SIAM J. CONTROL AND OPTIMIZATION
Vol. 30, No. 6, pp. 1423–1446, November 1992

TRACKING AND RESTRICTABILITY IN DISCRETE EVENT DYNAMIC SYSTEMS*

CÜNEYT M. ÖZVEREN† AND ALAN S. WILLSKY‡

Abstract. This paper formulates and analyzes notions of tracking and restrictability for discrete event dynamic systems (DEDS). The DEDS model used is a finite-state automaton in which there is control over some events. A second set of events, called the set of *tracking events*, is also specified, and the tracking problem is one of constructing a compensator so that the tracking event trajectory of the closed loop system follows a given string exactly. This problem is analyzed in detail and, in particular, a characterization of all trackable strings is characterized. The related notion of restrictability is analyzed in which the closed-loop system is required to generate tracking event strings in a given desired language. A relaxed version of this concept is also analyzed, allowing an initial transient before desired language tracking is achieved. Finally, a notion of reliability is introduced and analyzed, which allows for testing if the system can recover from errors in a finite number of transitions, and algorithms are presented for constructing compensators for reliable restrictability. A manufacturing system example is used to motivate and illustrate the problems considered and results obtained.

Key words. tracking, control, reliability, stability, discrete events

AMS(MOS) subject classification. 93

1. Introduction.

In the past few years, there has been considerable research on the topic of discrete event dynamic systems (DEDS) [1]–[3], [6]–[9], [18]–[21]. One characteristic of much of this activity is that the control objectives have frequently been stated in linguistic terms, i.e., in terms of characteristics of the possible closed-loop event trajectories. In contrast, in much of our previous work [13], [14], [16], we have focused directly on control concepts of stability, observability, stabilization, and output feedback, providing some of the elements required 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 [13], [14], [16], which can be considered an error recovery concept, appears to be a natural one in the discrete-event context.

In this paper, we develop another element 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 terms of the concept of *trackable languages*, as well as a second notion, *restrictability*, which is a slight generalization of the notion of (language) controllability of Ramadge and Wonham in [19]. While our analysis of restrictability represents a relatively modest addition to the existing theory of controllable languages, we also consider two related, new 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 or stable 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

* Received by the editors September 5, 1989; accepted for publication (in revised form) June 24, 1991. This research was supported by Air Force Office of Scientific Research grant AFOSR-88-0032 and by Army Research Office grant DAAL03-86-K0171. Part of this work was also done while the first author was employed by the Digital Equipment Corporation.

† Telecommunications and Networking, Digital Equipment Corporation, 550 King Street LKG1-2/A19, Littleton, Massachusetts 01460.

‡ Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

return to the desired, restricted behavior following a burst of errors or failures. As we will see, stable restrictability plays a key role in characterizing reliable restrictability.

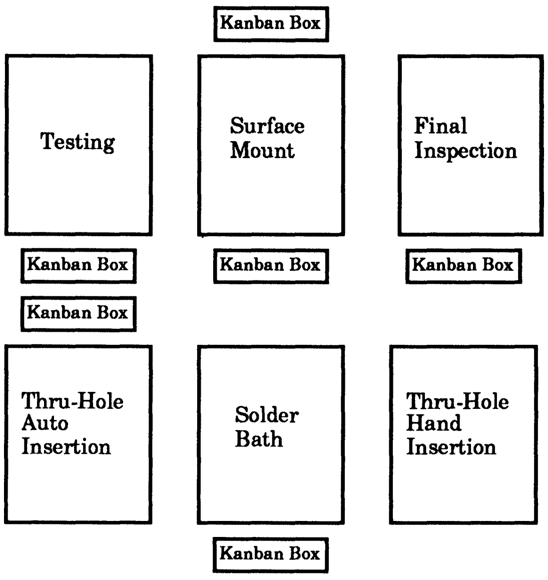

FIG. 1.1. *An example of a computer board manufacturing floor.*

To motivate the problems considered in this paper and to provide an example that we can use to illustrate their solution, let us briefly describe a particular manufacturing application. More detailed investigations of this and other applications of our regulator theory are given in [15]. Figure 1 illustrates the floor plan of a computer board manufacturing facility consisting of several workstations and capable of soldering surface mount chips on both sides of a board, mixed with thru-hole mounting (via auto-insertion and hand insertion). Another workstation is used for soldering both kinds of thru-hole devices. One workstation is used for testing random board samples at various phases of the manufacturing process, and finally, each board goes through a routine test and inspection after completion. This manufacturing floor uses a Japanese inventory system, termed the Kanban system. Boards are transported through specially marked "kanban" boxes in quantities of 1 to 10 in each box. There are very few kanban boxes between different workstations, guaranteeing that inventories are very low, and thus, among other things, that latency through the manufacturing process is also very low.

A typical board with both sides populated with surface mount components and with mixed thru-hole components goes through the following process: The board first visits the surface mount station for side 1 components, where, first, a solder paste is applied to the board; next, the components are placed on the board; and, finally, solder is applied. The board then goes to the auto-insertion workstation where thru-hole devices are automatically inserted. Next, if necessary, some components are inserted by hand, and the board arrives at the solder bath. There, the boards are first baked, to remove the moisture, and then passed through the wave solder. After that, if there are any side 2 surface mount components, the board goes to the surface mount workstation again for the mounting of side 2 components and, finally, the board goes through final inspection and testing. To construct a manageable example in the scope of this paper, we capture the dynamics of a system of this type with two workstations and two kinds of boards. We consider a surface mount workstation W1 and a thru-hole workstation W2. One kind of board has only side 1 surface mount components and some thru-hole components. The second kind of board has surface mount components on both sides, as well as thru-hole components. The surface mount workstation will perform two tasks: side 1 mounting (for either kind of board), which we call Task 1, and side 2 mounting, including the inspection for the second kind of board, which we call Task 2. The thru-hole workstation also performs two tasks: thru-hole mounting for the second board, denoted by Task 3, and thru-hole mounting and inspection for the second board, Task 4. Thus, to be completed, the first board must go through Tasks 1 and 4, and the second must go through Tasks 1, 3, and 2, in that order.

Let S_i (respectively, F_i) denote the starting (respectively, finishing) of Task i. In addition, we assume that there is a unit capacity buffer B1 to store boxes going from the surface mount to the thru-hole workstation and another unit capacity buffer B2 in the opposite direction; i.e., there is space for one kanban box in each direction. We then model this system using the automata in Fig. 1.2. Here, circles represent states, and the arcs represent transitions labeled with events. Also, :u indicates that the corresponding event is controllable (i.e., we can decide whether to start a task), and :! indicates thatthe corresponding event is a "tracking" event, which is identified as an event that is of interest in characterizing desired behavior (see §2 for the precise mathematical model). Suppose that, at a given time, the objective of the manager of such a plant is to manufacture equal amounts of each kind of board. Then, we must perform Task 1 twice, and all the other tasks once to produce one board of each kind. Furthermore, the correct production of these parts requires the correct sequencing of these tasks and the corresponding transfers of boards. In particular, suppose that the time needed to complete Task 1 is comparable to the time to complete Task 4, while the time for Task 3 is comparable the time for Tasks 1 and 2 combined. In this case, we can form a production schedule by performing first Tasks 2 and 1 on the surface mount workstation while performing Task 3 on the other, and then Task 1 on the surface mount workstation while performing Task 4 on the other. Note that it makes no difference if we reverse the order, i.e., require that Tasks 1 and 4 are done first, and so on. In essence, all we must know to construct a schedule is the list of tasks that must be performed and the time it takes to complete each relative to the others. Thus one "cycle" of the schedule, producing one board of each type, corresponds to the completion of any of the sequences in

$$(1.1)\quad L_S = (F_1(F_2F_3+F_3F_2)+F_2(F_1F_3+F_3F_1)+F_3(F_1F_2+F_2F_1))(F_1F_4+F_4F_1),$$

where multiplication in (1.1) corresponds to concatenation and addition to union (so that $F_1F_2F_3F_4F_1$ and $F_3F_1F_2F_1F_4$ are elements of L_S). Note that by constructing the schedule in this fashion, we are allowing for concurrency; that is, at some points in time, both machines may be working. The control problem then is to exercise the available event controls to ensure that the manufacturing system adheres to the schedule of a succession of sequences from L_S, perhaps with an initial start-up transient and hopefully with the ability to recover gracefully from errors or failures. In this paper, we provide a mathematical framework that allows us to solve problems such as this, and indeed we will revisit this example in later sections to illustrate the construction of controllers that meet design objectives such as this.

In the next section, we introduce our mathematical framework and collect several definitions and results. In §3 we formulate a notion of tracking and present algorithms

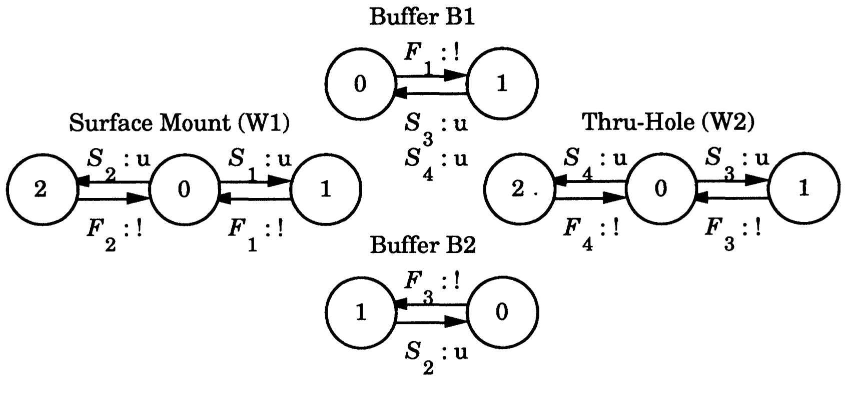

FIG. 1.2. *A model of the computer board manufacturing example.*

for constructing compensators for tracking specific strings (e.g., particular elements of L_S). In §4 we consider the problem of restricting behavior to a specified set of desired event sequences (e.g., restricting the manufacturing system to successive completion of elements of L_S), a concept very closely related to the notion of controllability of Wonham and Ramadge. Furthermore, in this section, we introduce concepts of eventual and stable restrictability that allow us to address questions concerning the transient behavior of controlled DEDS and investigate the reliability or error-correcting capability of such a system. As we will see, the stronger notion of stable restrictability leads, in general, to considerable computational efficiencies compared to the weaker concept of eventual restrictability. Finally, in §5 we summarize our results and discuss several directions for further work.

2. Background. The class of systems we consider are nondeterministic finite-state automata defined on $G = (X, \Sigma, \Xi, U)$, where X is the finite set of states, with $n = |X|$; Σ is the finite set of possible events; $\Xi \subset \Sigma$ is the set of events that we wish to track; and $U \subset 2^{\Sigma}$ is the set of admissible control inputs, corresponding to the choices of sets of controllable events that can be enabled. The dynamics defined on G are

$$x[k+1] \in f(x[k], \sigma[k+1]), \tag{2.1}$$

$$\sigma[k+1] \in (d(x[k]) \cap u[k]) \cup e(x[k]). \tag{2.2}$$

Here $x[k] \in X$ is the state, $\sigma[k] \in \Sigma$ is the next event, and $u[k] \in U$ is the next control input. The function $d : X \to 2^{\Sigma}$ specifies the set of possible events defined at each state, $e : X \to 2^{\Sigma}$ 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. Without loss of generality, we assume that $e(x) \subset d(x)$. 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 parts of the next section, we will use this general framework. In the remainder of this paper, however, we assume the slightly more restrictive framework of [19] 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 $e(x) = d(x) \cap \overline{\Phi}$.

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 intermediate events that are not of direct interest. We use $t : \Sigma^* \longrightarrow \Xi^*$, to denote the projection of strings over Σ into Ξ^*. The quintuple $A = (G, f, d, e, t)$[1] representing our system can also be visualized graphically as in Fig. 1.2, where the first symbol in each arc label denotes the event. We mark the controllable events by :u and tracking events by !.

We use several basic notions. First, given $Q \subset X$, we use $R(A, Q)$ to denote all the states that can be reached from Q in zero or more steps (so that $Q \subset R(A, x)$). Second, there is the notion of liveness: A DEDS is alive if $d(x)$ is nonempty for all x. We will assume that this is the case. A third notion that we need is the composition $A_{12} = A_1 \parallel A_2$ of two automata $A_i = (G_i, f_i, d_i, e_i, t_i)$, which share some common events. 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. Note that our manufacturing system can be described by the composition of the four automata in Fig. 1.2, with shared events capturing the fact that a task cannot begin if a board is not available.

Central to our work is the notion of stability studied in [16] (see also [17]). Let E be a given subset of X. We say that a state $x \in X$ is *E-prestable* if every trajectory starting from x passes through E in a bounded number of transitions. The state $x \in X$ is *E-stable* if every state reachable from x is E-prestable, and the DEDS is *E-stable* if every $x \in X$ is E-stable. Note that E-stability for all of A is identical to E-prestability for all of A, and that this condition guarantees that all trajectories go through E infinitely often. We refer the reader to [16] for a complete discussion of stability and for an $O(n^2)$ test for E-stability of a DEDS.

In [16] we also study state feedback laws of the form $K : X \longrightarrow U$, where the resulting closed-loop system is $A_K = (G, f, d_K, e, t)$ with $d_K(x) = (d(x) \cap K(x)) \cup e(x)$. Generally, we wish to avoid feedback laws so that $d_K(x)$ is empty for some x, and we build this constraint into our notions of stabilization. For example, a DEDS is E-stabilizable if there exists a feedback K so that A_K is both alive and E stable.

For many control problems, such as those considered in this paper, we must consider compensators that use both current state and event trajectory information. Such a compensator, which is described by a map $C : X \times \Sigma^* \longrightarrow U$, yields a closed-loop system A_C, which is the same as A but with

$$\sigma[k+1] \in d_C(x[k], s[k]) \triangleq (d(x[k]) \cap C(x[k], s[k])) \cup e(x), \tag{2.3}$$

where $s[k] = \sigma[0] \cdots \sigma[k]$ with $\sigma[0] = \epsilon$. Note that this class of compensators is similar to the class of supervisors introduced in [19], although, by allowing dependence on the current state, we can achieve a somewhat richer class of behaviors. Note also that we can always write A_C as a DEDS with an expanded (and possibly infinite) state space to realize the dynamics inherent to the map C. As we will see, for our purposes, we can restrict attention to finite state compensators.

In the following, we also use well-known notions of dynamic invariance [16], [18]: A subset Q of X is *f-invariant* if $f(Q, d) \subset Q$, where $f(Q, d) = \bigcup_{x \in Q} f(x, d(x))$. If

[1] On occasion, we will construct auxiliary automata for which we will not be concerned with either control or tracking. In such cases, we will omit e and t from the specification.

$V \subset X$ is f-invariant in A, we denote the restriction of A to V by $A|V$. We say that 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. However, recall that, in general, we must also preserve liveness. Thus we say that 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. Also, given any set $W \subset X$, there is a minimal (f, u)-invariant subset V of W with a corresponding unique *minimally restrictive* feedback K.

Note that, if there exists a cycle in A that consists solely of events that are *not* in Ξ, then the system may stay in this cycle indefinitely. It is not difficult to check for the absence of such cycles, and we assume that this is the case. On occasion, we use the image automaton that keeps track of the state only after the occurrence of tracking events. The state space Y^t of this automaton consists of the union of the set Y_1^t of states that can be reached by tracking events and the set Y_0^t of states to which no events are defined (Y_0^t captures possible start-up behavior). Let $r = |Y^t|$.

It is useful to phrase questions concerning event trajectories in terms of languages [4]. Let L be a regular language over a finite alphabet and let (A_L, x_0) be a minimal recognizer for L. Given $s = pqr$ for some strings p, q, and r over Σ, where p is a prefix of s and r is a suffix of s, we use s/pq to denote r, and we say that q is a substring of s. Finally, we will use the notion of a *complete* language: L is complete if (a) every $s \in L$ is a proper prefix of some other $r \in L$ (so that all trajectories have unlimited extensions), and (b) L is prefix-closed (so that all initial segments of a trajectory are in L). Note that, for a complete language, all strings generated by the recognizer (A_L, x_0) are in L (so that all states are "final" [4]).

3. 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 [16]. Furthermore, 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.

3.1. Trackable languages. We define tracking as being able to restrict the system behavior so that the automaton starting from the current state must generate a desired string:

DEFINITION 3.1. 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 $t(L(A_C, x)) \subset s\Xi^*$.

As an example, consider the system in Fig. 3.1. Any string in $(\alpha\beta)^*$ is trackable from 0, and a compensator for tracking all such strings can be defined by $C(0, \emptyset) = \{\alpha\}, C(1, \alpha) = \{\beta\}, C(3, \alpha\gamma) = \{\beta\}, C(0, \alpha\beta) = \{\alpha\}$, and so on. 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. In the context of manufacturing systems, this notion would be useful in checking if a part can be manufactured at all by the system. In our example, it is obvious that, for example, the second board can be manufactured since the task sequence 1,3,2 can be "tracked." However, realistically, in a complex system it may not be so obvious if a certain board can be manufactured at all.

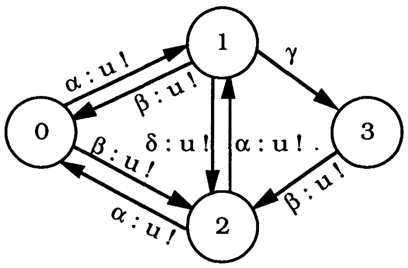

FIG. 3.1. *Simple example.*

DEFINITION 3.2. A language L is a *trackable language from* x if it is complete and if each string in L is trackable from x.

The class of trackable languages is closed under arbitrary unions, and we 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 it is prefix closed. However, we can construct the supremal complete sublanguage of the intersection. Let the function $\chi : 2^{\Xi^*} \to 2^{\Xi^*}$ denote removing all the strings, in a given language L, that have no infinite extensions in L, i.e.,

$$\chi(L) = \{s \in L \mid s \text{ has an infinite extension in } L\}. \tag{3.1}$$

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, the first event of a string $s \in L_T(A, x)$ must be defined at some state that is reachable from x by events in Ξ; i.e., it must be in the set

$$d^t(x) = d(R(A|\Xi, x)) \cap \Xi. \tag{3.2}$$

In Fig. 3.1, 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, must be trackable from x. We now characterize the set $l_T(x)$ of such events (i.e., the strings of length 1 in $L_T(A, x)$). 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 $R(A|\Xi, x)$ is no longer alive; see below:

$$e^t(x) = (d^t(x) \cap \overline{\Phi}) \cup \{\tau \in d^t(x) | \exists y \in R(A|\Xi, x) \text{ such that } d(y) = \{\tau\}\}. \tag{3.3}$$

For example, in Fig. 3.1, $e^t(1) = \{\beta\}$, 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 *any* event from x, and if it contains one event, then we can *only* 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 we have the following proposition.

PROPOSITION 3.3. *It holds that* $l_T(x) = \{\tau \in d^t(x) | \{\tau\} \cup e^t(x) = \{\tau\}\}$.

For example, in Fig. 3.1, $l_T(1) = \{\beta\}$.

After some $\tau \in l_T(x)$ is tracked, the automaton is in some state in $f^t(x, \tau) \triangleq f(R(A|\Xi, x), \tau)$. Consequently, the remaining part of the string that can be tracked, with τ as prefix, must be trackable from *all* these states. Thus we have the following implicit characterization of $L_T(A, x)$.

PROPOSITION 3.4. *It holds that* $L_T(A,x) = \bigcup_{\tau \in l_T(x)} \tau\chi(\bigcap_{y \in f^t(x,\tau)} L_T(A,y))$.

To solve this equation, we construct an automaton $A^t = (G^t, f^t, d^t, e^t, 1)$, where $G^t = (Y^t, \Xi, \Xi, U)$ and 1 is the identity map. For the system in Fig. 3.1, A^t is illustrated in Fig. 3.2. Recall that, if $e^t(x)$ contains one element, then we can only track that event from x. In this case, let

$$K'(x) = \begin{cases} \emptyset & \text{if } e^t(x) \neq \emptyset, \\ d^t(x) & \text{otherwise.} \end{cases} \tag{3.4}$$

In Fig. 3.2, $K'(1) = \emptyset$, 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.,

$$D_T = \{x \in Y^t | e^t(x) \geq 2\}. \tag{3.5}$$

To be able to track complete languages, we must avoid 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 Fig. 3.2, V is all of the states. Finally, let $K(x) = K_V(x) \cap K'(x)$. Note that, for $x \in V$, all the events in $d^t_K(x)$ are trackable from x, i.e., $l_T(x) = d^t_K(x)$.

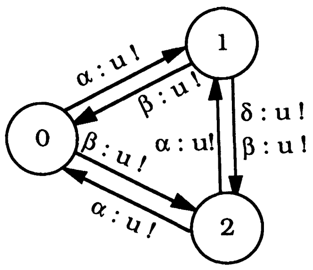

FIG. 3.2. *The automaton* A^t *for Fig.* 3.1.

LEMMA 3.5. *If* $x \in Y^t \cap \overline{V}$, *then* $L_T(A,x) = \emptyset$. *If* $x \in V$, *then* $L_T(A,x) \subset L(A^t_K, x)$.

Proof. The proof is straightforward. □

To compute $L_T(A,x)$, let us first focus on the case in which $A^t_K|V$ is deterministic. In this case, since $l_T(x) = d^t_K(x)$ for all $x \in V$, then, for any $x \in V$, the language generated from x in A^t_K is certainly trackable from x, and, in fact, it is the supremal such language.

PROPOSITION 3.6. *If* $A^t_K|V$ *is deterministic, then for all* $x \in V$, $L_T(A,x) = L(A^t_K, x)$. *Furthermore, for all* $x \in Y^t \cap \overline{V}$, $L_T(A,x) = \emptyset$.

Proof. The proof is straightforward using Lemma 3.5 and the fact that, for all $x \in V$ and $\tau \in d^t_K(x)$, $f^t_K(x,\tau)$ is single-valued. □

To complete the picture when $A^t_K|V$ is deterministic, we must construct $L_T(A,x)$ for the states x in $\overline{Y}^t$. Let us first seek any such x that is also in $R(A|\Xi, V)$. 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 $A^t_K|V$ 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 to achieve invariance for V. Thus we must also disable this event at x.[2] Thus, if we define

$$l_V(x) = \{\tau \in l_T(x) | f^t(x,\tau) \in V\}, \tag{3.6}$$

then

$$L_T(A,x) = \bigcup_{\tau \in l_V(x)} \tau L_T(A, f^t(x,\tau)), \tag{3.7}$$

which allows us to compute $L_T(A,x)$ from $L_T(A,y)$, $y \in V$. Next, suppose that $x \notin R(A|\Xi, V)$ 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

$$L_T(A,x) \supset \tau L_T(A, f^t(x,\tau)), \tag{3.8}$$

and indeed, if *only* this case occurs, $L_T(A,x)$ is given as in (3.7). 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 $A^t_K|V$ 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

$$w^t(\hat{x},\tau) = \bigcup_{x \in \hat{x}} f^t(x,\tau), \tag{3.9}$$

$$v^t(\hat{x}) = \bigcap_{x \in \hat{x}} (d^t(x) \cap K(x)) \cup e^t(x). \tag{3.10}$$

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 under these dynamics. Specifically, we take the state space Z^t of O^t to be

$$\begin{aligned} Z^t = R(O^t, \{\hat{x} \in 2^V | \hat{x} = \{x\} \text{ and } x \in V, \\ \text{or } \hat{x} = f^t(x,\tau) \text{ for some } x \in \overline{Y}^t, \tau \in l_T(x)\}). \end{aligned} \tag{3.11}$$

Figure 3.3 illustrates O^t for the automaton in Fig. 3.2. Note that $l_T(3) = \{\beta\}$ and $f^t(3,\beta) = \{2\}$.

[2] The existence of the feedback K, in fact, guarantees that τ can be disabled while preserving liveness.

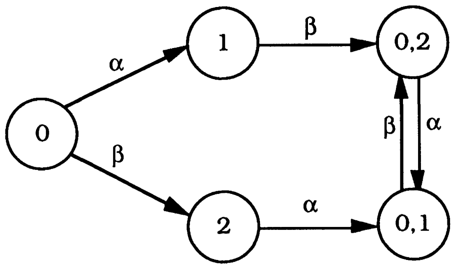

FIG. 3.3. *The automaton O^t for Fig. 3.2.*

Let D_z be the set of dead states in Z^t, i.e.,

$$D_z = \{\hat{x} \in Z^t | v^t(\hat{x}) = \emptyset\}; \tag{3.12}$$

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 3.7. *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

$$L_T(A, x) = \bigcup_{\tau \in l'_T(x)} \tau L(O_{K^t}^t, f^t(x, \tau)).$$

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 must ensure 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 (3.9), and the allowable event function v^t, defined via an intersection (3.10), 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 that 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^t_V. 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 [13], $|Z^t|$ may be exponential in $|V|$, and thus computing L_T may have exponential complexity in $|V|$. In [13] we provide some bounds on observer state space size and give examples 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 [13] can be used to compute a bound on the size of the recurrent part of Z^t. Refer to [10] for an example illustrating 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 [13].

3.2. 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 Fig. 3.1, $(\beta\alpha)^*$ 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 3.8. 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 in the following, if a state x is E-prestabilizable for some E, then any string trackable from *all* states in E is eventually trackable from x. For the example in Fig. 3.1, 2 is $\{0, 1\}$-prestabilizable, and $(\beta\alpha)^*$ is trackable from both 0 and 1.

LEMMA 3.9. *Given $x \in X$ and $E \subset X$ such that x is E-prestabilizable,*

$$L_{ET}(A, x) \supset \bigcap_{y \in E} L_T(A, y).$$

Proof. The proof is 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-prestabilizable, since, otherwise, a trajectory from x may cycle arbitrarily through states from which s is *not* trackable.

PROPOSITION 3.10. *Given x, let $\mathbf{E}$ be the set of all sets $E \subset X$ such that x is E-prestabilizable. 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. The proof is 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 obtain a slightly tighter formula for $L_{ET}(A,x)$ as follows. Let $Y' \subset X$ (with $r' = |Y'|$) be the set of states from which at least one tracking event is defined, i.e.,

$$Y' = \{x \in X | d(x) \cap \Xi \neq \emptyset\}. \tag{3.13}$$

COROLLARY 3.11. *Given x, let $\mathbf{E}'$ be the set of all sets $E' \subset Y'$ such that x is E'-prestabilizable. Then*

$$L_{ET}(A,x) = \bigcup_{E' \in \mathbf{E}'} \bigcap_{y \in E'} L_T(A,y).$$

Proof. ($\supset$) The proof is trivial by the above proposition.

($\supset$) Let $s \in L_{ET}(A,x)$, and let $E \subset X$ be a set so that x is E-prestabilizable and $s \in L_T(A,x')$ for all $x' \in E$. Next, let

$$E' = R(A|\Xi, E) \cap Y'.$$

Thanks to our assumption that it is not possible for A to generate arbitrarily long sequences of events in $\overline{\Xi}$, E is E'-prestable. Thus x is E'-pre-stabilizable, and $E' \in \mathbf{E}'$. Also, since all events in $\overline{\Xi}$ are uncontrollable, $s \in L_T(A,y)$ for all $y \in E'$. Therefore $s \in \bigcup_{E' \in \mathbf{E}'} \bigcap_{y \in E'} L_T(A,y)$. □

To compute $\mathbf{E}'$, we must check, for each subset E' of Y', if x is E'-prestabilizable. Thus computing $L_{ET}(A,x)$ has complexity exponential in r'. However, testing if a string s is eventually trackable (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 must do is to compute the set of states in Y' from which s is trackable and test if x is prestabilizable with respect to this set. For example, $(\beta\alpha)^*$ is trackable from 0 and 1 in Fig. 3.1. Since 2 is $\{0,1\}$-prestabilizable, $(\beta\alpha)^*$ is eventually trackable from 2. In fact, both $(\alpha\beta)^*$ and $(\beta\alpha)^*$ are eventually trackable from *all* the states.

4. Restrictability.

In this section, we first address the problem of restricting the output behavior of a system to a given language, representing a slight generalization of the notion of controllable languages in the Wonham and Ramadge framework as we also consider arbitrary initial states. Next, we present the concept of eventual restrictability and stable restrictability, which allow us the flexibility of restricting the behavior after a finite number of transitions. Finally, we present and analyze a notion of reliability that allows us to model failure or error events and to test if the system can be made to recover 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.

4.1. 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 4.1. 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.

The class of L-restrictable sets is closed under arbitrary unions and intersections. Let X_L denote the maximal L-restrictable set. To compute X_L, we first construct a recognizer for L and then formulate the problem of restrictability as one 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 that 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

$$f'_L(x,\sigma) = \begin{cases} f_L(x,\sigma) & \text{if } x \neq b \text{ and } \sigma \in d_L(x), \\ \{b\} & \text{otherwise.} \end{cases} \tag{4.1}$$

As an example, consider the system illustrated in Fig. 4.1(a), which is identical to an example in [22]. 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. 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 Fig. 4.1(b). 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 must 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 Fig. 4.2, where we have taken $\Xi = \{\beta, \delta\}$ as the tracking alphabet.

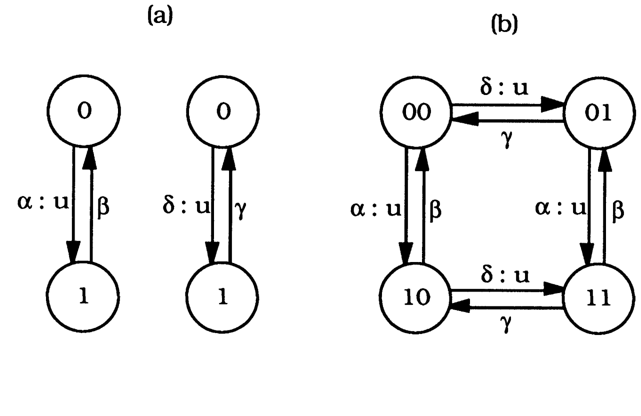

FIG. 4.1. *Example for restrictability.*

Let A^{LA} denote the composite $A'_L \parallel A$ and let

$$E^{LA} = \{(x,y) \in X^{LA} | x \neq b\}. \tag{4.2}$$

For example, Fig. 4.3 denotes the composite of the automata in Figs. 4.1(b) and 4.2, where the first component of the labels of each state represent the state of A'_L, 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 that do *not* have b as their first component.

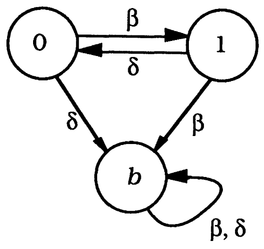

FIG. 4.2. *Automaton A'_L.*

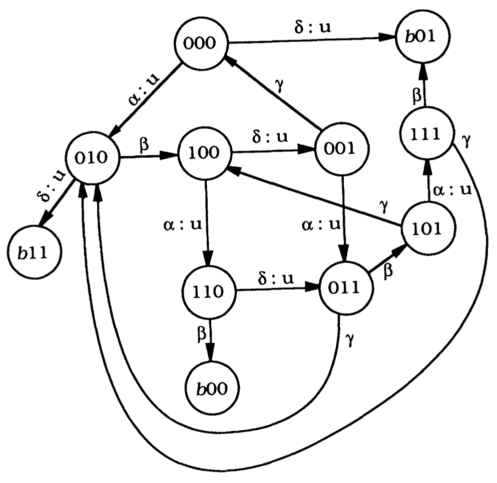

FIG. 4.3. *Composite of A and A'_L.*

Given $Q \subset X$, let $I(Q)$ denote the maximal sustainably (f, u)-invariant subset of Q and let K_I denote the associated minimally restrictive feedback. Then we have the following proposition.

PROPOSITION 4.2. *A state $x \in X$ is L-restrictable if and only if $(x_0, x) \in I(R(A^{LA}, (x_0, x)) \cap E^{LA})$. 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 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. The proof is straightforward by assuming the contrary in each direction. □

In Fig. 4.3, 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.

The compensator C is implemented as follows: Given the initial state, x, of A, we initiate $A^{LA}_{K_I}$ at (x_0, x). The compensator is simply the set-valued function of the present state of $A^{LA}_{K_I}$ given in Proposition 4.2.

Finally, the following result presents a straightforward construction for X_L.

PROPOSITION 4.3. *We have that* $X_L = \{x \in X | (x_0, x) \in I(R(A^{LA}, S_L))\}$, *where* $S_L = \{(x_0, x) \in X^{LA}\}$, *and the complexity of this computation is* $O(|X^{LA}|^2)$.

Proof. The proof is 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 [16]). □

In our board manufacturing example, our objective is to follow the specified schedule that corresponds to restricting system behavior to L_{BM}, the prefix closure[3] of L_S^*. It is not difficult to check that some of the states of the system of Fig. 1.2 are restrictable with respect to L_{BM}. In particular, let the quadruple of the states of W1, W2, B1, and B2 represent the state of the composite system. Then Fig. 4.4(a) represents the closed-loop system after restricting the behavior from state $(0,0,1,1)$. (For simplicity in this figure, intermediate states are not shown explicitly, but the end of each transition terminates at a state.) Note, for example, that initially, only S_2 is enabled, since enabling S_1 could lead to F_1, which would overflow B1, and enabling S_3 could lead to F_3, which would overflow B2.

We can now relate our results to the notion of controllable languages of Wonham and Ramadge. We refer the reader to [19] 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 4.4 follows.

PROPOSITION 4.4. $L(A^{LA}_{K_I}, (x_0, x))$ *is the supremal controllable sublanguage of the legal language* L.

Proof. This is straightforward to check from the definitions in [19] and the fact that K_I is minimally restrictive. □

As an example, if the initial state of the system in Fig. 4.1(b) is 00, then the supremal controllable sublanguage of L is the language generated by state 000 in Fig. 4.3 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 [22] for this example.

As a final comment, note that, from the development in §3, it might be expected 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. For example, if $e(x) \neq \emptyset$, disable all controllable events at this state;

[3] This allows for the fact that we may be in the *middle* of one of the sequences in L_S in (1.1), which certainly is consistent with our desire to follow the schedule.

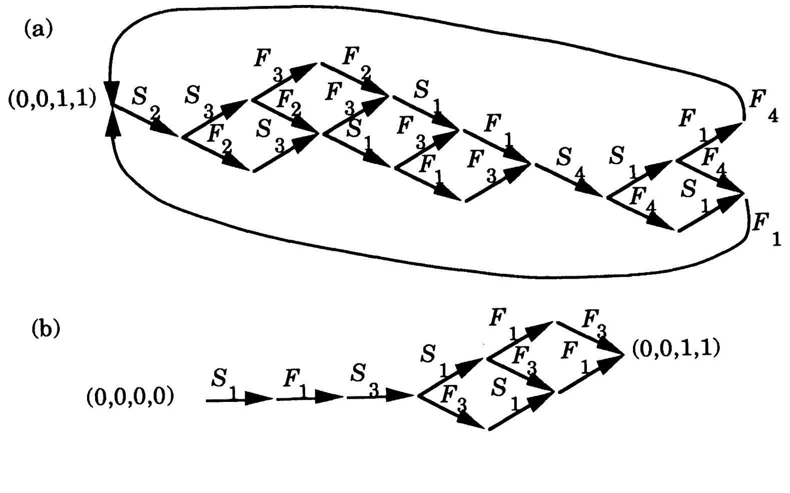

FIG. 4.4. *Part of the closed loop system for the compensated board manufacturing example.*

if $e(x) = \emptyset$, disable all but one controllable event. Thus, in this context, it is more meaningful to *fix* L and consider the questions we have addressed here.

4.2. Eventual restrictability and stable restrictability. As noted in the preceding section, some of the states in the manufacturing system of Fig. 1.2 are L_{BM}-restrictable. Others (such as (0,0,0,0)) are not. However, for such states, it *is* possible to design control rules so that we do begin to follow the desired schedule after a short initial set-up transient. This provides the motivation for a natural generalization of our notion of restrictability. For example, consider the system in Fig. 4.5, where $\Xi = \Sigma$, and suppose that $L = (\alpha\gamma + \beta\delta)^*$. The automaton A'_L is illustrated in Fig. 4.6 and the automaton A^{LA} is illustrated in Fig. 4.7, where the transitions 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.

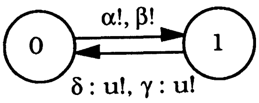

FIG. 4.5. *Example for eventual restrictability.*

DEFINITION 4.5. 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.

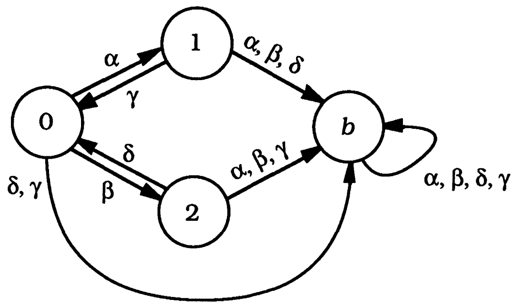

FIG. 4.6. *Automaton A'_L for $L = (\alpha\gamma + \beta\delta)^*$.*

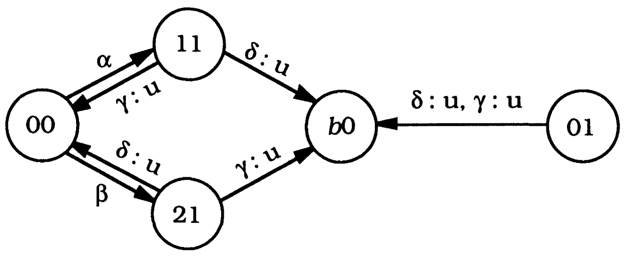

FIG. 4.7. *Composite of A and A'_L for the eventual restrictability example.*

The class of eventually L-restrictable sets is closed under arbitrary unions and intersections, and thus it has a maximal element X_{EL}. A set closely related to X_{EL} is X_{SL}, the maximal X_L-prestabilizable set, i.e., the set of states that can be driven into states from which L-restrictability can be achieved. The advantage of considering X_{SL} is that it is easy to compute and (directly from the results on prestabilizability [16]) for states in X_{SL}, $n_a \leq r$ and the computation of X_{SL} has complexity $O(n^2)$.

DEFINITION 4.6. Given $x \in X$ and a complete language L over Ξ, x is *stably L-restrictable* if x is X_L-prestabilizable. Given $Q \subset X$, Q is *stably L-restrictable* if all $x \in Q$ are stably L-restrictable. Finally, A is *stably L-restrictable* if X is stably L-restrictable.

A compensator for stable restrictability can be constructed by using two compensators in tandem: The first one is a state feedback that prestabilizes A with respect to X_L. The second one is the compensator of Proposition 4.2 for restricting the language generated by x to L, where x is the element of X_L that the trajectory first visits.

One natural question that arises concerns the relationship between X_{EL} and X_{SL}. Clearly, $X_{EL} \supset X_{SL}$, and, in fact, for many systems and languages the two sets are equal (in particular, this is true if A is stably L-restrictable). For example, it can be verified that our computer board manufacturing system is stably L_{BM}-restrictable, and Fig. 4.4(b) illustrates part of the closed-loop system that ensures that $(0,0,0,0)$

reaches $(0,0,1,1)$ in a finite number of transitions. However, as first shown in [5],[4] there are systems and languages for which $X_{EL} \neq X_{SL}$, and, in some of these cases, the length n_a of the initial transient until we begin strings in L need not be bounded by r and can be quite long.

While it is beyond the scope of this paper to present a full investigation of the relationship between X_{EL} and X_{SL} and conditions under which they are equal (or at least n_a is small), we can make a few remarks concerning this issue. A simple example adapted from [5] is given in Fig. 4.8, where all events are tracking events and no event is controllable. If we let $L = \delta^* + \alpha^k\alpha^*\beta\delta^*$ for some fixed but arbitrary integer k, then $X_L = \{1\}$, $X_{EL} = \{0,1\}$, $X_{SL} = \{1\}$, and $n_a = k$. Note that, if L were taken as δ^*, then $X_{EL} = X_{SL} = \{1\}$, while, if L were $\alpha^k\alpha^*\beta\delta^*$, then X_{EL} and X_{SL} are both empty. The difficulty thus appears to be related to the interaction between the two components of L together with the long prefix $\alpha^k\alpha^*\beta$ of one of these components. Some of these difficulties are removed if we restrict our attention to a subclass of languages corresponding to the successive completion of a sequence of "tasks," i.e., to languages of the form $(L_f)^{*c}$, the prefix closure of the language of all concatenations of strings in the finite set $L_f = \{w_1, \cdots, w_m\}$ (note that this is exactly the form of $L_{BM} = (L_S)^{*c}$ for our manufacturing example).

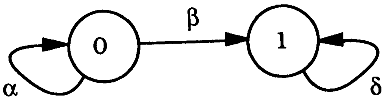

FIG. 4.8. *Example illustrating the bound on n_a for $L = \delta^* + \alpha^k\alpha^*\beta\delta^*$.*

Restricting ourselves to languages of this form does eliminate the situation depicted in Fig. 4.8, but it is not sufficient to guarantee that $X_{EL} = X_{SL}$. For example, consider the system in Fig. 4.9, where all events are tracking events and no event is controllable. If $L = (L_f)^{*c}$ with $L_f = \{\alpha\beta, \beta\alpha, \alpha\delta, \beta\delta, \mu\}$, then $n_a = 1$ and $X_L = X_{SL} = \{0,3\}$, but $X_{EL} = \{0,1,2,3\}$. One of the difficulties in this case is that there are ambiguities in the parsing of strings that are eventually in L. For example, the string $\alpha\beta\alpha\beta$ can be given the following two parsings: (1) two occurrences of $w_1 = \alpha\beta$; or (2) an initial prefix of α, an occurrence of $w_2 = \beta\alpha$, and the initiation β of either another occurrence of w_2 or an occurrence of $w_4 = \beta\delta$. While a complete answer to the constraints on L_f under which $X_{EL} = X_{SL}$ for $L = (L_f)^{*c}$ remains open, there are some sufficient conditions that guarantee this. We present here one such condition, which, on the one hand, is more restrictive than necessary, but, on the other hand, is easily interpreted and should not be an unreasonable assumption in many applications.

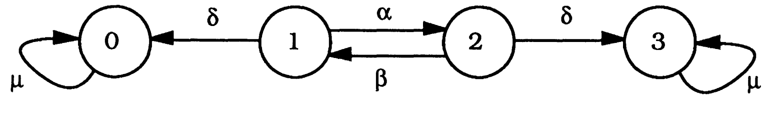

FIG. 4.9. *Example illustrating unequivalent X_{EL} and X_{SL}.*

[4] We are grateful to the authors of [5] for pointing out this subtlety to us.

PROPERTY 4.7. *A set of strings L_f has the* unique parsing *property if, for any string $s \in \Xi^*$, either s possesses no substring that is an element of L_f, or there is a* unique *way in which to write*

$$s = p_1 w_{i_1} p_2 w_{i_2} \cdots p_m w_{i_m} p_{m+1},$$

where $w_{i_1}, \cdots, w_{i_m}$ are (not necessarily distinct) elements of L_f, and none of the strings $p_1, \cdots, p_{m+1}$ contains a substring that is an element of L_f.

Note that, in general, this property need not be an easy one to check. There are, however, a number of important necessary conditions for this property to hold. In particular, no element of L_f can be a substring of another, and no prefix of one element of L_f can be a suffix of another (so that no word can be a cyclic permuattion of another). Note further that this condition does *not* hold for our manufacturing example, since the string $F_1F_2F_3F_4F_1F_2F_3F_4F_1$ can be thought of either as the word $F_1F_2F_3F_4F_1$ in L_S followed by the string $F_2F_3F_4F_1$ *or* as the string $F_1F_2F_3F_4$ followed by the word $F_1F_2F_3F_4F_1$ in L_S. One simple condition under which this property does hold is if there is either a unique special element of Ξ that only appears at the end of each element of L_f, indicating "task completion" (or, equivalently, a special element that appears only at the start of each element of L_f, indicating "task initiation"). For example, in our manufacturing example this would correspond to a simple modification to include explicitly a final event in each element of L_S corresponding to transferring the completed pair of boards to the final inspection station.

PROPOSITION 4.8. *Let $L = (L_f)^{*c}$, where $L_f = \{w_1, \cdots, w_m\}$ has the unique parsing property. Then $X_{EL} = X_{SL}$.*

Proof. As shown in [5], there is in general a (very large) upper bound on n_a and a finite-state compensator C such that $t(L(A_C, x)) \subset (\Xi \cup \{\epsilon\})^{n_a} L$ for all $x \in X_{EL}$. Suppose that $X_{EL} \neq X_{SL}$ and take any x_0 in $X_{EL} \setminus X_{SL}$, the complement of X_{SL} in X_{EL}. Thanks to unique parsing, any state reachable (in A_C) from a state in X_{EL} must also be in X_{EL}, so that $f(x_0, s) \subset X_{EL}$ for any $s \in L(A_C, x_0)$. Furthermore, since $x_0 \notin X_{SL}$, we can find a state path of arbitrarily long length beginning at x_0 that does not enter X_{SL}. By the finiteness of the composition of A and C, there then must exist a string in $L(A_C, x_0)$ that produces a cycle in the composite that stays in $X_{EL} \setminus X_{SL}$. That is, we can find a string p and s so that $ps^* \subset L(A_C, x_0)$, corresponding to a path completely within $X_{EL} \setminus X_{SL}$, where s represents the string of events around the cycle. Since $x_0 \in X_{EL}$ and since n_a is bounded, for k sufficiently large, $t(ps^k)$ contains a suffix in L. Let us first examine the case when $t(s)$ does not contain a substring in L_f. Then, thanks to eventual restrictability and to the finite length of words in L_f, we know that, for k sufficiently large, we must encounter strings of the form

$$ps^k = p \cdots u_1 w_{i_1} w_{i_2} \cdots w_{i_m} v_{m+1}, \tag{4.3}$$

where

(i) $w_{i_j} = v_j s^{n_j} u_{j+1}$,
(ii) $t(w_{i_j}) \in L_f$,
(iii) n_j is a nonnegative integer,
(iv) $s = u_i v_i$ for $i = 1, \cdots, m$

(see Fig. 4.10). Since s is a finite string and L_f is a finite set, it must be that, and such that, for some k and some $l < m$,

$$w_{i_l} = w_{i_m}. \tag{4.4}$$

In this case, since $v_m = v_l$ and $s = u_l v_l$, we see that

$$w_{i_l} w_{i_{l+1}} \cdots w_{i_{m-1}} = v_l s^{n_l} u_{l+1} \cdots v_{m-1} s^{n_{m-1}} u_l = \sigma^N, \tag{4.5}$$

where $\sigma = v_l u_l$ and $N = n_l + \cdots + n_{m-1} + m - l$. Note that (1) σ is simply a cyclic permutation of s, so that $\sigma \in L(A_C, y)$ for some $y \in X_{EL} \setminus X_{SL}$ on the cycle, and (2) $t(\sigma^N)$ is precisely a concatenation of strings in L_f. In the other case in which s contains a substring in L_f, we can still obtain a parsing as in (4.3) with the second condition changed to the statement that there exists a finite integer $r > 0$ so that $t(w_{i_j}) \in (L_f)^r$ (here we can take r as any integer greater than the ratio of the length of s to the length of the shortest element of L_f). Then, because of the finiteness of $(L_f)^r$, we can again find $l < m$, so that (4.4) and (4.5) hold, and thus so that the same two conditions hold for σ and y.

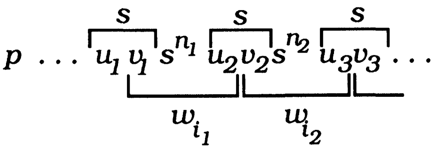

FIG. 4.10. *The parsing of* ps^k.

Consider then this state y. Since $y \notin X_L$, there is some string $d \in L(A_C, y)$ such that $t(d)$ is not in L. However, $(\sigma^N)^* d \subset L(A_C, y)$. By unique parsing, since σ^N is a sequence of elements of L_f, $t[(\sigma^N)^k d]$ *cannot* be an element of L for any k, since $t(d) \notin L$. On the other hand, since $y \in X_{EL}$ such strings *must* be in L as they are the suffixes of words of arbitrarily long length in $t[(\sigma^N)^* d]$ This establishes a contradiction to the assumption that $X_{EL} \setminus X_{SL}$ is nonempty. □

As we have indicated, the complete characterization of the relationship between X_{EL} and X_{SL} remains open. Several other less restrictive conditions on the structure of L are known that guarantee $X_{EL} = X_{SL}$, but none of these appear to have as simple and reasonable interpretation as Property 4.7. Completely open is the question of conditions on the automaton A (rather than the language L) that guarantee $X_{EL} = X_{SL}$. However, while these represent interesting research questions, it is our opinion that X_{SL} is a more meaningful object to begin with, since in essence for $x \in X_{EL} \setminus X_{SL}$ eventual restrictability happens in a sense by "accident." In contrast to $x \in X_{SL}$, we explicitly drive the system to X_L at which point we *know* that generation of strings in L commences. Thus, as briefly discussed in the §5, it is the notion of stable restrictability that plays a central role in [11] in our development of a theory of hierarchical aggregation based on the concept of task completion.

4.3. Reliability. Our final generalization of restrictability is very similar to the notion of resiliency introduced in [12]. 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. For example, in our manufacturing system, suppose that parts are detected to be defective as a kanban box arrives at W2. In particular, suppose that this "failure event" happens immediately after S_2 occurs from state (0,0,1,1) in Fig. 4.4(a). Then we would essentially observe a transition to state (2,0,0,0), since B1 suddenly becomes empty while W2 is still idle. To start production again, our goal at this point is to reach state (0,0,1,1). Note that we can stabilize (2,0,0,0) with respect (0,0,1,1), since all we must do is wait until T_2 finishes and the state transitions to (0,0,0,0), which we know is stable. Thus, in this case, we can recover from the initial failure within a few steps. We capture this recovery procedure as follows: 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. (We build this notion on the notion of stable restrictability.)

DEFINITION 4.9. 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 stably 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. Also, for all failure sequences $s \in L(A_C, x)$ with at most i failures, $f(x, s)$ is stably 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.

The class of i-reliably L-restrictable sets is closed under unions and intersections. Let X_R^i denote the maximal i-reliably L-restrictable set, and let $X_R^\infty \triangleq \bigcap_{i=1}^\infty X_R^i$. Note that $X_R^0 = X_{SL}$, where X_{SL} is defined for $A|\overline{\Xi}_f$. The following proposition is immediate.

PROPOSITION 4.10. *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 ∞.

It remains 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.,

$$Y^i = \{x \in X | d_f(x) = \emptyset \text{ or for all } \sigma \in d_f(y), f(y, \sigma) \subset X_R^i\}, \tag{4.6}$$

where $d_f(x) = d(x) \cap \Xi_f$. 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 $x \in X_R^0$. Second, we must have that any state that can be reached from x with one failure event must be stably L-restrictable with failure events turned off. To be precise, define

$$L_1(A, x) = \{s \in L(A, x) | \text{ only the last element of } s \text{ in } \Xi_f\}. \tag{4.7}$$

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 4.11.

LEMMA 4.11. *We have that $X_R^1 = R^0$, the maximal stably 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 stably 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, we 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 stable 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.

PROPOSITION 4.12. *X_R^{i+1} is the maximal stably L-restrictable subset of X^i in $A_{K^i}|\overline{\Xi}_f$.*

Proof. The proof is similar to the proof of Lemma 4.11. □

Thus the full recursive procedure is the following: (1) Compute X_R^0 using the stable restrictability results of the preceding section applied to $A|\overline{\Xi}_f$; (2) Given X_R^i, compute Y^i from (4.6); (3) Compute X^i and K^i using the (f, u)-invariance results discussed in §2; (4) Compute X_R^{i+1} using the stable restrictability results applied to $A_{K^i}|\overline{\Xi}_f$. Also, as a byproduct of the above construction, we obtain the following result.

COROLLARY 4.13. *It holds that $X_R^\infty = X_R^i$ for some $i \leq |Y'|$, where $Y' = \{x | d_f(x) \neq \emptyset\}$.*

Thus we can compute X_R^∞ in a finite number of steps, and, in fact, the complexity of this computation is $O(|Y'||X^{LA}|^2)$.

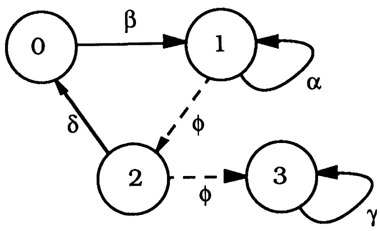

FIG. 4.11. *Reliable restrictability example: $\Xi_t = \{\alpha, \beta, \delta, \gamma\}$, $\Phi = \emptyset$, $\Xi_f = \{\phi\}$.*

As an example, consider the system in Fig. 4.11, 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. Also, $X_R^1 = \{0\}$. Thus state 0 can recover from a single failure. However, if the failure ϕ occurs at state 2, then a transition is made to state 3 that is *not* stably L-restrictable. Continuing, we obtain $X_R^2 = \emptyset$. Thus state 0 cannot recover from 2 or more failures.

5. Conclusions.

In this paper, we have investigated notions of tracking, restrictability, and reliability for discrete-event dynamic systems. We have developed algorithms for constructing trackable languages, testing restrictability and reliability, and constructing compensators for stable and reliable restriction of system behavior. As we have illustrated, the concepts arise naturally in operation sequence control in flexible manufacturing systems, and we expect that they will also prove to be relevant in a number of other contexts as well. The work in this paper complements our stability analysis in [16] 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 paper can be combined with our results on stabilization by output feedback in [14] to address problems of tracking and restrictability in the context of intermittent observations of events. As we have shown in [13] and [14], 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 the reason behind the NP-completeness of this problem [20] in the case of partial observations is the cardinality of the observer state space. Thus, if, in fact, the observer has polynomial state space (as it does in many cases [13]), 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 subsystems each with n states, then their composition may have n^m states. In this case, it would be extremely worthwhile to develop methods for obtaining aggregate models for each subsystem before addressing higher-level problems involving their interconnection. For example, consider again the manufacturing system of Fig. 1.2. Obviously, the "event" F1, corresponding to side 1 mounting, involves a sequence of commands for the surface mount workstation W1 (indeed, there may be several such sequences corresponding to mounting several different parts). Thus, at a lower level, we see that we have a restrictability problem for the control of each of the machines in Fig. 1.2, and this figure represents a higher-level version of each component system, aggregated to a level appropriate for the consideration of multi-machine coordination. An obvious question, then, concerns the problem of constructing higher-level models as in Fig. 1.2 from lower-level descriptions. In [11] we use the notions of restrictability and stable restrictability presented in this paper to develop such an hierarchical aggregation procedure based on the idea of transforming restricted event sequences at a lower level to single "task" events at higher levels. Obviously, we can also imagine performing such an aggregation procedure at a number of scales. For example, suppose that we have a set of schedules corresponding to different production operating points corresponding to distinct percentage of mixes of several computer boards. We can then construct compensators for implementing each, and eventual or stable restrictability will provide us with the means of changing the set-up from one schedule to another. Thus we can construct a higher-level model based on the set of all schedules by combining the respective compensators for each. Each occurrence of a higher-level event in this model would correspond to completing a cycle of some schedule, i.e., completing a certain number of each type of board. Then the plant manager could try to meet the actual demand distribution by switching between appropriate schedules based on this aggregate, higher-level model capturing operating behavior for all schedules.

REFERENCES

[1] P. E. Caines and S. Wang, *Classical and logic based regulator design and its complexity for partially observed automata*, in Proc. 28th Conference on Decision and Control, Tampa, FL, pp. 132–137.

[2] H. Cho and S. I. Marcus, *On the supremal languages of sublanguages that arise in supervisor synthesis problems with partial observations*, Math. Control Signals Systems, 2 (1989), pp. 47–69.

[3] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, *Supervisory control of discrete-event processes with partial observations*, IEEE Trans. Automat. Control, 33 (1988), pp. 249–260.

[4] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, Reading, MA, 1979.

[5] R. Kumar, V. Garg, and S. I. Marcus, *Language stability of deds*, in Proc. of Internat. Conference on Mathematical Theory of Control, Bombay, India, December 1990.

[6] F. Lin and W. M. Wonham, *Controllability and observability in the state-feedback control of discrete-event systems*, in Proceedings of 27th CDC, December 1988.

[7] ———, *Decentralized control and coordination of discrete-event systems*, in Proceedings of 27th CDC, December 1988.

[8] ———, *On observability of discrete event systems*, Inform. Sci., 44 (1988), pp. 173–198.

[9] J. S. Ostroff and W. M. Wonham, *A temporal logic approach to real time control*, in Proceedings of 24th CDC, December 1985.

[10] C. M. Özveren, *Analysis and control of discrete event dynamic systems: A state space approach*, Ph.D. thesis, MIT, Cambridge, MA, August 1989; Laboratory for Information and Decision Systems Report LIDS-TH-1907, MIT.

[11] C. M. Özveren and A. S. Willsky, *Aggregation and multi-level control in discrete event dynamic systems*, Automatica, May 1992.

[12] ———, *Invertibility of discrete event dynamic systems*, Math. Control Signals Systems, 1992.

[13] ———, *Observability of discrete event dynamic systems*, IEEE Trans. Automat. Control, 35 (1990), pp. 797–806.

[14] ———, *Output stabilizability of discrete event dynamic systems*, IEEE Trans. Automat. Control, 35 (1990), pp. 797–806.

[15] ———, *Applications of a regulator theory for discrete event dynamic systems*, in Proceedings of IFAC Distributed Intelligence Systems Symposium, Arlington, VA, August 1991, pp. 925–935.

[16] C. M. Özveren, A. S. Willsky, and P. J. Antsaklis, *Stability and stabilizability of discrete event dynamic systems*, J. Assoc. Comput. Mach., 38 (1991), pp. 730–752.

[17] P. J. Ramadge, *Some tractable supervisory control problems for discrete event systems modeled by buchi automata*, IEEE Trans. Automat. Control, 36 (1989), pp. 10–19.

[18] P. J. Ramadge and W. M. Wonham, *Modular feedback logic for discrete event systems*, SIAM J. Control Optim., 25 (1987), pp. 1202–1218.

[19] ———, *Supervisory control of a class of discrete event processes*, SIAM J. Control Optim., 25 (1987), pp. 206–230.

[20] J. N. Tsitsiklis, *On the control of discrete event dynamical systems*, Math. C. S. S., 1989.

[21] A. F. Vaz and W. M. Wonham, *On supervisor reduction in discrete event systems*, Internat. J. Control, 44 (1986), pp. 475–491.

[22] W. M. Wonham and P. J. Ramadge, *On the supremal controllable sublanguage of a given language*, SIAM J. Control Optim., 25 (1987), pp. 637–659.