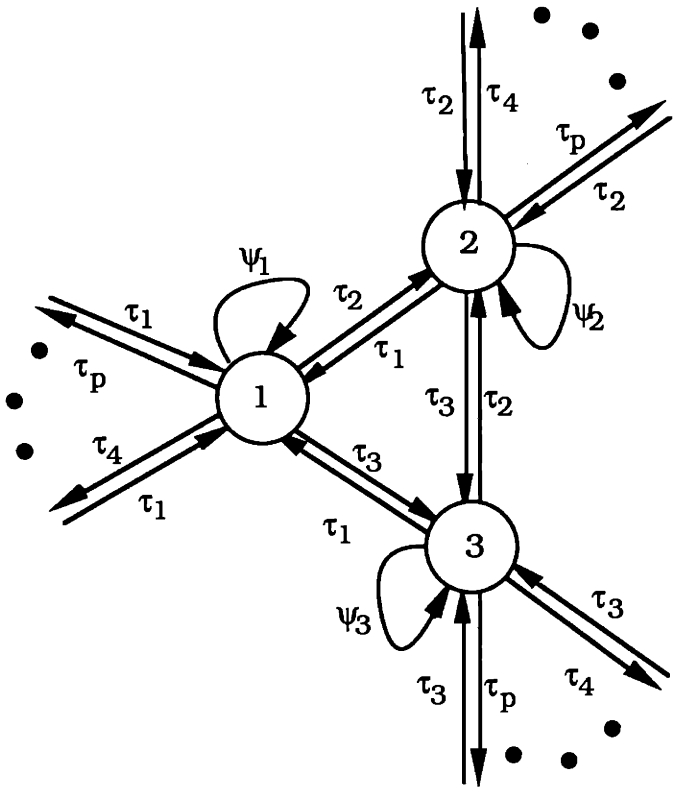

Figure 8.8: Task Standard Form: All events are controllable and observable.

denote the maximum number of tracking transitions that can occur in A for any trajectory in O that starts from a recurrent state of O up to and including the transition that takes the trajectory to a state in $E_o(L_i^{*c})$, then at most n_e tracking transitions can occur after C switches to C_i and before the behavior of the closed loop system is restricted to L_i^{*c}. Thus, we must choose H_e so that $H_e(\tau_i) \subseteq \tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$.

2. We also need to ensure the minimality of H_e. Specifically, . we now know that $H_e(\tau_i) \subseteq \tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$. Suppose that we let $H_e(\tau_i) = \tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$. Then, no suffix of a string in $H_e(\psi_i)$ can be in $H_e(\tau_i)$ since all strings in $H_e(\tau_i)$ start with τ_i^F. Also, no suffix of a string in $H_e(\tau_i)$ can be in $H_e(\tau_j)$ even if $i = j$. However, a suffix of a string in $\tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$ may be in $H_e(\psi_j)$ for some j. Thus, we let $H_e(\tau_i) = (\Xi \cup \{\epsilon\})^{n_e} \cap \overline{(\Xi \cup \{\epsilon\})^{n_e} L_T}$. Note that thanks to consistency, the strings in L_T cannot occur in a set-up of a task. Therefore, eliminating strings that end with a string in L_T will not cause any problems in restrictability.

Proposition 8.15 Given a set of tasks **T** that is reachable by output feedback and observable, A_{TSF} is an H_e-model of A_{CD}.

Proof: We first verify the detectability condition. Before defining H_o, let us define $t' : \Phi_T \cup \Gamma_T \rightarrow \Sigma_{TSF}$ as $t'(\tau_i^F) = \tau_i$ for all i and $t'(\psi_i^F) = \psi_i$ for all i. We then pick H_o as t' of the projection of the observation sequence over $\Gamma \cup \Phi_T \cup \Gamma_T$ to $\Phi_T \cup \Gamma_T$, i.e., $H_o(s) = t'(s \downarrow (\Phi_T \cup \Gamma_T))$, where, $s \downarrow \Pi$, in general, denotes that part of the string s over the alphabet $\Pi \subset \Sigma$. Finally, let n_d be n_t divided by the length of the shortest string in L_T. Then, thanks to observability, the first detectability condition is satisfied. Also, using minimality it is straightforward to verify the second detectability condition.

To verify the restrictability condition, we proceed as follows: Note that A_{TSF} is eventually restrictable to any infinite length string s in $L(A_{TSF})$. Thus, if we show

that A_{CD} is eventually $H_e(s)^c$-restrictable by output feedback, then the restrictability condition is verified. Let us now proceed with showing this. If the first event of s is some τ_i, then we simply force τ_i^F and look at the second event. If the first event of s is some ψ_i, then we force τ_i^F and wait until ψ_i. When ψ_i occurs, we look at the second event. In both cases, when we look at the second event, we repeat the same process. It then follows that A_{CD} is restricted as desired. Therefore, A_{TSF} is an H_e-model of A_{CD}. ❑

8.4 High-Level Models of Composite Systems

The formalism described in the previous section can obviously be applied in an iterative fashion to obtain a hierarchy of aggregate models in which words (i.e., tasks) at one level are translated into letters (higher level events) at the next level. In addition to such a vertical configuration of such models, it is also of considerable interest to investigate horizontal configurations. Specifically, in many applications the overall DEDS is actually an interconnection of a number of simpler DEDS. For example, a flexible manufacturing system (FMS) can be viewed as an interconnection of workstations and buffers. In such a system, typically the system-wide tasks to be performed can be broken down into individual tasks performed by subsystems. In this case, it makes sense to develop individual task controllers for each subsystem and then consider their interconnections, and obviously the computational savings from such an approach may be considerable. In this section we develop conditions under which we can construct such a system-wide task-level model from local task models. Once we have such an aggregate system-wide model one can then consider higher-level coordinated control of the entire system, and this we illustrate through a simple model of an FMS.

8.4.1 Composition

In this section we focus on a simple class of models. Specifically, we wish to examine a DEDS that consists of a set of uncoupled subsystems, i.e., as described by automata with no shared events. For example, a collection of workstations can be thought of in this manner. Obviously, for an FMS to produce anything useful, it will need to coordinate the activity of these workstations, e.g., by connecting them with conveyor belts, including buffers, etc. However, such coordination affects behavior at the task level—i.e., it does not influence how a workstation performs a specific task but rather what sequence of tasks is performed by what sequence of workstations . Thus, such coordination is in essence a higher-level control which can be viewed as a restriction on the behavior of the composite system. In the next subsection we will illustrate how such coordination can be incorporated, and this allows us to focus here on the uncoupled behavior of the subsystems.

Suppose that our system has m subsystems and that there is a set of reachable and observable tasks, Γ^j defined for each subsystem j. Let $A^j_{TSF} = (G^j_{TSF}, f^j_{TSF}, d^j_{TSF})$ be the task standard form for subsystem j, where Σ^j_{TSF} consists of τ^j_i and ψ^j_i for all tasks $i \in \Gamma^j$. Then, the (uncoupled) interconnections of these subsystems can be simply represented by

$$A^C = (G^C, f^C, d^C, h^C, t^C) = \|^m_{j=1} A^j_{CD} = A^1_{CD} \parallel \cdots \parallel A^m_{CD} \tag{8.15}$$

If we also let

$$A^{SF} = (G^{SF}, f^{SF}, d^{SF}, h^{SF}, t^{SF}) = \|^m_{j=1} A^j_{TSF} \tag{8.16}$$

and we let

$$H_e(\sigma) = H^j_e(\sigma), \text{ for } \sigma \in \Sigma^j_{TSF} \tag{8.17}$$

then since A^j_{TSF} share no events, H_e is minimal.

In order to state the main result of this section we need the following, where $A_\emptyset$ denotes A with all controllable events disabled:

Definition 8.16 Task $i \in \mathbf{T}$ is preventable if for all $s \in L(A_\emptyset)\, t(s) \notin L_i$. $\mathbf{T}$ is preventable if all its tasks are preventable. ❑

In the rest of this section, we will assume that $\mathbf{T}^j$ is preventable for all j:

Proposition 8.17 A^{SF} is an H_e-model of A^C.

Proof: Let us first verify the detectability condition. We let Φ_T^j and Γ_T^j denote the sets Φ_T and Γ_T of subsystem j. As in the proof of Proposition 8.15, we define a function t' so that $t'(\tau_i^{Fj}) = \tau_i^j$ and $t'(\psi_i^{Fj}) = \psi_i^j$ for all i and j. Then, we let

$$H_o(s) = t'(s \downarrow (\bigcup_{j=1}^{m} \Phi_T^j \cup \bigcup_{j=1}^{m} \Gamma_T^i))$$

and the rest of this proof follows as in the proof of Proposition 8.15.

To verify the restrictability condition, we proceed as follows: Note that A^{SF} is eventually restrictable to any infinite length string s in $L(A^{SF})$. Thus, if we show that A^C is eventually $H_e(s)^c$-restrictable by output feedback, then the restrictability condition is verified. Let us now proceed with showing this. If the first event of s is some τ_i^j, then we force τ_i^{Fj}, disable all events of all the other subsystems and look at the second event. If the first event of s is some ψ_i^j, then we force τ_i^{Fj}, disable all events of all the other subsystems and wait until ψ_i^j. When ψ_i^j occurs, we look at the second event. In both cases, when we look at the second event, we repeat the same process. Also, thanks to preventability, it then follows that A^C is restricted as desired. Therefore, A^{SF} is an H_e model of A^C. ❑

The concept of preventability and the nature of our higher level model deserve some comment. The framework we have described here is essentially one of serial

operation—i.e., our system-wide task level model describes the sequence in which tasks are completed by all subsystems (i.e., task 1 by workstation 1, then task 3 by workstation 2, ...). While preventability is an essential concept in any system, in practice we only want to prevent a subsystem from operating if its next action is truly serially dependent on the completion of another task by another subsystem. In other cases we may want to allow several subsystems to be operating in parallel. While it is possible to infer parallelism by a detailed examination of event and task sequences (much as one can when examining the event trajectory for the shuffle product [49] of subsystems), this is not a totally acceptable solution. To capture parallelism in a more direct way we need to add complexity to our task standard form by keeping track not only of task completion but also task initiation, so that in the task-level composite our state at any time will indicate what set of tasks are ongoing. The detailed development of these ideas is left to future work.

8.4.2 Subsystem Interactions and Higher Level Control

As we indicated previously, interactions between subsystems can often be modelled via restrictions on the system-wide task-level model. In the following example, we illustrate how the presence of buffers between workstations can be represented as such restrictions. Furthermore, we will see that the restrictions imposed by each buffer can be dealt with independently, and each restriction can be viewed as the action of a compensator.

Example 8.18 Suppose that we have two systems, each capable of performing two reachable and observable tasks. We let task 1 and 2 be the tasks associated with the first system, and tasks 3 and 4 be the tasks associated with the second system. We model these systems by the task-level automata illustrated in Figure 8.9

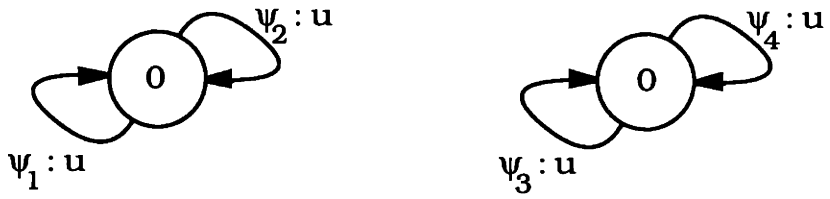

Figure 8.9: Two Systems

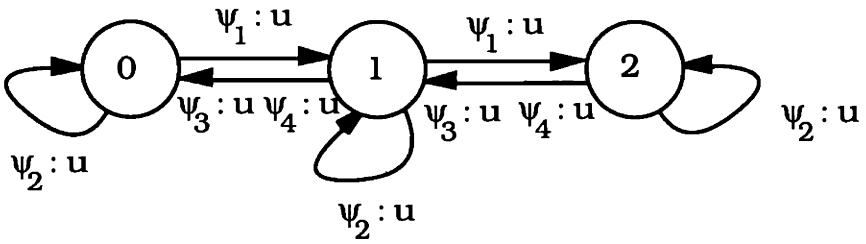

Figure 8.10: First Buffer Implementation

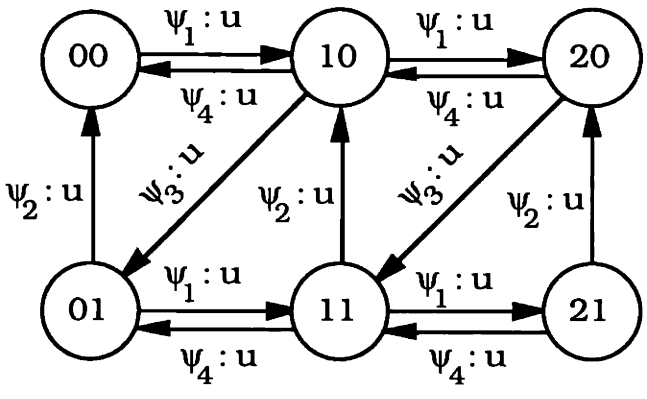

Figure 8.11: Second Buffer Implementation

where all tasks are also assumed to be observable and for simplicity of exposition, we have assumed that there is no set-up involved in switching between tasks (i.e., the workstations produce no extraneous events between the event sequences corresponding to the various tasks). In this case there is no need to include both ψ_i and τ_i in the task-level model. Suppose that each system is a model of a workstation in an FMS, which manufactures two different parts, so that the first part can be manufactured by performing tasks 1, 3, and 2 successively, and the second part can be manufactured by performing task 1 followed by task 4. Let us assume that there is a buffer of size two between the two workstations so that every occurrence of ψ_1 increases this buffer and every occurrence of ψ_3 or ψ_4 decreases it. This buffer imposes two restrictions on the system: First, ψ_1 should not occur when the buffer is full, and second, ψ_3 and ψ_4 should not occur when the buffer is empty. These restrictions can be implemented in a straightforward way, and the closed loop system is illustrated in Figure 8.10. Let us also assume that there is a buffer of size one between the two workstations so that every occurrence of ψ_3 increases this buffer and every occurrence of ψ_2 decreases it. Restricting the system further by the conditions imposed by this buffer yields the closed loop system in Figure 8.11. ❑

In general, given A^C, A^{SF}, and H_e so that A^{SF} is an H_e-model of A^C, suppose that a buffer restriction can be implemented by a compensator C on A^{SF}. Then, thanks to Proposition 8.7, we can find a compensator C' so that the closed loop system A^{SF}_C is an H_e-model of the closed loop system $A^C_{C'}$. Furthermore, we can consider further restrictions on the behavior of A^{SF}_C corresponding to higher-level primitives, called procedures, which consist of sequences of tasks. Let us illustrate this for Example 8.18. In this example, we let A^C_{SF} represent the system in Figure 8.11. Based on

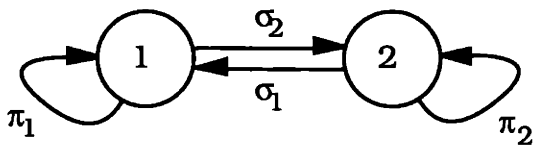

Figure 8.12: Procedure Standard Form for Example 8.18

the two parts to be manufactured defined in Example 8.18, we define two procedures for this system, namely $L_1 = \psi_1\psi_3\psi_2$, and $L_2 = \psi_1\psi_4$. It is straightforward to show that the states 00 and 10 are L_1^{*c}-restrictable, and states 00,10,01, and 11 are L_2^{*c}-restrictable, and furthermore, A_{SF}^{C} is both eventually L_1^{*c}-restrictable and eventually L_2^{*c}-restrictable. Note that since all tasks are observable, we do not need to worry about procedure detection in this case. Thus, we can construct compensators for each procedure as we did for tasks in the previous section. Let K denote the combined compensator, let A_{CK}^{SF} denote the closed loop system, and let A_{PSF} denote the automaton in Figure 8.12, where σ_i represents switching to procedure i and π_i represents completing procedure i. Then we can find a map H_e so that A_{PSF} is an H_e-model of A_{CK}^{SF}. We term A_{PSF} the procedure standard form. Finally, thanks to Proposition 8.8, A_{PSF} also models $A_{C'K}^{C}$, our original system combined with task compensators for each system, buffer restrictions implemented by C', and the procedure compensator K.

8.5 Discussion

In this chapter, we have introduced concepts of higher-level modelling for DEDS based on a given set of primitive event sequences corresponding to tasks which the system may perform. Through our investigation of task reachability we constructed task

compensators and this, together with task observability allowed us to construct simple high-level models of automata so that events in the high-level model correspond to set-up and completion of tasks. We have also considered the higher level modelling of systems that are composed of a set of subsystems and we have shown how the coordinated control of such a system can be effected by higher-level control which we have referred to as procedures.

The aggregation scheme presented in this chapter addresses an important issue of computational complexity in DEDS problems of interest. In particular, for computations involving systems that are composed of m subsystems so that the description of each subsystem has n states, the complexity is a function of n^m. Our aggregation procedure is important in reducing n considerably since the cardinality of the state space of the high-level model equals the number of tasks.

Chapter 9
Conclusions

In this thesis, we have investigated a series of problems concerning discrete event dynamic systems (DEDS). The DEDS formulation is a useful framework for the analysis and control of the key behavior of many complex systems, and we believe that our work has added significantly to this framework. In particular, the main contribution of this thesis has been the development of a regulator theory for DEDS. In the next section, we outline our contributions in more detail.

9.1 Contributions of This Thesis

The major components of the development of our regulator theory for DEDS can be summarized as follows:

Stabilization and Compensators: In our analysis, we started from the assumption that a set of "good" states, E, representing the desired modes of or starting points for operation, is given. Based on this, we formulated and analyzed notions of stability and stabilizability, which guarantee that the system returns to E in a finite number of transitions. Then, we turned our attention to the system with partial observations and focused on estimating the current state of the system based on these observations. Chapter 5 combined the work on stability and observability to address a problem of stabilization by output feedback. Thanks also to our work on resiliency, our work

up to and including Chapter 5 has allowed us to characterize the existence of and to construct compensators that guarantee that the system returns to the desired modes of operation in finite time in spite of a burst of observation errors.

Event Trajectory Reconstruction and Error Propagation: Our treatment of invertibility in Chapter 6 has dealt with problems of reconstructing event trajectories from partial observations. Such a characterization of invertibility can be used to address problems related to monitoring complex systems, post-mortem analysis, troubleshooting, etc. Further, our notion of resilient invertibility has allowed us to address problems related to catastrophic error propagation in sequential decoding. In particular, we discussed the existence and the construction of a resilient inverter which confines the reconstruction error within a finite window around the burst of observation errors.

Regulation: In Chapter 7, our concern was to specify the desired modes of operation, i.e., the set E. In particular, Chapter 7 characterized the tracking capabilities of DEDS, and formulated and analyzed restrictability. Restrictability has also allowed us to establish connections to the work of Wonham and Ramadge. The notion of reliability also formulated in Chapter 7 has allowed us to address the problem of recovery from system failures and we have shown that this notion is closely tied to stability. As shown in the beginning of Chapter 8, this work can be combined with our results in Chapter 5 to address problems of tracking and restrictability in the context of intermittent observations of events.

Multi-Level Control: Our work in Chapter 8 has drawn from the concepts developed in all the previous chapters to present a framework for the higher-level modelling of DEDS based on a given primitive set: event sequences corresponding to tasks which the system may perform. Using the notion of eventual restrictability by

output feedback, we characterized what we mean by completing a task. The notion of task detection has used concepts from both the development of observability and the development of invertibility, except that since our goal was also to control the system based on task detection, we could not tolerate any delay in task observability. Constructing task compensators and task detectors allowed us to build simple high-level models of controlled DEDS so that events in the high-level model correspond to set-up and completion of tasks. We have also considered the higher level modelling of systems that are composed of a set of subsystems and we have shown how the coordinated control of such a system can be effected by higher-level control which we have referred to as procedures. Finally, the simple nature of our higher-level models also facilitate quantitative analysis of DEDS.

Computational Complexity: An important contribution of this thesis is in the treatment of computational complexity of DEDS problems. We have shown that the source of the NP-completeness of the supervisory control problem in the case of partial observations, is the cardinality of the observer state space. Thus, if in fact the observer has polynomial state space, then the problem of controllable languages for the case of partial observations can also be solved in polynomial time. Also, the aggregation scheme presented in Chapter 8 addressed an important issue of computational complexity in DEDS problems of interest. In particular, for computations involving systems that are composed of m subsystems so that the description of each subsystem has n states, the complexity is a function of n^m. Our aggregation procedure is important in reducing n considerably since the cardinality of the state space of the high-level model equals the number of tasks.

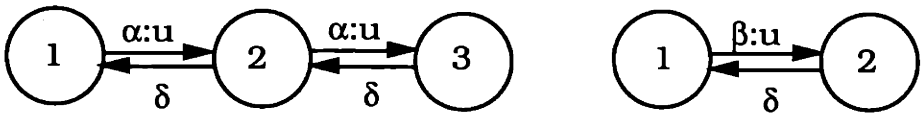

Figure 9.1: Two Systems

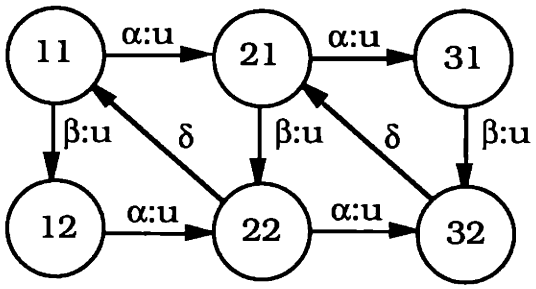

Figure 9.2: Composition of the Systems in Figure 9.1

9.2 Future Work

The work presented in this thesis suggests many areas for future work. Several of these are listed below:

More Powerful Models: One may also extend the concepts formulated in this thesis to models that have more descriptive power than finite state automata. One potentially fruitful extension would be adopting the model of Finitely Recursive Processes (FRP) (see for example [27]). FRP may involve infinite state space since it allows for recursion, but the model is expressed in terms of a finite number of processes. Then our notion of stability may be formulated in the FRP framework by choosing a set of "good" processes, representing the desired modes of operation, and testing the stability of the other processes with respect to this set of "good" processes.

Stability in the Composite: As commented in Section 3.4, our assumption, in treating stabilizability of a composite system, that only shared events are controllable is a restrictive assumption and it should be relaxed even at the cost of some additional complexity. However, if we still wish to keep complexity comparable to that of the algorithms in Section 3.4, then the stabilizing feedback corresponding to states must be of a particular type. Specifically, recall that $\widetilde{X}$ is the set of composite states that are either in E or from which a shared event is defined. Then, we would like the stabilizing feedback to be independent for the subsystems A_1 and A_2 when the composite state is not in $\widetilde{X}$. More precisely, there should exist a stabilizing feedback $K : X \to U$ for the composite so that there exists $K_i : X_i \to U_i$ for $i = 1, 2$ such that for all $x = (x_1, x_2) \in \overline{\widetilde{X}}$, $K(x) = K_1(x_1) \cup K_2(x_2)$. For example, consider the two systems illustrated in Figure 9.1 and their composition illustrated in Figure 9.2. Let $E = \{32\}$ and note that $\widetilde{X} = \{22, 32\}$. It turns out that there exists a stabilizing feedback for the system in Figure 9.2 which is unique (modulo the feedback for states in $\widetilde{X}$): disable β at states 11 and 21. Since this feedback unique but it does not satisfy the above independency condition, we cannot construct a stabilizing feedback for this system while keeping complexity low as we did for the restrictive case in Section 3.4. Thus, relaxing this restriction and designing efficient stability tests would first require characterizing systems that can be stabilized by feedback that satisfies the independency condition. Note that this involves stabilization by local, or decentralized feedback.

Composite Systems and Decentralized Control: Recall that in Chapter 3, we have outlined an approach for constructing an efficient algorithm for testing stability of a composite of m subsystems, and we have raised several issues related to the coupling assumption. It is important to relate these issues to practical applications and

to establish connections to the problem of testing fairness in computer/communication systems. Also, by formulating a problem of satisfying strong or weak coupling by state feedback, one could introduce control to the notion of fairness. Finally, related to the formulation of composite systems is the problem of decentralized control, for example, as formulated in [10]. This problem assumes that both the observations and the inputs are local, i.e., a compensator only has access to the observable and controllable events in its neighborhood. In this case, one would be concerned with properties of compensators that are composed of other compensators. We can translate such a problem into our framework as follows:

Given $\Gamma_i \subset \Gamma$ and $\Phi_i \subset \Phi$ for $i = 1, \ldots, j$ for some j, and a complete language L, A is eventually L-restrictable by decentralized output feedback if there exist $C_i : \Gamma_i^* \rightarrow U$ such that the closed loop system A_C eventually generates strings in L, where C is defined by $C(s) = \bigcap_{i=1}^{p}(C_i(s \downarrow \Gamma_i) \cup \overline{\Phi}_i)$ for all $s \in \Gamma^*$.

The treatment of this problem is of obvious importance in communication systems, since the commands to perform tasks such as transmitting packets and information on the completion of such tasks need to be transmitted through the network, and as these control transmissions increase, the network performance deteriorates.

Parallelism: An important area of further research is extending our approach in Chapter 8 to capture parallelism in interconnected systems. This is an important extension since many problems of interest involving interconnected systems deal with parallel execution of specified processes. For example, scheduling problems in FMS involve processing certain parts in parallel at different workstations. Such an extension would involve two issues: First, we would need a notion to represent strings that may be executed in parallel. More precisely, given two strings s_1 and s_2 that may be executed in parallel, we let $s_1 \parallel s_2$ denote the shuffle of s_1 and s_2, i.e., if we let σ_i

denote the first event in s_i, then $s_1 \parallel s_2$ is defined in a recursive fashion as follows:

$$s_1 \parallel s_2 \triangleq \sigma_1((s_1/\sigma_1) \parallel s_2) \cup \sigma_2(s_1 \parallel (s_2/\sigma_2))$$

and, to complete the definition, we let $\epsilon \parallel s = s$. Then, $s_1 \parallel s_2$ denotes the set of possible strings that may be generated by executing s_1 and s_2 in parallel. Using such a tool, our notion of modelling can be applied to primitives that consist of strings that may be executed in parallel. For example, for the FMS example presented in Chapter 8, a primitive, that corresponds to a schedule which manufactures equal quantities of Part 1 and Part 2, may be represented as $\psi_1\psi_3\psi_2 \parallel \psi_1\psi_4$. Our modelling concept can then be applied to such a primitive. Second, even if two strings are specified in the primitive in a way that would permit them to be executed in parallel, this does not necessarily imply that the underlying system can execute these strings in parallel. For example, suppose that two events α and β, each on a different workstation are enabled simultaneously. One interpretation of this is that both α and β are being executed in parallel. Suppose then that α occurs and the system makes a transition to some state x. Since β is also enabled, it may occur after α, and thus β must be defined at x, since otherwise, parallel execution of α and β is inconsistent with the system model. Thus, identifying events or strings that can be executed in parallel involves testing if these events satisfy conditions such as the one above. Finally, the formulation outlined so far for adapting our framework allows parallel execution but only in an implicit way, since our methods of analysis do not explicitly identify what is being performed in parallel (rather, one must look in detail at the event sequence to determine this). To make it explicit, one needs to model the "start-up" of each task as well as the completion, i.e., a task is now represented by two events instead of one event. Such a formulation would double the complexity of the task standard form, but by keeping track of each task which is started at a workstation, we can explicitly

characterize the set of tasks that are being executed in parallel.

Performance Analysis: The task standard form is a simple automaton with a simple and extremely regular structure. Thus, its use should facilitate the application of performance analysis techniques, such as the ones presented in [12,18,23,30], and this could then result in major simplifications as compared to the direct performance analysis of the original system.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, 1974.

[2] M. Ben-Ari. *Principles of Concurrent Programming*. Prentice Hall, 1982.

[3] A. Benveniste and P. Le Guernic. A denotational theory of synchronous communicating systems. Report, INRIA, 1987.

[4] G. V. Bochmann. *Distributed System Design*. Springer Verlag, 1983.

[5] J. R. Büchi and D. Siefkes. *Lecture Notes in Mathematics: Decidable Theories II*. Springer Verlag, 1973.

[6] P. E. Caines and S. Wang. Classical logic based regulators for partially observed automata: Dynamic programming formulation. In *CISS John Hopkins University*, March 1989.

[7] J. C. Carroll and D. Long. *Theory of Finite Automata*. Prentice-Hall, Inc., 1989.

[8] T. M. Chin and A. S. Willsky. Stochastic petri net modelling of wave sequences in cardiac arrhythmias. *Comput. and Biomed. Research*, April 1989.

[9] H. Cho and S. I. Marcus. Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations. Department of Electrical and Computer Engineering, The University of Texas at Austin.

[10] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. on Automatic Control*, March 1988.

[11] E. M. Clarke and O. Grumberg. Research on automatic verification of finite-state concurrent systems. *CMU-CS-87-105*, Jan 1987.

[12] G. Cohen, D. Dubois, J. P. Quadrat, and M. Viot. A linear system theoretic view of discrete event process. In *Proceedings of CDC*, December 1983.

[13] E. W. Dijkstra. Solution of a problem in concurrent programming cotrol. *CACM*, 5(9), Sept 1965.

[14] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *CACM*, 17(11):643–644, Nov 1974.

[15] S. Eilenberg. *Automata, Languages, and Machines.* Academic Press, Inc., 1976.

[16] G. A. Frank, D. L. Franke, and W.F. Ingogly. An architecture design and assessment system. *VLSI Design*, August 1985.

[17] R. F. Garzia, M. R. Garzia, and B. P. Zeigler. Discrete event simulation. *IEEE Spectrum*, December 1986.

[18] S. B. Gershwin. A hierarchical framework for manufacturing systems scheduling: A two-machine example. In *Proceedings of CDC*, December 1987.

[19] W. B. Gevarter. Expert systems: Limited but powerful. *IEEE Spectrum*, August 1983.

[20] C. H. Golazewski and P. J. Ramadge. Control of discrete event processes with forced events. In *Conference on Decision and Control*, Dec 1987.

[21] C. H. Golazewski and P. J. Ramadge. Mutual exclusion problems for discrete event systems with shared events. In *Conference on Decision and Control*, Dec 1988.

[22] A. Göllü and P. Varaiya. Hybrid dynamical systems. In *Proceedings of CDC*, December 1989.

[23] Y. Ho. Performance evaluation and perturbation analysis of discrete event dynamic systems. *IEEE Trans. on Automatic Control*, July 1987.

[24] C. A. R. Hoare and J. C. Shepherdson. *Mathematical Logic and Programming Languages*. Prentice-Hall, Inc., 1984.

[25] W. M. L. Holcombe. *Algebraic Automata Theory*. Cambridge University Press, 1982.

[26] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[27] K. İnan and P. Varaiya. Finitely recursive process models for discrete event systems. *IEEE Trans. on Automatic Control*, July 1988.

[28] R. E. Kalman, P. L. Falb, and M.A. Arbib. *Topics in Mathematical System Theory*. McGraw-Hill, 1969.

[29] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete event systems. Systems Control Group Report 8612, University of Toronto, July 1986.

[30] O. Z. Maimon and S. B. Gershwin. Dynamic scheduling and routing for flexible manufacturing systems that have unreliable machines. LIDS Publication LIDS-P-1610, MIT, September 1986.

[31] O. Z. Maimon and G. Tadmor. Efficient supervisors in discrete event systems. In *Proceedings of 1986 Int. Conf. Systems, Man and Cybernetics*, 1986.

[32] M. E. Merchant. Production: A dynamic challenge. *IEEE Spectrum*, May 1983.

[33] R. Milner. *Lecture Notes in Computer Science: A Calculus of Communicating Systems*. Springer Verlag, 1980.

[34] J. S. Ostroff and W. M. Wonham. A temporal logic approach to real time control. In *Proceedings of CDC*, December 1985.

[35] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., 1982.

[36] W. W. Peterson and Jr. E. J. Weldon. *Error-Correcting Codes*. The MIT Press, 1972.

[37] P. J. Ramadge. Observability of discrete event systems. In *Proceedings of CDC*, December 1986.

[38] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM J. of Cont. and Opt.*, September 1987.

[39] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. of Cont. and Opt.*, January 1987.

[40] J. Sifakis. Deadlocks and livelocks in transition systems. Technical Report 185, R.R., January 1980.

[41] G. Tadmor and O. Z. Maimon. From system constraints to finite state machine models in discrete event systems.

[42] G. Tadmor and O. Z. Maimon. Control of large discrete event systems: Constructive algorithms. LIDS Publication LIDS-P-1627, MIT, December 1986.

[43] J. G. Thistle and W. M. Wonham. Control problems in a temporal logic framework. *Int. J. Control 44 (4)*, 1986.

[44] W. Thomas. Automata on infinite objects. *Lehrstuhl für Informatik II, RWTH Aachen, D-5100 Aachen*, April 1988.

[45] L. Tobias and J. L. Scoggins. Time-based air-traffic management using expert systems. *IEEE Control Systems Magazine*, April 1987.

[46] J. N. Tsitsiklis. On the control of discrete event dynamical systems. *Math. C. S. S.*, 1989.

[47] A. F. Vaz and W. M. Wonham. On supervisor reduction in discrete event systems. *International Journal of Control*, 1986.

[48] W. M. Wonham. *Linear Multivariable Control: A Geometric Approach.* Springer-Verlag Inc., 1985.

[49] W. M. Wonham. A control theory for discrete-event systems. In *NATO ASI Series, Advanced Computing Concepts and Techniques in Control Engineering.* Springer-Verlag, 1988.

[50] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. of Cont. and Opt.*, May 1987.