Stability and Stabilizability of Discrete Event Dynamic Systems

CÜNEYT M. ÖZVEREN AND ALAN S. WILLSKY

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

PANOS J. ANTSAKLIS

University of Notre Dame, Notre Dame, Indiana

Abstract. A finite-state automaton is adopted as a model for Discrete Event Dynamic Systems (DEDS). Stability is defined as visiting a given set E infinitely often. Stabilizability is defined as choosing state feedback such that the closed loop system is stable. These notions are proposed as properties of resiliency or error-recovery. An important ingredient in stability is shown to be a notion of transition-function-invariance. Relations between our notions of stability and invariance, and the notions of safety, fairness, livelock, deadlock, etc., in computer science literature are pointed out. Connections are established between our notions of invariance and the classical notions of A-invariance and (A, B)-invariance of linear systems. Polynomial algorithms for testing stability and stabilizability, and for constructing a stabilizing control law are also presented.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*computations on discrete structures*; *sequencing and scheduling*; F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages—*algebraic language theory*; *classes defined by grammars or automata*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms*; G.4 [**Mathematical Software**]: Algorithm Analysis; *reliability and robustness*; H.2.8 [**Database Management**]: Database Applications; J.7 [**Computers in Other Systems**]: Command and Control; *process control*

General Terms: Algorithms, Design, Languages, Reliability, Theory

Additional Key Words and Phrases: Reliability, self-stabilizing systems, stability, stabilizability, state feedback

---

Research was supported by the Air Force Office of Science Research under grant AFOSR 88-0032 and by the Army Research Office under grant DAAL03-86-K0171.

The research of C. M. Özveren and A. S. Willsky was partially done during their stay at Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), Rennes, France, and A. S. Willsky was also supported by IRISA during this time.

Authors addresses: C. M. Özveren and A. S. Willsky, Laboratory for Information and Decision Systems, MIT, Cambridge, MA 02139; P. J. Antsakis, University of Notre Dame, Notre Dame, IN 46556.

1. *Introduction*

Discrete Event Dynamic Systems (DEDS) have received considerable attention in the control literature recently. Many large-scale dynamic systems seem to have a DEDS structure, at least at some level of description. Some examples are manufacturing systems [12, 21], communication systems (such as data networks, and distributed systems) [3], and expert systems (such as CPU design, or air-traffic management) [7, 8, 23].

The notion of the control of a DEDS was, to our knowledge, first explicitly introduced in the work of Wonham, Ramadge, et al. [10, 13, 18, 19, 24, 26]. 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, so 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, that is, the set of possible strings of events.

This work 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 had in mind the development of the elements needed for a regulator theory for DEDS. In this paper, we develop notions of stability and stabilizability for DEDS that might, more concretely, be thought of as properties of resiliency or error-recovery.

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, what we seek here is to develop control methods for reestablishing 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, that is, 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." In some of our other work [15, 16], we focus on notions of tracking desired sets of strings, which can be used as tools for identifying the set of "good" states, that is, those states from which one can generate the desired behavior. Furthermore, the notion of stability presented in this paper plays an important role in formulating and testing observability (see [17]), that is, the ability to reconstruct state knowledge when only the occurrence of particular events are available as directly observed quantities.

Another goal of this work is to establish connections with the 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 prestability and the algorithm we provide is exactly the same as the notion of inevitable reachability and the algorithm of Sifakis [20]. Other notions of Sifakis can be characterized using stability and transition-function-invariance (f-invariance of Section 2). Thus, our results in stabilizability can be directly applied to his notions if control was to be introduced in his framework. In [6], 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. Finally, fair execution sequences, in concurrent systems, are defined as those execution sequences in which each process is executed infinitely often [4]. This notion is also connected to our notion of stability and our results on stabilizability can tell us how to achieve fairness.

In the next section, we introduce the mathematical framework considered in this paper and address the problem of stability. In particular, we first introduce a notion of prestability, which is based on testing if all trajectories from a state go through the "good" states. We then define transition-function-invariance (f-invariance) in our framework and characterize stability in terms of prestability and f-invariance. We also present algorithms for testing prestability and stability. In Section 3, we address the problem of stabilizability. We first present an algorithm to construct a prestabilizing state feedback, and we classify different prestabilizing feedbacks by the degree of restriction imposed on the behavior. As an extension of f-invariance, we examine achieving f-invariance by control inputs and we then impose the constraint that f-invariance is achieved while keeping the system alive. We combine this with prestabilizability and present an algorithm to construct a stabilizing state feedback. Finally, in Section 4, we summarize our results and outline further research directions.

2. *Stability*

In this section, we define our notion of stability and provide an algorithm that tests stability.

2.1. Preliminaries. The class of systems we consider are nondeterministic finite-state automata. The basic object of interest is:

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

where X is the finite set of states, with $n = |X|$, and Σ is the finite set of possible events. The dynamics of the system are characterized by two functions f and d:

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

$$\sigma[k+1] \in d(x[k]). \tag{2.3}$$

Here $x[k] \in X$ is the state after the kth event, and $\sigma[k+1] \in \Sigma$ is the $(k+1)$st event. The function d: $X \to 2^{\Sigma}$ is a set valued function that specifies the set of possible events defined at each state (so that, in general, not all events are possible from each state), and the function f: $X \times \Sigma \to X$ is also set valued, so that the state following a particular event is not necessarily known with certainty. The triple $A = (G, f, d)$ representing our system can also be visualized graphically as in Figure 1. Here circles denote states, and arcs denote transitions, where the symbol in each arc label denotes the event corresponding to that transition. Thus, in this example, $X = \{0, 1, 2, 3\}$, $\Sigma = \{\alpha, \beta, \delta\}$, and, for example, $d(1) = \{\alpha, \delta\}$, $f(0, \beta) = \{0, 3\}$, etc. A transition, also denoted as $x \to^{\sigma} y$, consists of a source state, x, an event, $\sigma \in d(x)$, and a destination state, $y \in f(x, \sigma)$.

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 $i = 0 \cdots j - 1$, where

$$f(x, d(x)) = \bigcup_{\sigma \in d(x)} f(x, \sigma).$$

We say $x \in \mathbf{x}$ if $x_i = x$ for some i. Let $\mathscr{X}(A, x)$ 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 $x_{i_1} = x_{i_2}$, that is, if it contains no other cycles. For example, in Figure 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 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 $f(x, \epsilon) = x$, where ϵ is the empty string. In Figure 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.

For most applications of interest to us, it is desired that the system can never reach a point at which no event is possible. This is a notion of liveness. For example, a manufacturing system should, always, be capable of producing something. We use

$$R(A, x) = \{y \in X \mid x \to^* y\}$$

to represent the set of states that can be reached from x, where $\to^*$ denotes any number of transitions, including no transitions. Thus, $R(A, x)$ always includes x. For example, in Figure 1, the reach of 1 is X itself. Intuitively, we define a state to be alive if all event trajectories from that state have infinite length extensions. We let

$$D = \{x \in X \mid d(x) = \varnothing\} \tag{2.4}$$

denote the set of states which have no events defined, and term these the *dead* states. We formally define liveness as follows:

Definition 2.1. A state $x \in X$ is *alive* if $d(y) \neq \varnothing$ 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.

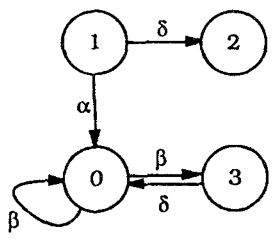

FIG. 1. Simple example.

For example, in Figure 1, $D = \{2\}$ and 0 is alive, whereas 1 is not. Clearly, the class of live sets is closed under arbitrary unions and intersections. The maximal live subset of X, X_a, is given by the set of states that *cannot* reach the dead states:

$$X_a = \overline{R(A^{-1}, D)}, \tag{2.5}$$

where overline denotes the complement, $R(A^{-1}, D) = \bigcup_{x \in D} R(A^{-1}, x)$, and A^{-1} denotes A with the transitions reversed, that is, $A^{-1} = (G, f^{-1}, d^{-1})$ where:

$$f^{-1}(x, \sigma) = \{y \in X \mid x \in f(y, \sigma)\}, \tag{2.6}$$

$$d^{-1}(x) = \{\sigma \in \Sigma \mid \exists y \in X \text{ such that } x \in f(y, \sigma)\}. \tag{2.7}$$

Thus, $R(A^{-1}, D)$ is the set of states that can reach D in the original system. For example, in Figure 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.

Note that if we replace D by any set of states, then X_a is the maximal set of states that avoid D. This situation arises, for example, in mutual exclusion problems, in the context of manufacturing systems [11] and computer systems [1, 2, 5], 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 set of states that avoid this violation. Golazewski and Ramadge [9] address this problem, in the context of DEDS problems that consist of many interacting components. Our contributions to this problem are presented in [14].

We conclude this section by presenting an algorithm to compute the reach of a set of states. It immediately follows from the definition that the reach of a set of states, X_0, is the fixed point of $R = f(R, \Sigma) = \bigcup_{x \in R} f(x, d(x))$ 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 $R_{k+1} = R_k$.

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

2.2 PRESTABILITY. The notion of stability we wish to capture can be thought of as error recovery. Specifically, as in Wonham and Ramadge, one can imagine a set of desired event trajectories that one would like to see in 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 nonfailure states or it might simply be the set of start-up states, which, in the Wonham and Ramadge framework, can be thought of as the initial state 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

$$L(A, E) = \bigcup_{x \in E} L(A, x). \tag{2.8}$$

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 trajectory by at most a finite prefix.

Given E, we define a state $x \in X$ to be *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 2, where $E = \{0, 3\}$, 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 [22]: An infinite event trajectory is Büchi acceptable if there exists a corresponding state trajectory which visits a terminal state (where a set of terminal states is given) infinitely often. In our notion of stability, if we let E be the set of terminal states, *all* possible state trajectories, from a stable state, visit E infinitely often.

Our notion of stability is captured in two stages. We term x prestable 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 2, state 0, 2, 3, and 4 are prestable. This notion is exactly the same as the notion of inevitable-reachability of Sifakis [20]. A state is then stable if all the states in its reach are prestable. In Figure 2, state 0 and 4 are not stable since they can reach 1, which is not prestable.

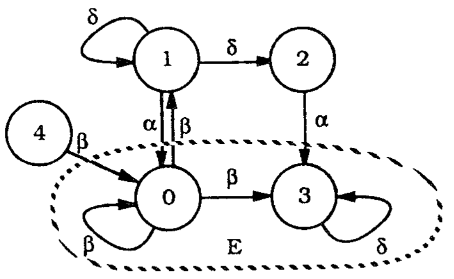

FIG. 2. Stability Example.

We formalize prestability as follows:

Definition 2.3. Given a live system A and some $E \subset X$, a state $x \in X$ is *prestable* with respect to E if for all $\mathbf{x} \in \mathscr{X}(A, x)$ 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 prestable with respect to E if all its elements are prestable and a system A is prestable with respect to E, if all states in X are prestable.

If all paths from x go through E, then they do so in less than n transitions since, otherwise, x has a cycle that does *not* go through E, and thus, there exists a path that never goes through E. Equivalently, prestability 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 2.2, $R_E(A, 4) = \{0, 3, 4\}$. 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 2, we only remove the transition $0 \rightarrow^{\beta} 1$. Then, x is prestable if and only if all primary cycles in $R_E(A, x)$ go through E. In Figure 2, the self-loop at 1 is a primary cycle and does not go through E.

PROPOSITION 2.4. *Given a live A and $x \in X$, x is prestable with respect to E iff all primary cycles in $R_E(A, x)$ include at least one state in E. In general, A is prestable with respect to E 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 prestable with respect to E, is closed under arbitrary unions and intersections. Now, we derive an algorithm that computes X_P, the maximal subset of X that is prestable with respect to E. Our algorithm, which is the same as the one given for inevitable reachability in Sifakis [20], is based on starting from E and growing the currently known set of prestable 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 prestable, then either x is in E or all the events defined from x take x to a prestable state:

LEMMA 2.5. *$x \in X$ is prestable with respect to E iff $x \in E$ or $f(x, d(x))$ is prestable with respect to E.*

Proof. Straightforward. □

Also, note that given a set of prestable states Q, that include E, we can test prestability of other states by testing prestability with respect to Q:

Lemma 2.6. *Given Q, $Q_2 \subset X$ such that Q_1 is prestable with respect to E and $Q_1 \supset E$, Q_2 is prestable with respect to E iff Q_2 is also prestable with respect to Q_1.*

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 2.4, this cycle goes through some state $x_1 \in Q_1$. Then, x_1 cannot be prestable with respect to E, and we have a contradiction. □

These two lemmas lead to the following algorithm:

Proposition 2.7. *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 $X_{k+1} = X_k$.

Proof. Clearly, X_0 is prestable with respect to E. Suppose that X_k is prestable with respect to E. By Lemma 2.5, X_{k+1} is prestable with respect to X_k and by Lemma 2.6, it is also prestable with respect to E. 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 exists $\sigma \in d(x_1)$ 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 that never reaches X_r. Since also $X_r \supset E$, x_1 is *not* prestable with respect to E 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 2, $X_1 = X_2 = X_P = \{0, 2, 3, 4\}$. Note that the number of steps in which this algorithm terminates is a notion of radius for the prestable 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 prestable state and the state in E at which this trajectory enters E for the first time. In Figure 2, 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.

2.3 Stability and f-Invariance. As motivated in the previous sections, we define stability as follows:

Definition 2.8. Given a live system A and some $E \subset X$, a state $x \in X$ is *stable* with respect to E, if all infinite state trajectories starting from x visit E infinitely often. More precisely, x is stable if for all $\mathbf{x}_1, \mathbf{x}_2 \in \mathscr{X}(A, x)$ 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 revisit E within a finite number of transitions. In Figure 2, 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 prestability of its reach:

PROPOSITION 2.9. *Given a live A and $x \in X$, x is stable with respect to E iff $R(A, x)$ is prestable with respect to E.*

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 2.10. *A is a system stable with respect to E iff it is also prestable with respect to E.*

We also have the following counterpart of Proposition 2.4:

PROPOSITION 2.11. *Given a live A and $x \in X$, x is stable with respect to E iff all primary cycles in $R(A, x)$ include at least one state in E. In general, A is stable with respect to E iff all primary cycles in X include at least one state in E.*

PROOF. Straightforward. □

If we compare this to Proposition 2.4, note that the second statements are exactly the same. This is due to Proposition 2.10.

The class of sets stable with respect to E is closed under arbitrary unions and intersections. Let X_S denote the maximal set stable with respect to E. (Note that X_S can be the empty set, for example, if we let $E = \{0\}$ in Figure 2.) Then, X_S is the set of states in X_P from which we can only reach prestable 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 2.12. 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 2.13. *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 2.14. *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 it gets out of E it stays in X_f and thus in X_{P}. Therefore, X_f is stable. $\square$

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

$$X_{\mathrm{S}} = \overline{R(A^{-1}, \overline{X_P})}. \tag{2.9}$$

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 prestabilizability (resp., stabilizability) as finding a state feedback such that the closed loop system is prestable (stable). We present a polynomial algorithm for constructing a prestabilizing feedback. This algorithm is a natural extension of Algorithm 2.7 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 prestable 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.

3.1 Prestabilizability. To introduce control, we modify our system model as follows:

$$G = (X, \Sigma, U), \tag{3.1}$$

where, U is the set of admissible control inputs. We introduce control by allowing certain events at each state to be disabled. This framework is the same as that of Ramadge and Wonham, except that in our case, an event that is controllable at one state may not be controllable at another state. We let $U = 2^{\Sigma}$ and the dynamics are described by:

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

$$\sigma[k+1] \in (d(x[k]) \cap u[k]) \cup e(x[k]), \tag{3.3}$$

where, $u[k] \in U$ is the control input after the kth event, and e: $X \to 2^{\Sigma}$ is a set valued function that specifies the set of events that *cannot* be disabled at each state. Without loss of generality, we assume that $e(x) \subset d(x)$ for all x. The interpretation of (3.3) is straightforward. The set $d(x)$ represents an "upper bound" on the set of events that can occur at state x—no matter what we do, the next event will be in $d(x)$. The set $e(x)$, on the other hand, is a lower bound—no matter what we do, any event in $e(x)$ may still occur. The effect of our control action is adjusting the set of possible events between these bounds, by disabling some of the controllable events, that is, elements of the set $d(x) \cap \overline{e(x)}$. Note, therefore, that while in our definition $u(x)$ can be any subset of Σ, we can, without loss of generality assume that $u(x) \subset d(x)$

$\cap \overline{e(x)}$. We will make this assumption throughout. The quadruple $A = (G, f, d, e)$ representing our system can also be visualized graphically as in Figure 3. In addition to the graphical representation of the previous section, we mark the controllable events by "$:u$". For example, in Figure 3, γ is controllable at 1 and 2, whereas β is controllable only at 3.

A state feedback law is a map K: $X \to U$. Given a state feedback K, let $A_K = (G, f, d_K, e)$ denote the closed loop system where

$$d_K(x) = \big(d(x) \cap K(x)\big) \cup e(x). \tag{3.4}$$

We define prestabilizability as follows:

Definition 3.1. Given a live system A and some $E \subset X$, $x \in X$ is *prestabilizable* with respect to E if there exists a state feedback K such that x is alive and prestable with respect to E in A_K. A set of states, Q, is a *prestabilizable set* if there exists a feedback law K so that every $x \in Q$ is alive and prestable in A_K, and A is a *prestabilizable system* if X is a prestabilizable set.

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

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

Proposition 3.2. *Given prestabilizable sets Q_1 and Q_2, $Q_1 \cup Q_2$ is also prestabilizable.*

Proof. We show this by constructing a feedback that prestabilizes the union. First let K_i prestabilize 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 4). 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 prestable in the closed loop system A_F. By Lemma 2.6, Q_2 is also prestable, 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 prestabilizes $Q_1 \cup Q_2$. □

We immediately have the following corollary:

Corollary 3.3. *A maximal set that is prestabilizable with respect to E exists. Let $P(E)$ denote this set.*

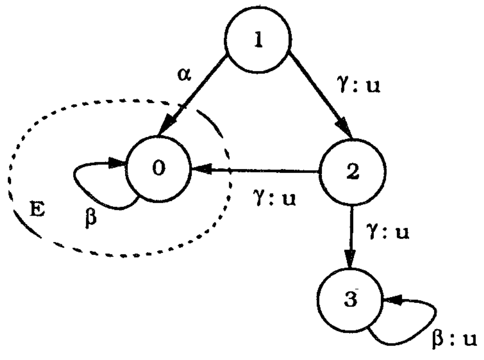

FIG. 3. Example for the notion of prestabilizability.

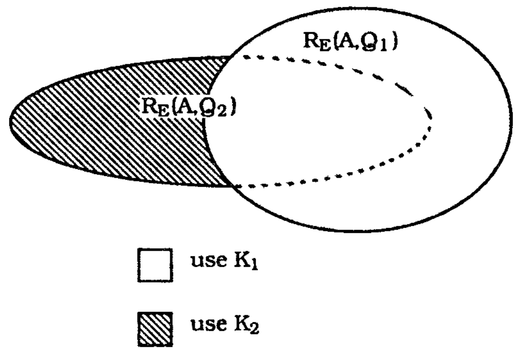

FIG. 4. Feedback that prestabilizes the union of two sets.

Recall, from Lemma 2.5, that a necessary and sufficient condition for the prestability of a state x is the prestability 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, that is, $d_K(x) = (d(x) \cap K) \cup e(x) \neq \emptyset$, and all those events take x to prestabilizable states, that is, $f(x, d_K(x))$ is a prestabilizable set:

LEMMA 3.4. *$x \in X$ is prestabilizable with respect to E 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 prestabilizable with respect to E.*

PROOF

($\rightarrow$) Immediate from Lemma 2.5.

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

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

Then, also by Lemma 2.6, K_2 prestabilizes 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. □

Now, we can construct a natural counterpart of Algorithm 2.7 using the above lemma. As in Algorithm 2.7, we start with E and then add in the states that satisfy Lemma 3.4. 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 prestabilizable, 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 5, we start with the state 0. At the first step, we include 1 and 2, and at the second step we include 3.

PROPOSITION 3.5. *The following algorithm computes $P(E)$ and a feedback that prestabilizes it. It has complexity $O(n^2)$:*

Algorithm

```
Let $X_0 = E$ and iterate:
    $P_{k+1} = \left\{ \begin{array}{l} x \mid (e(x) \neq \emptyset \text{ and } f(x, e(x)) \subset X_k) \text{ or} \\ \quad (e(x) = \emptyset \text{ and } \exists \sigma \in d(x) \text{ such that } f(x, \sigma) \subset X_k) \end{array} \right\}$
    $K(x) = \left\{ \begin{array}{ll} \emptyset & \text{if } e(x) \neq \emptyset, \\ \text{some } \sigma \text{ such that } f(x, \sigma) \subset X_k & \text{otherwise}, \end{array} \right\} \quad \text{for } x \in P_{k+1}$
    $X_{k+1} = X_k \cup P_{k+1}$.
Terminate when $X_{k+1} = X_k$.
```

PROOF. Straightforward by following the proof of Proposition 2.7 and using Lemma 3.4. □

In Figure 5, 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.

We say that a feedback law K is *prestabilizing* if $P(E)$ is prestable in A_K. Algorithm 3.5 produces one such feedback and, in fact, leads to a feedback that disables as many events as possible. We formalize this as follows:

Definition 3.6. A prestabilizing feedback K is *maximally restrictive*, if for any prestabilizing feedback K' such that $d_{K'}(x) \subset d_K(x)$ for all $x \in P(E) \cap \overline{E}$, then $K'(x) = K(x)$ for all $x \in P(E) \cap \overline{E}$.

We immediately have the following result:

PROPOSITION 3.7. *A prestabilizing feedback K is maximally restrictive iff*

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

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

PROOF. Straightforward. □

Thus, Algorithm 3.5 leads to a maximally restrictive feedback.

The feedback of Algorithm 3.5, also minimizes the maximum number of transitions it takes to go from a state to E. Clearly, it also minimizes the radius. In Figure 5, 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, $r(A, x)$ for $x \in P(E)$ 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 3.8. A prestabilizing state feedback K is *path minimizing* if for any prestabilizing feedback K' such that $r(A_{K'}, x) \leq r(A_K, x)$ for all $x \in P(E)$, $r(A_{K'}, x) = r(A_K, x)$ for all $x \in P(E)$.

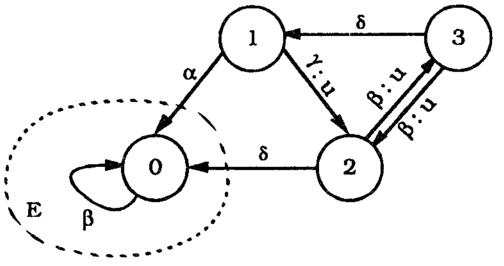

FIG. 5. Example for the prestabilizability algorithm.

For any two path-minimizing feedbacks, the longest path lengths are equal at each state:

PROPOSITION 3.9. *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. In this case, the set $S = \{x \in P(E) \mid r(A_{K_1}, x) \neq r(A_{K_2}, x)\}$ is nonempty. Let r_0 denote the smallest value of r at which we see such a discrepancy, that is,

$$r_0 = \min_{x \in S} \left\{ \min\left[r(A_{K_1}, x), r(A_{K_2}, x) \right] \right\}$$

and let $x_0 \in S$ be a point at which this minimum is achieved. Without loss of generality, we assume that

$$r_0 = r(A_{K_1}, x_0) < r(A_{K_2}, x_0).$$

What we now show is that K_2 cannot be path-minimizing yielding the required contradiction and thus proving the proposition.

Define the feedback law

$$F(x) = \begin{cases} K_1(x) & \text{if } x \in R_E(A_{K_1}, x_0), \\ K_2(x) & \text{otherwise}. \end{cases}$$

Let us first show that F is a prestabilizing feedback. Obviously, every $x \in R_E(A_{K_1}, x_0)$ is prestable in A_F, since the trajectory starting from such a state until it reaches E is the same in A_F and A_{K_1}. Consider then a trajectory in A_F starting some $x \in P(E) \cap \overline{R_E(A_{K_1}, x_0)}$. Either this trajectory does not pass through $R_E(A_{K_1}, x_0)$ or it does. If it does not, then the trajectory is the same as in A_{K_2} and thus passes through E. If it does, then once it enters $R_E(A_{K_1}, x_0)$ we have already seen that the subsequent trajectory is as in A_{K_1} and therefore also passes through E.

All that remains to show is that $r(A_F, x) \leq r(A_{K_2}, x)$ for all $x \in P(E)$ and that this inequality is strict for at least one such x. By construction, we have that

$$r(A_F, x_0) = r(A_{K_1}, x_0) = r_0 < r(A_{K_2}, x_0).$$

Also, consider any $x \in R_E(A_{K_1}, x_0)$, $x \neq x_0$. Then, since $r(A_{K_1}, x) < r(A_{K_1}, x_0) = r_0$, from the definition of r_0 and F we have that

$$r(A_F, x) = r(A_{K_1}, x_0) = r(A_{K_2}, x).$$

Consider next any $x \in P(E) \cap \overline{R_E(A_{K_1}, x_0)}$ and a trajectory in A_F that achieves $r(A_F, x)$. If this path does not pass through $R_E(A_{K_1}, x_0)$ before it reaches E, then obviously $r(A_F, x) = r(A_{K_2}, x)$ since the trajectory in A_F is the same as is in A_{K_2}. If the path does pass through $R_E(A_{K_1}, x_0)$, let y denote the first point in $R_E(A_{K_1}, x_0)$ along the path. Then

$$\begin{aligned} r(A_F, x) &= \text{length of path from } x \text{ to } y + r(A_F, y), \\ &= \text{length of path from } x \text{ to } y + r(A_{K_1}, y), \\ &\leq \text{length of path from } x \text{ to } y + r(A_{K_2}, y), \\ &= r(A_{K_2}, x), \end{aligned}$$

where the last equality follows from the fact that the part of the trajectory from x to y is the same in A_F and A_{K_2}. This completes the proof. □

Let $\bar{r}(x)$, $x \in P(E)$ denote $r(A_K, x)$ for a path-minimizing feedback K. Finally, we show that Algorithm 3.5 leads to a path-minimizing feedback, and it also constructs $\bar{r}(x)$:

Proposition 3.10. *The feedback of Algorithm* 3.5, K, *is path minimizing. Furthermore,* $x \in P_k$ *at step* k *of the algorithm, if and only if* $k = \bar{r}(x)$, *and thus, the algorithm also constructs* $\bar{r}(x)$.

Proof. Suppose that the prestabilizing feedback K as constructed in Algorithm 3.5 is path minimizing for the set X_k as defined in the algorithm, that is, for any other prestabilizing feedback K' such that $r(A_{K'}, x) \leq r(A_K, x)$ for all $x \in X_k$, then $r(A_{K'}, x) = r(A_K, x)$ for all $x \in X_k$. If $X_k = X_{k+1}$, then $X_k = P(E)$ and we are done. If $X_k \neq X_{k+1}$, there exists an $x \in P_{k+1}$ such that $f(x, d_K(x)) \cap P_k \neq \emptyset$ since otherwise $x \in P_i$ for some $i < k + 1$. Therefore, K is path minimizing for X_{k+1}. Since K is trivially path minimizing for $X_0 = E$, we have by induction that 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 (e.g., 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 3.11. A prestabilizing feedback K is *minimally restrictive*, if for any prestabilizing feedback K' such that $d_{K'}(x) \supset d_K(x)$ for all $x \in P(E) \cap \overline{E}$, then $K'(x) = K(x)$ for all $x \in P(E) \cap \overline{E}$.

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, that is, 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 5, consider the feedback which disables γ at 1 and β at 3. Enabling either of these events makes the corresponding state unstable.

LEMMA 3.12. *A prestabilizing 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 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 5, 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 5, if we start with state 1 or 2, we get the minimally restrictive feedback that 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 3.13. *Given $x \in \overline{E}$ 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 3.14. *The following algorithm computes a minimally restrictive feedback. It has complexity $O(n^2)$:*

Algorithm

```
For all $x \in P(E)$ do:
    $S = K(x) \cup \{\sigma \in d(x) \cap \overline{K(x)} \cap \overline{e(x)} \mid f(x,\sigma) \cap R_E(A_K^{-1}, x) = \emptyset\}$,
    $K(x) = S$.
```

PROOF. The proof follows from Lemma 3.12. The complexity is $O(n^2)$ since all states are visited and the reach operation has complexity $O(n)$. □

3.2 STABILIZABILITY AND (f, u)-INVARIANCE. Stabilizability, like prestabilizability, 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 3.15. Given a live system A and some $E \subset X$, $x \in X$ is *stabilizable* with respect to E if there exists a state feedback K such that x is alive and stable with respect to E in A_K. A set of states, Q, is a *stabilizable set* if there exists a feedback law K so that every $x \in Q$ is alive and stable in A_K, and A is a *stabilizable system* if X is a prestabilizable 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 3.16. *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:

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

F clearly stabilizes $Q_1 \cup Q_2$. □

We immediately have the following corollary:

Corollary 3.17. *A maximal set that is stabilizable with respect to E 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 3.18. 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, that is, $A_\emptyset = (X, f, e, e)$. 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 [25] that a subspace $\mathscr{V}$ is (A, B)-invariant iff $A\mathscr{V} \subset \mathscr{V} + \mathscr{B}$, where $\mathscr{B}$ is the range of B (compare to item 2 below):

Proposition 3.19. *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 → 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.

($2 \to 3$) Assume the contrary. By Proposition 2.13, 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.

($3 \to 1$) 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 exists, and let $T(Q)$ denote this set. Clearly, $T(Q) = \overline{R(A_{\varnothing}^{-1}, \overline{Q})}$. 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 that requires that we can find a state feedback such that Q is both alive and f-invariant in the closed loop system:

Definition 3.20. 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 6, Q_1 (resp., Q_2) can be made (f, u)-invariant and alive by disabling δ (resp., 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.

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 3.21. *Given* $Q \subset X$, *let*

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

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

Proof

($\to$) 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 $K(x) = \{\sigma \mid f(x, \sigma) \subset Q\}$. □

This result implies that in order to find $I(Q)$, we can apply the operation

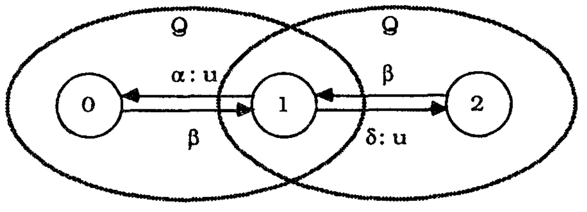

FIG. 6. Example for the sustainable (f, u)-invariance of unions and intersections.

T iteratively by throwing away the states that are no longer alive after the last step:

PROPOSITION 3.22. *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 $X_{k+1} = X_k$.

PROOF. Clearly, this algorithm terminates in a finite number of steps, say r steps. By Lemma 3.21, $X_r \subset I(Q)$. 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 prestabilizable set with respect to E. If $P(E)$ were sustainably (f, u)-invariant, we could be 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 prestabilizable, 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 that were prestabilizable with respect to E but *not* with respect to E', and we therefore must repeat the process. For example, in Figure 7, $E = \{0, 3\}$, $P(E) = \{0, 1, 2, 3\}$, and $I(P(E)) = \{1, 2, 3\}$, so that $E' = \{3\}$. Now, $\{1\}$ is *not* prestabilizable with respect to E' 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 3.23. *Given $E, Q \subset X$, let*

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

If $Q' = Q$, then Q is stabilizable with respect to E.

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

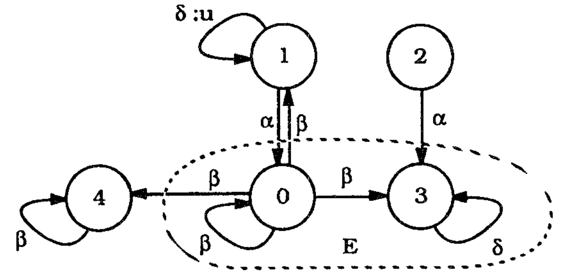

FIG. 7. Example for the stabilizability algorithm.

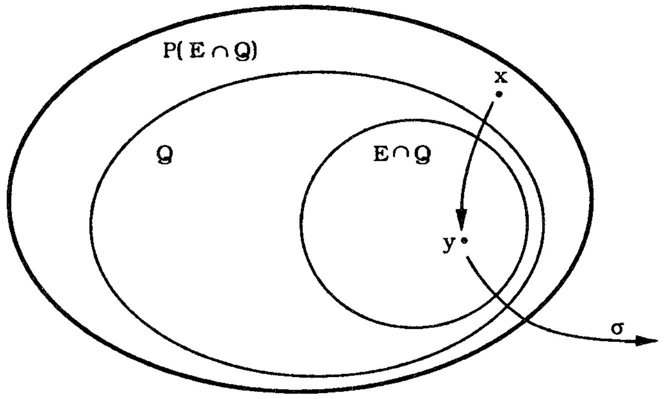

FIG. 8. Illustration for the proof of Lemma 3.23.

exists some $x \in P(E \cap Q)$ such that $x \notin Q$ (see Figure 8). 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 $f(y, \sigma) \not\subset P(E \cap Q)$. 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.

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

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

Q is clearly alive in A_K. Q is also prestable in A_K since K_P insures that all paths from states in $Q \cap \bar{E}$ 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 3.24. *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 $X_{k+1} = X_k$.

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 3.23, $X_r \subset S(E)$. 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 prestability at step r, say F, also achieves stability for $S(E)$, since states in $\overline{S(E)}$ are not prestable with respect to $E \cap S(E)$ 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

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

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 prestability and another for invariance. Clearly, there is no flexibility in choosing the feedback to achieve invariance (events that do *not* take the state out of P_{k+1} should *not* be disabled of course). For prestability, either a minimally restrictive or a maximally restrictive feedback can be chosen. Then, the corresponding stabilizing feedback is, accordingly, minimally restrictive or maximally restrictive.

4. *Conclusions*

In this paper, 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, a feedback control law that makes a stabilizable set or system stable. Our work has drawn on a blend of concepts from computer science and from dynamic systems and control. In particular, the notion of prestability 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 control concepts in systems and control.

The stability concepts that we introduced here can be thought of as notions of error recovery or resiliency in that the system always returns to "good" states. From the control perspective, one can also formulate several related concepts and problems. For example, in many applications, one may not have (or want) access to full state or event information but may still wish to stabilize the system. This leads directly to questions of state reconstruction, observability, and output feedback. Also, motivated by problems such as schedule-following in a flexible manufacturing system, one can 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. Analysis addressing these and related problems will be the subjects of subsequent papers.

ACKNOWLEDGMENTS. The authors wish to acknowledge Professors George C. Verghese and Gilead Tadmore for their insightful comments in a preliminary phase of this work. Professor Willsky and Dr. Özveren also wish to acknowledge Institut de Recherche en Informatique et Systèmes Aléatoires (INRIA), Rennes, France, for their support during our stay, and Professor Jean-Claude Raoult, and Drs. Albert Benveniste, Paul Le Guernic, and Bernard Le Goff for their comments on this work.

REFERENCES

1. Ben-Ari, M. *Principles of Concurrent Programming*. Prentice-Hall, Englewood Cliffs, N.J., 1982.
2. Bochmann, G. V. *Distributed System Design*. Springer-Verlag, New York, 1983.
3. Cieslak, R., Desclaux, C., Fawaz, A., and Varaiya, P. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Automatic Contr. 53*, 3 (Mar. 1988), 249–260.
4. Clarke, E. M., and Grumberg, O. Research on automatic verification of finite-state concurrent systems. Tech Rep. CMU-CS-87-105. Carnegie-Mellon Univ., Pittsburgh, Pa., Jan. 1987.
5. Dijkstra, E. W. Solution of a problem in concurrent programming control. *Commun. ACM 5*, 9 (Sept. 1965), 569–570.
6. Dijkstra, E. W. Self-stabilizing systems in spite of distributed control. *Commun. ACM 17*, 11 (Nov. 1974), 643–644.
7. Frank, G. A., Franke, D. L., and Ingogly, W. F. An architecture design and assessment system. *VLSI Design* (Aug. 1985).
8. Gevarter, W. B. Expert systems: Limited but powerful. *IEEE Spectrum* (Aug. 1983).
9. Golazewski, C. H., and Ramadge, P. J. Mutual exclusion problems for discrete event systems with shared events. In *Proceedings of the Conference on Decision and Control* (Houston, Tex., Dec.). IEEE, New York, 1988.
10. Lin, F., and Wonham, W. M. Decentralized supervisory control of discrete event systems. Systems Control Group Report 8612. Univ. Toronto, Toronto, Ont., Canada, July 1986.
11. Maimon, O. Z., and Tadmor, G. Efficient supervisors in discrete event systems. In *Proceedings of 1986 International Conference of Systems, Man, and Cybernetics*. 1986.
12. Merchant, M. E. Production: A dynamic challenge. *IEEE Spectrum* (May 1983).
13. Ostroff, J. S., and Wonham, W. M. A temporal logic approach to real time control. In *Proceedings of Conference on Decision and Control* (Ft. Lauderdale, Fla., Dec.). IEEE, New York, 1985.
14. Özveren, C. M. Analysis and control of discrete event dynamic systems: A state space approach. Laboratory for Information and Decision Systems Report, LIDS-TH-1907. PhD dissertation. MIT, Cambridge, Mass. Aug. 1989.
15. Özveren, C. M., and Willsky, A. S. Aggregation and multi-level control in discrete event dynamic systems. *Automatica*, submitted for publication.
16. Özveren, C. M., and Willsky, A. S. Tracking and restrictability in discrete event dynamic systems. *SIAM J. Cont. Optimization*, submitted for publication.
17. Özveren, C. M., and Willsky, A. S. Observability of discrete event dynamic systems. *IEEE Trans. Automatic Cont.*, (May 1990).
18. Ramadge, P. J. and Wonham, W. M. Modular feedback logic for discrete event systems. *SIAM J. Cont. Optimization, 25*, 5 (Sept. 1987), 1202–1217.
19. Ramadge, P. J., and Wonham, W. M. Supervisory control of a class of discrete event processes. *SIAM J. Cont. Optimization 25*, 1 (Jan. 1987), 206–207.

20. Sifakis, J. Deadlocks and livelocks in transition systems. Tech. Rep. 185., Jan. 1980.
21. Tadmor, G., and Maimon, O. Z. Control of large discrete event systems: Constructive algorithms. LIDS Publication LIDS-P-1627. MIT, Cambridge, Mass., Dec. 1986
22. Thomas, W. Automata on infinite objects. In *Lehrstuhl für Informatik II*. RWTH Aachen, D-5100 Aachen, Apr. 1988.
23. Tobias, L., and Scoggins, J. L. Time-based air-traffic management using expert systems. *IEEE Control Syst. Mag.* (Apr. 1987).
24. Vaz, A. F., and Wonham, W. M. On supervisor reduction in discrete event systems. *Int J Cont. 44*, 2 (1986), 475–491.
25. Wonham, W. M. *Linear multivariable control: A geometric approach.* Springer-Verlag, New York, 1985.
26. Wonham, W. M., and Ramadge, P. J. On the supremal controllable sublanguage of a given language. *SIAM J. Cont. Optimization 25*, 3 (May 1987), 637–659.

RECEIVED APRIL 1989; REVISED JANUARY 1990; ACCEPTED MARCH 1990