Discrete Event Dynamic Systems: An Overview

MS-CIS-MS-CIS-91-39
GRASP LAB 264

Tarek M. Sobh

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

May 1991

Discrete Event Dynamic Systems : An Overview

Author : Tarek M. Sobh

GRASP Laboratory

Department of Computer and Information Science

School of Engineering and Applied Science

University of Pennsylvania

Special Area Examination Report

Examination Committee :

Dr. Ruzena Bajcsy

Dr. Keith Ross

Dr. Xiaoping Yun

Contents

Discrete Event Dynamic Systems : An Overview

Abstract

In this report we present an overview for the development of a theory for discrete event dynamic systems (DEDS). Dynamic systems are usually modeled by finite state automata with partially observable events together with a mechanism for enabling and disabling a subset of state transitions. DEDS are attracting considerable interests, current applications are found in manufacturing systems, communications and air traffic systems, future applications will include robotics, computer vision and AI. We will discuss notions of modeling, stability issues, observability, feedback and invertibility. We will also discuss the perturbation analysis technique (PA) for analyzing and describing the behaviour of DEDS.

1 Introduction

In this report, we describe a recently developed framework for analyzing and controlling discrete event dynamic systems (DEDS) [5]. The approach used in this framework is a state space approach that focuses on the qualitative aspects of DEDS. We consider the issues of stability, observability, stabilizability by output feedback and invertibility within this framework. We also touch upon some ideas concerned with another, more quantitative-oriented, technique called perturbation analysis [6,7].

1.1 What is a discrete event dynamic system ?

Discrete event dynamic systems (DEDS) are dynamic systems (typically asynchronous) in which state transitions are triggered by the occurrence of discrete events in the system. Many existing dynamic system have a DEDS structure, manufacturing systems and communication systems are just two of them. The state space approach in representing and analyzing such systems will probably lead to more applications that might be incorporated into the framework of DEDS. It will be assumed in the development of the state space approach of analyzing DEDS that some of the events in the system are *controllable*, i.e, can be enabled or disabled. The goal of controlling DEDS is to "guide" the behaviour of the system in a way that we consider "desirable". It is further assumed that we are able to observe only a subset of the event, i.e, we can only *see* some of the events that are occurring in the system and not all. In some cases we will be forced to make decisions regarding the state of the system and how to control a DEDS based upon our observations only.

1.2 Organization of the Report

In the next section we will discuss the finite state model of a DEDS. This representation of a DEDS will be used in sections 3, 4, 5 and 6. This model will be a simple non-deterministic finite-space automaton. Graphical representations for DEDS automatons will be used as examples to explain the definitions and ideas presented in the next four sections. In section 3, the notions of stability for a DEDS will be introduced and discussed. In section 4, we focus on the questions of observability and state reconstruction from intermittent observations of the event trajectory. Section 5 combines the ideas of sections 3 and 4 to address the problem

of stabilization by output feedback and section 6 address the problem of reconstructing the event trajectory from observations. Section 7 touches upon the perturbation analysis technique for evaluating the performance of DEDS.

2 Modeling

The discrete event dynamic systems under consideration can always be modeled by a non-deterministic finite-state automata with partially observable and controllable events. In particular, one can make the distinction between classical automata theory [1,2,3,4] and our representation of DEDS in terms of the state transitions. In classical automata the events are inputs to the system, whereas in DEDS the events are assumed to be generated internally by the system and the inputs to the system are the control signals that can enable or disable *some* of these events. We can represent our DEDS as the following quadruple :

$$G = (X, \Sigma, U, \Gamma)$$

where X is the finite set of states, Σ is the finite set of possible events, U is the set of admissible control inputs consisting of a specified collection of subsets of Σ, corresponding to the choices of sets of controllable events that can be enabled and $\Gamma \subseteq \Sigma$ is the set of observable events. Some functions can also be defined on our DEDS as follows :

- $d : X \rightarrow 2^{\Sigma}$
- $e : X \rightarrow 2^{\Sigma}$
- $f : X \times \Sigma \rightarrow 2^{X}$

where d is a set-valued function that specifies the set of possible events defined at each state, e is a set-valued function that specifies the set of events that cannot be disabled at each state, and f is the set-valued function that specifies state transitions from a state under different events. An output process can be formalized simply : whenever an event in Γ happens we see it, otherwise we don't see anything.

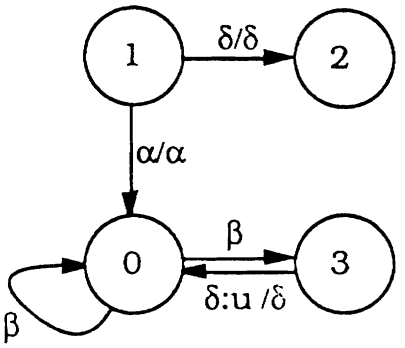


Figure 2.1: A Simple Example

We can visualize the concept of DEDS by an example as in Figure 2.1, the graphical representation is quite similar to a classical finite automaton. Here, circles denote states, and events are represented by arcs. The first symbol in each arc label denotes the event, while the symbol following "/" denotes the corresponding output (if the event is observable). Finally, we mark the controllable events by ":u". Thus, in this example, $X = \{0, 1, 2, 3\}$, $\Sigma = \{\alpha, \beta, \delta\}$, $\Gamma = \{\alpha, \delta\}$, and δ is controllable at state 3 but not at state 1. Also $d(1) = e(1) = \{\alpha, \delta\}$, $d(3) = \{\delta\}$, $e(3) = \phi$, $f(0, \beta) = \{0, 3\}$ etc. A transition, $x \rightarrow^{\sigma} y$, consists of a

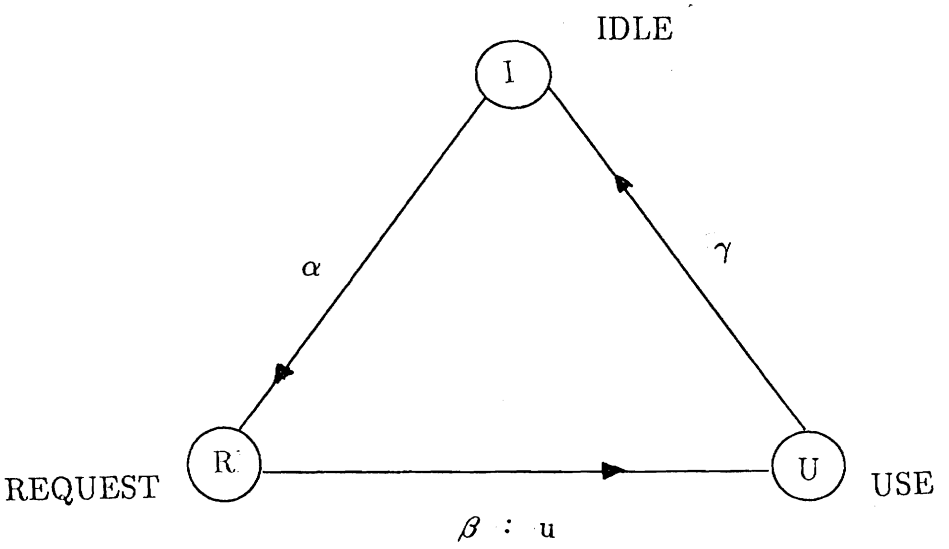
source state, $x \in X$, an event, $\sigma \in d(x)$, and a destination state, $y \in f(x, \sigma)$.

In general, a DEDS automaton A is a nondeterministic finite state automaton, however, if $f(x, \sigma)$ is single valued for each $x \in X$ then A can be termed as a deterministic finite state automaton. A finite string of states, $\mathbf{x} = x_0 x_1 ... x_j$ is termed a path or a state trajectory from x_0 if $x_{i+1} \in f(x_i, d(x_i))$ for *all* $i = 0...j-1$. Similarly, a finite string of events $s = \sigma_1 \sigma_2 ... \sigma_j$ is termed an event trajectory from $x \in X$ if $\sigma_1 \in d(x)$ and $\sigma_{i+1} \in d(f(x, \sigma_1 \sigma_2 ... \sigma i))$ for *all* i, where we extend f to Σ^* via

$$f(x, \sigma_1 \sigma_2 ... \sigma_i) = f(f(x, \sigma_1 \sigma_2 ... \sigma_{i-1}), \sigma_i)$$

with $f(x, \epsilon) = x$. In our graphical example (Figure 2.1), $\alpha\beta\beta\delta$ is an event trajectory.

Another realistic and simple example for a DEDS can be modeled as a resource user. Where the Automaton will be a deterministic one in this case, with three states I (IDLE), R (REQUEST) and U (USE), and with transitions as shown.



Here we take $X = \{I, R, U\}$, $\Sigma = \{\alpha, \beta, \gamma\}$, $\Gamma = \{\alpha, \beta, \gamma\}$. The (two) control patterns corresponds to enabling and disabling event β at state R. A transition $R \rightarrow U$ may occur only when β is enabled. More interesting examples for using resources arise with the concurrent control of several of the above resource user example.

2.1 Generated Languages

A collection of strings $L \subset \Sigma^*$ is termed a language over the alphabet Σ [9]. For example, for any $x \in X$, $L(A, x)$ is a language over Σ which we refer to as the language generated by x in A. In our first example (Figure 2.1), $L(A, 0)$ can be expressed as $(\beta + \beta\delta)^*$, where "+" denotes the union of β and $\beta\delta$.

A language is termed a regular language if it can be expressed by using concatenations, unions and *. Since we use a nondeterministic finite automaton to represent a DEDS, and we know from classical automata theory that any nondeterministic finite automaton can be converted to a deterministic finite one, it will always be the case that the languages generated by a state in A are regular, as deterministic finite automata always produce regular languages as opposed to more powerful models such as pushdown automata, grammars and turing machines. It will never be the case that a state will generate a palindrome language or a language like $\{\alpha^i\beta^i | \alpha, \beta \in \Sigma, i \in N\}$, where N is the set of natural numbers. A recognizer can always be constructed for such a regular language, it is also a fact that there exists a recognizer with the least possible number of states. Such a recognizer is termed minimal.

2.2 Ranges and Liveness

If we denote a transition labeled by σ by $\rightarrow^{\sigma}$, then we can similarly let $\rightarrow^{s}$ denote a string of transitions s and $\rightarrow^{*}$ denote any number of transitions, including no transitions. We can define the range of a state x by

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

indicating the set of states that can reached from x, we can also define the range of a subset of states Q in X by

$$R(A,Q) = \bigcup_{x \in Q} R(A,x)$$

An algorithm for computing $R(A,X_0)$ for any $X_0 \subset X$ that runs in $O(n)$ where $n = |X|$ can be easily formalized as follows :

Let $R_0 = Q_0 = X_0$ and iterate

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

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

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

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

3 Stability

In this section we discuss the notions of stability and the possibility of stabilizing a discrete event dynamic system. In particular, we are going to concentrate on stability notions with respect to the *states* of a DEDS automaton. Assuming that we have identified the set of

"good" states, E, that we would like our DEDS to "stay within" or do not stay outside for an infinite time, the problem would reduce to :

- Checking out whether all trajectories from the other states will visit E infinitely often.
- Trying to "guide" the system using the controllable events in a way such that the system will visit the "good" states infinitely often.

We shall start by defining and testing for different notions of stability and then discuss ways to stabilize a system. We shall start by assuming that the DEDS model under consideration is an uncontrolled system with perfect knowledge of the state and event trajectories ($\Sigma \cap \overline{\Gamma} = \phi$), to simplify developing the definitions and examples.

3.1 Pre-Stability

To capture the idea of stability , we can suppose that we have already identified a subset of states E in X that returning to E implies being in a position to continue desired behaviour from that point on. We can define the notion of a state in the DEDS being stable with respect to E in two stages. The first stage will be the weaker notion and will be termed pre-stability. We say that $x \in X$ is pre-stable if *all* paths from x can go to E in a finite number of transitions, i.e, no path from x ends up in a cycle that does not go through E.

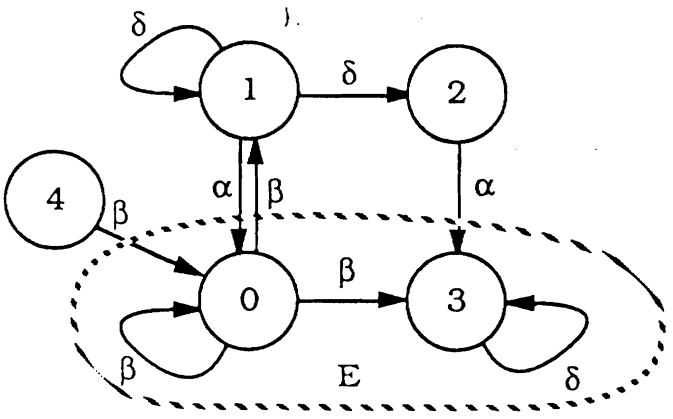


Figure 3.1: Stability Example

In Figure 3.1, states 0, 2, 3, and 4 are pre-stable, since all transitions from them can goto $\{0, 3\}$ in a finite number of transitions. State 1 is not pre-stable since it will stay forever outside E if an infinitely long string of δ's occurs.

A definition of pre-stability can be formalized as follows :

Given a live system A and some $E \subset X$, a state $x \in X$ is pre-stable with respect to E (or E-pre-stable) if for all $\mathrm{x} \in \mathcal{X}(A, x)$ such that $|\mathrm{x}| \geq n$, there exists $y \in \mathrm{x}$ such that $y \in E$. We say that a set of states is E-pre-stable if all its elements are E-pre-stable and a system A is pre-stable if X is E-pre-stable.

The restriction for liveness can be flexible in the sense that if all the dead states are within E, then an automaton might still be E-pre-stable. It follows from the above definition that a state $x \in X$ is E-pre-stable iff $x \in E$ or $f(x, d(x))$ is E-pre-stable. The following algorithm computes the maximal E-pre-stable set X_p within a system :

Let $X_0 = E$ and iterate :

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

Terminate when $X_{k+1} = X_k$, then $X_p = X_k$.

In Figure 3.1, it can be noticed that $X_1 = X_2 = X_p = \{0, 2, 3, 4\}$.

3.2 Stability

The stronger notion of stability corresponds to returning to the set of "good" states E in a finite number of transitions following any excursion outside of E. Thus, given E, we define a state $x \in X$ to be E-stable if all paths go through E in a finite number of transitions and then visit E infinitely often.

As an example, in Figure 3.1, where $E = \{0, 3\}$, only 2 and 3 are stable states. State 1 is not stable since the system can loop at 1 infinitely. State 0 although in E is not stable since the system can make a transition to 1 and then stays there forever, the same applies to state 4.

We can use the previously defined notion of pre-stability and define a state to be E-stable if all the states in its reach are E-pre-stable. In Figure 3.1, 0 and 4 are not E-stable since they can reach 1, which is not E-pre-stable. We can define stability as follows :

Given a live A and $x \in X$, x is E-stable iff $R(A, x)$ is E-pre-stable. A $Q \subset X$ is stable if all $x \in Q$ are stable. A system A is stable if X is a stable set, from which we can conjecture that A is E-stable iff it is also E-pre-stable.

3.3 f-Invariance

A much stronger notion of stability can be described as "staying" within a given set of states. We thus define f-invariance for a subset Q in X as follows :

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 follows that any trajectory that starts in an f-invariant set stays in that set *forever*, it also follows that a set Q is f-invariant iff $R(A,Q) \subset Q$.

3.4 Pre-Stabilizability

In this section we introduce control and reconsider the stability notions discussed before. We try to "guide" our system or some states of it to behave in a way that we consider desirable. Pre-stabilizability is described as finding a state feedback such that the closed loop system is pre-stable. We can then define pre-stabilizability formally as follows :

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

As an example, in Figure 3.2, state 1 is pre-stabilizable since disabling γ pre-stabilizes 1. However, disabling γ at state 2 leaves no other defined events at 2 and "kills" it, so neither state 2 or 3 is pre-stabilizable.

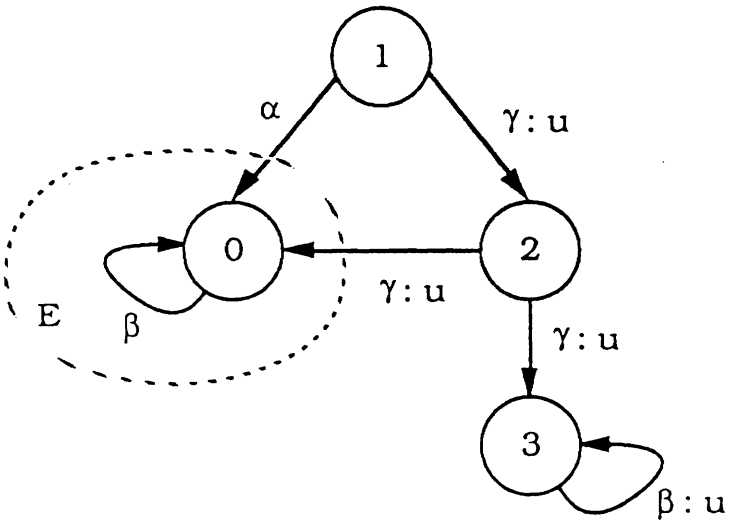


Figure 3.2: Example for the Notion of Pre-Stabilizability

3.5 Stabilizability and (f,u)-Invariance

Stabilizability is an extension of pre-stabilizability. Stabilizability is described as finding a state feedback such that the closed loop system is stable. We can then define stabilizability formally as follows :

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

In Figure 3.3, disabling β at state 2 is sufficient to make the whole system stable with respect to state 0. Disabling γ at state 1 will help stabilize only state 1, because the system can then continue looping between states 2 and 3. Disabling β at state 3 will not help stabilize or pre-stabilize any state.

Using control patterns to "drive" a subset of a system to be f-invariant is still another notion of stabilizability. 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. Another notion of (f,u)-invariance is sustainable (f,u)-invariance. 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.

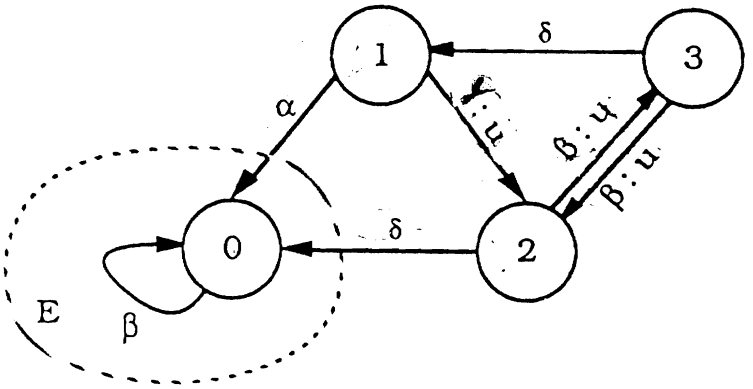


Figure 3.3

For example, in Figure 3.4, disabling event α at state 1 will make the subset {1, 2} sustainably (f,u)-invariant. Also, disabling event δ at state 1 will make the subset {0, 1} sustainably (f,u)-invariant.

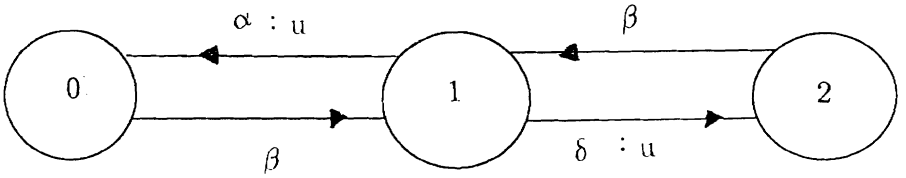


Figure 3.4

4 Observability

In this section we address the problem of determining the current state of the system. In particular, we are interested in observing a certain sequence of *observable* events and making a decision regarding the state that the DEDS automaton A might possible be in. In our definition of observability, we visualize an intermittent observation model, no direct measurements of the state are made, the events we observe are only those that are in $\Gamma \subset \Sigma$, we will not observe events in $\Sigma \cap \overline{\Gamma}$ and will not even know that any of which has occurred. State ambiguities are allowed to develop (which must happen if $\Sigma \neq \Gamma$) but they are required to be resolvable after a *bounded* interval of events. This notion of observability can be illustrated graphically as in Figure 4.1.

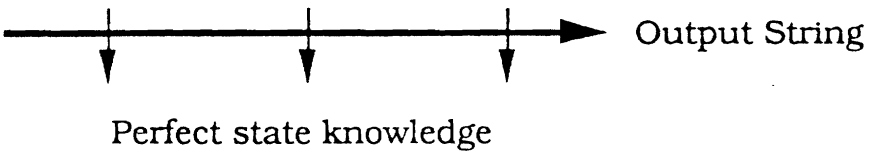


Figure 4.1: Notion of Observability: The state is known perfectly only at the indicated instants. Ambiguity may develop between these but is resolved in a bounded number of steps.

4.1 Requirements

In developing the theory and examples we shall concentrate on uncontrolled models of DEDS automatons with partial knowledge of the event trajectory. Due to the fact that we are "seeing" only observable events in Γ in our system, it is not desirable to have our automaton

generate arbitrarily long sequences of unobservable events in $\Sigma \cap \overline{\Gamma}$. A necessary condition to guarantee this is that the automaton after removing the observable events $A|\overline{\Gamma}$, must not be alive. In fact, it is also essential that every trajectory in $A|\overline{\Gamma}$ is killed in finite time by being forced into a dead state. It can be seen that the condition for a DEDS automaton to be unable to generate arbitrarily long sequences of unobservable events, is that $A|\overline{\Gamma}$ must be D-stable, where D is the set of states that only have observable events defined (i.e, $D = \{x \in X | d(x) \cap \overline{\Gamma}\}$).

4.2 State Observability

As illustrated in Figure 4.1, a DEDS is termed *observable* if we can use the observation sequence to determine the current state exactly at intermittent points in time separated by a *bounded* number of events. More formally, taking any sufficiently long string, s, that can be generated from any initial state x. For any observable system, we can then find a prefix p of s such that p takes x to a *unique* state y and the length of the remaining suffix is bounded by some integer n_o. Also, for any other string t, from some initial state x', such that t has the same output string as p, we require that t takes x' to the same, unique state y.

In Figures 4.2 and 4.3 a simple system and its observer are illustrated. It can be seen that the observer will never know when will the system be in states 3, 4 or 5, since the events that takes the system to those states are unobservable (δ/ϵ means that $\delta \in \Sigma \cap \overline{\Gamma}$), namely δ and γ. There are two states in the observer which are ambiguous, however, another two states are singleton states, i.e, when our observer reaches them, we'll know the exact state that the DEDS in currently in.

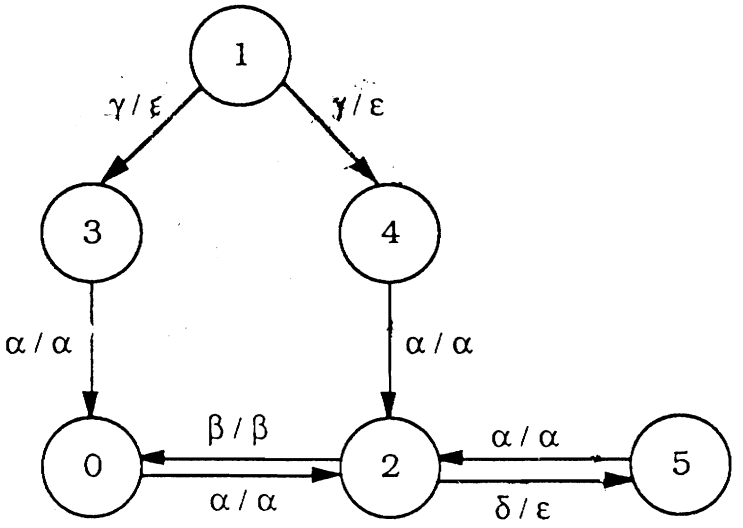


Figure 4.2: A Simple Example

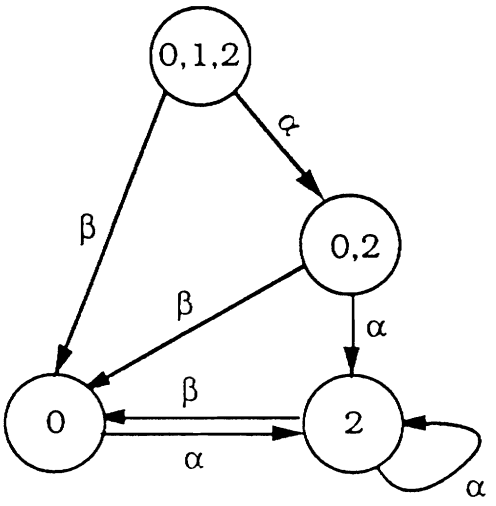


Figure 4.3: Observer for the system in Figure 4.2

Had it been the case that our observer could, for example, loop forever in ambiguous states, then the DEDS would be unobservable. This leads to the following formal definition of observability that ties it with the notion of stability :

A DEDS automaton A is observable iff E is nonempty and O is E-stable.

where O is the observer for A and E is the set of singleton states of O. It can be seen that the observer in Figure 4.3 is stable with respect to the nonempty subset of states {0, 2} and thus the DEDS of Figure 4.2 is observable.

4.3 Indistinguishability

We term pair of states (x, y) indistinguishable if they share an infinite length output (observable) event sequence. If we define :

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

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

$$Y = Y_0 \cup Y_1$$

Then Y is the set of states x such that either there exists an observable transition defined from some state y to x, or x has no transition defined to it. As discussed above, the observer only uses the states in Y, and thus we can formally define indistinguishability for states in Y as follows :

Given $x \in X$, let $L_\infty(A, x)$ denote the set of infinite length event trajectories generated from x, and $h(L_\infty(A, x))$ the corresponding set of output (observable) trajectories. The pair $(x, y) \in Y \times Y$ is an indistinguishable pair if $h(L_\infty(A, x)) \cap h(L_\infty(A, y)) \neq \phi$.

It can be noticed that in Figure 4.2, (0,2) is an indistinguishable pair since an infinite string of α's is one of the possible observable output event sequences from either states. However, this system was shown to be observable, thus the non-existence of indistinguishabilities is not required for observability. If there are indistinguishable states, we will not always be able to determine which of these we were in at some point in the past, but this does not rule out the possibility that we may occasionally know the current state.

4.4 WD Observability

A system is termed WD observable if it is observable with a delay. It is required that there is perfect knowledge of the state some finite number of transitions into the past at intermittent points in time. Figure 4.4 illustrates the concept of WD observability.

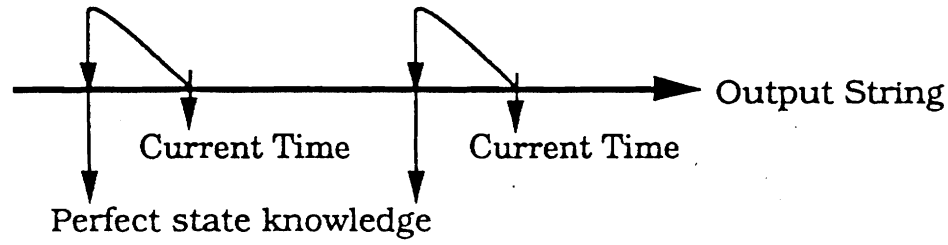


Figure 4.4: Observability with a Delay: The state, a finite number of transitions into the past, is known perfectly at intermittent (but not necessarily fixed) points in time.

As an example of a WD observable DEDS, Figure 4.5 represent such an automaton and its observer. All events in this example are assumed to be observable. The system is not observable since the observer does not have any singleton states (E is empty). When α or

β occurs, we do not have perfect knowledge of the current state, but when either α or β happens we know perfectly what was the *previous* state.

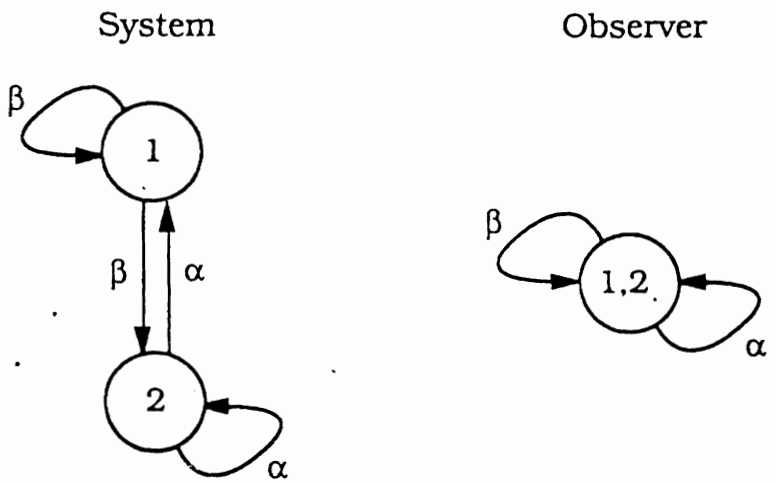


Figure 4. 5: Example for WD Observability

5 Output Feedback Stabilizability

In this section we combine the ideas discussed in the previous two sections regarding observability and stability to address the problem of stabilization by dynamic output feedback under partial observations. In this section we concentrate on partially controlled systems with partial knowledge of the event trajectory. In particular, our goal is to develop stabilizing compensators by cascading and a stabilizing state feedback defined on the observer's state space.

5.1 Requirements

To attack the problem of output feedback stabilization, it should be noticed that we are actually trying to "manipulate" the system's observer, in other words, what we have available in a sequence of observable events (the system's *output*) and we are trying to use this output to control the behaviour of the system using *only* the events that we can control. It is then possible to redefine the problem of output feedback stabilization as the stabilization of the observer by state feedback.

The obvious notion of output E-stabilizability (stabilizability with respect to $E \subset X$) is the existence of a compensator C so that the closed-loop system A_C is E-stable. It is possible that such a stabilizing compensator exists, such that we are sure that the system passes through the subset E infinitely often (E-stable) *but* we never know when the system is in E. A stronger notion of output feedback stabilizability would not only requires that the system passes through subset E infinitely often, but also that we regularly know when the system is in E. In out example and discussion we shall concentrate on this stronger notion of output stabilizability.

5.2 Strong Output Stabilizability

The basic idea behind strong output stabilizability is that we will know that the system is in state E iff the observer state is a subset of E. The fact that the observer state should be a subset of E instead of having the observer state of interest *includes* states in E is because we want to *guarantee* that our system in within E. Our compensator should then *force* the observer to a state corresponding to a subset of E at intervals of at most a finite integer i

observable transitions. We can then formalize the notion of a strongly output stabilizable system as follows :

A is strongly output E-stabilizable if there exists a state feedback K for the observer O such that O_K is stable with respect to $E_O = \{ \hat{x} \in Z \mid \hat{x} \subset E \}$.

where Z is the set of states of the observer.

As an example, considering the DEDS and its observer in figure 5.1, where $E = \{1, 2\}$, we have to check the observer stability (or stabilize the observer) with respect to E_O, because this is the only observer state that is a subset of E. As a start, we do not know which state is our system in (as denoted by the state $\{0, 1, 2, 3\}$), however, using the observer transitions we can see that to achieve E_O-stability for the observer we only need to disable α at the observer state $\{0, 2\}$. It should be noted that all the events are observable in this DEDS automaton.

6 Invertibility

In this section we will discuss the notion of invertibility. The problem of invertibility arises due to the fact that a DEDS is, in general, a *partially* observable system. That is, "seeing" some events while observing a system does not imply that those events were the only ones that actually happened. The problem of reconstructing the *full* event sequence given only output (observable) events is what we term the invertibility problem.

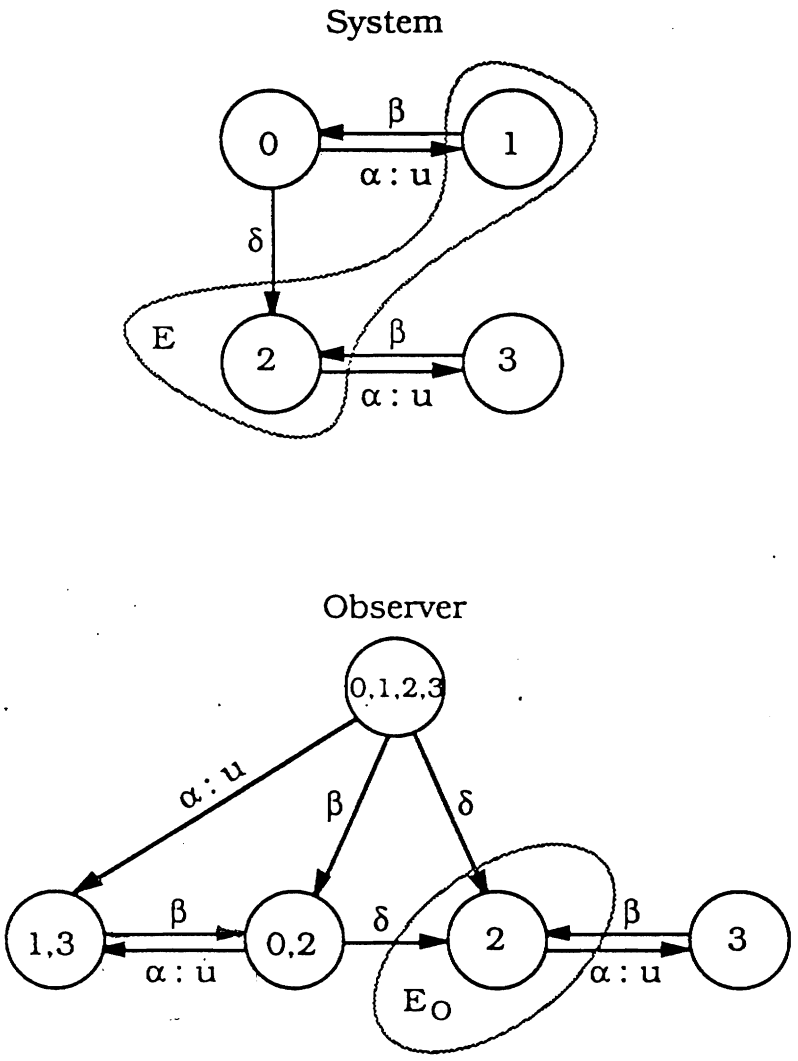


Figure 5.1: Example for Strong Output Stabilizability (all the events are observable)

6.1 Requirements

In order to be able to tackle this problem we need to use the automata model of a DEDS, so it will be assumed that the model of the DEDS behaviour is known a-priori. The invertibility problem arises due to the fact that $\Gamma \subset \Sigma$, had $\Gamma = \Sigma$, invertibility would have been a trivial problem. The model we shall use in this section will be the standard model of a DEDS discussed in section 2, with partial knowledge of the event trajectory, however, the assumption that the system in uncontrollable will be made to simplify developing the theory. There are two notions of invertibility : The first notion assumes that the initial state in the DEDS automaton in known, the second notion does not assume that. It should be quite clear that the second notion will be harder to analyze, because it involves estimating the current state first. In our treatment of the problem we will discuss the first notion.

6.2 WD-Invertibility

By WD-invertibility we mean invertibility with a delay. We consider the DEDS automaton A that is the minimal automaton generating the event language $L = L(A, x_0)$, so that all the states can be reached from x_0, and no two states generate the same language. We also assume that A is deterministic. It should be "safe" enough to make those assumptions, because we will be concerned with the estimation of elements in L, we also can always choose a minimal deterministic automaton with an initial state that generates L, due to that fact that L will always be a regular language.

In particular, we are concerned with the problem that given L (or A and x_0), whether we can reconstruct an event trajectory $s \in L$ when we only observe the part of s in Γ.

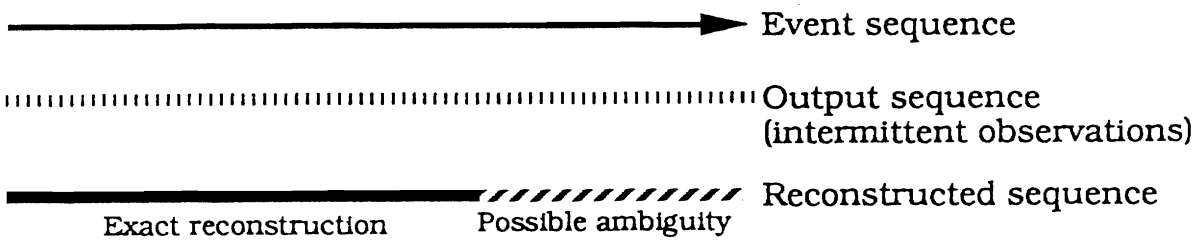


Figure 6.1: Invertibility with a Delay: Given the output sequence, the event sequence is reconstructed exactly but with some delay. The ambiguity at the end of the reconstructed string will be resolved using future observations.

We define a WD-invertible language, as one in which we can, at any time, use knowledge of the output (observable) sequence up to that time to reconstruct the full event sequence up to a point *at most* an integer number of events n_d into the past. Figure 6.1 shows a graphical explanation of the notion of WD-invertibility.

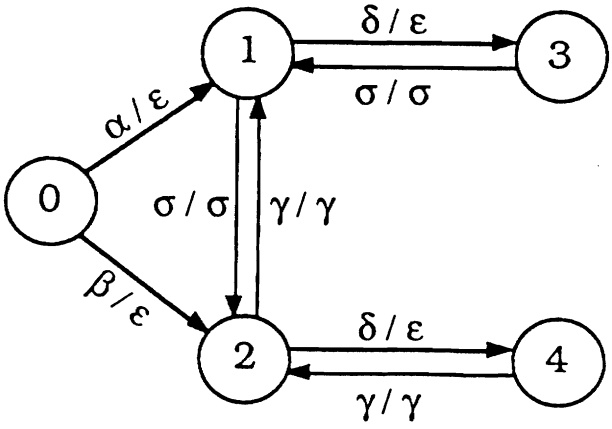


Figure 6.2: Example for WD-Invertibility: State 0 is the initial state.

WD-invertibility can be illustrated by an example as in Figure 6.2. In this system, state 0 in the initial state. The notation α/ϵ means that event α is not observable. In this case, L is WD-invertible with $n_d = 4$. It is not invertible *at all* without delay (i.e, with $n_d = 0$). As an example, if we observe σ^2, the original input sequence could be $\alpha(\delta\sigma)^2$ or $\alpha(\delta\sigma)^2\delta$ or $\alpha\delta\sigma\sigma$, etc., but the first three events are known with certainty.

6.3 Ambiguity and Non-Invertible DEDS

To discuss the notions of ambiguity and non-invertibility we need to define a few notations on languages. In particular :

- $L_f(A, x)$: All the strings in $L_(A, x)$ with observable events as their last events.
- $L_1(A, x)$: Those strings in $L_f(A, x)$ that have only one observable event.
- $L_\sigma(A, x)$: The set of strings in $L_1(A, x)$ that have $\sigma \in \Gamma$ as the observable event.

A DEDS automaton A is termed ambiguous if for some $x \in X$ and $\gamma \in \Gamma$, there exists distinct strings $s, t \in L_\gamma(A, x)$ such that $f(x, s) = f(x, t)$. Moreover, if A ambiguous, then L is not WD-invertible. In other words, if there exists two different sequences of events taking a state to another, and with the same observable event for both sequences, then the language is not WD-invertible. This is because no future behaviour will enable us to distinguish between those strings.

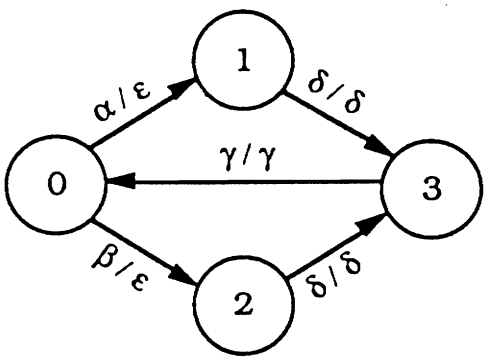


Figure 6.3: Example for an Ambiguous System

In the above example (Figure 6.3), the system is ambiguous as both $\alpha\delta$ and $\beta\delta$, which produce the same output (observable events), take state 0 to state 3. Thus the language generated from state 0 is not invertible.

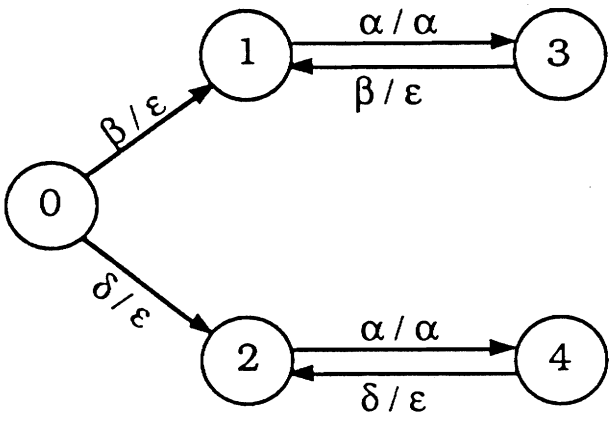


Figure 6.4: Example for an Unambiguous but not Invertible System: State 0 is the initial state.

A DEDS automaton A might be non-invertible although it is unambiguous, that is, unambiguity alone is not sufficient for invertibility. For example, the automaton in Figure 6.4, where 0 is the initial state, is not ambiguous, but L is not invertible, since the event trajectories $(\beta\alpha)^*$ and $(\delta\alpha)^*$ both have the same output α^*. Following from the fact that $R(A, x_0) = X$, one can say that L is WD-invertible iff $L(A, x)$ is WD-invertible for each $x \in X$.

7 Perturbation Analysis

In this section we are going to examine another technique for studying the behaviour of discrete event dynamic systems, namely, perturbation analysis (PA). The PA approach to analyzing DEDS is different from the analysis techniques that we discussed in the previous sections for the automaton model of DEDS, the existence of a consistent and pre-defined automata model of the system under consideration is not necessary to perform PA. For example, if we consider a serial production line with M stations with a queue space of size K_i for each station. Then the total number of states for such a system would be $(\prod_{i=1}^{M}(K_i + 1))(2^M)$, which can amount to billions for relatively small values of K_i and M. It is quite clear that modeling such systems as finite state machines is inefficient, if not impossible. It should also be mentioned that the finite state machines approach is more suitable for answering qualitative rather than quantitative questions.

7.1 What is Perturbation Analysis ?

Perturbation analysis (PA) is a technique that calculates the sensitivity of performance measure of DEDS with respect to system parameters by analyzing its sample path. The object of PA is to obtain the perturbed performance from a nominal experiment or sample path without doing a perturbed experiment. To avoid doing more than one experiment or simulate a perturbed experiment is the goal of PA.

7.2 Infinitesimal Perturbation Analysis (IPA)

Perturbation analysis (PA) should calculate the sensitivity of a particular performance measure T with respect to a system parameter θ, in other words, we are interested in the value

$$dT/d\theta = \lim_{\Delta\theta \to 0} [T(\theta + \Delta\theta) - T(\theta)]/\Delta\theta$$

In order to relate the right hand side to experimental values $\hat{T}$, the above formula can be re-written as

$$dT/d\theta = \lim_{\Delta\theta \to 0} \lim_{N \to \infty} [\hat{T}(\theta + \Delta\theta, N) - \hat{T}(\theta, N)]/\Delta\theta$$

where N is the number of the "client processes" for which the performance measure is being evaluated during the experiment. This can be expressed in terms of the change in the performance measure estimate as

$$dT/d\theta = \lim_{\Delta\theta \to 0} \lim_{N \to \infty} \Delta\hat{T}(\Delta\theta, N)/\Delta\theta$$

The problem at hand now would be how to calculate the change in the performance measure estimate by observing the *unperturbed* experiment. It was shown that under the assumptions

of small perturbation values and in the near-absence of "dramatic" changes in the system's behaviour due to the perturbation (for example, a small influence on the structure of busy periods and a neglected amount of coalescing in the path in a GI/G/1 queue) that an experimental estimate which converges to the true value of $dT/d\theta$ as $N \rightarrow \infty$ can be easily computed while the nominal (unperturbed) experiment is evolving. It should be noted that this gradient estimate is an infinitismal PA estimates, and for "sufficiently small" $\Delta\theta$ the IPA estimate will be equal to the finite difference estimator. However, one should notice that the correct definition of the gradient involves letting $N \rightarrow \infty$ first and *then* $\Delta\theta \rightarrow 0$ for convergence to $dT/d\theta$, for the IPA estimate we see that for small $\Delta\theta$ ($\Delta\theta \rightarrow 0$) the IPA behaves like the finite difference estimator for a *fixed* N after that we allow a large number of experiments to be performed ($N \rightarrow \infty$) and thus the IPA should not, in theory, converge to $dT/d\theta$, since the order of taking limits is reversed. Fortunately, for a class of systems, such change of limits can be mathematically correct.

7.3 Smoothed Perturbation Analysis (SMA)

In some cases, the IPA technique discussed above will fail to work. One instance of this failure will be due to the assumption that small changes in the system parameter θ will not produce coalescing of busy periods in a GI/G/1 queue because of small $\Delta\theta$. If the performance measure is the average number of customers served in a busy period, then clearly our assumption will lead to an IPA estimate of sensitivity equal to zero !

The idea of using conditional probabilities to develop an extension for the IPA was introduced to avoid some of the such failures. A conditioning variable can be introduced to

decompose the gradient estimate expectation expression. The fact that more information is used in developing the conditional probability counts for the "smoother" kind of performance measure estimate curve that is obtainable by using this method.

7.4 Extended Perturbation Analysis (EPA)

For systems that can be represented by markov chains, a new approach that may overcome the potential inconsistency of IPA can be applied. The idea behind the extended perturbation analysis is the fact that the perturbed and unperturbed systems should be statistically evolving similarly once they enter a common state x, due to their markovian property. This method works by choosing a finite $\triangle\theta$ and predicting, from the nominal path, where the perturbed path would have branched to a different state, say y, while the nominal path continues in, say, state x. Up to this point, an IPA-like estimator is used to compute the effects of perturbation, but at this point, the computation is "frozen".the algorithm then waits for the system to enter state y during the nominal path, then EPA restarts. The problem with EPA is the inactivity for sections in the nominal path.

7.5 Other Perturbation Techniques

Another Perturbation technique is finite perturbation analysis (FPA), this technique was introduced to overcome the IPA assumption that events do not change order. However, FPA considers changes in order of events to a pre-specified limit, it considers changes in the order of adjacent events. Originally FPA was heuristic and experimental in nature, however, recent research has been performed to provide more theoretical foundations for it. Other

techniques to make IPA work include changing the system parameter under consideration to transform problems into "easier" versions. Using a different representation for the system sometime helps in performing IPA.

8 Discussion and Future Work

In this report, we have discussed some basic notions related to discrete event dynamic systems. We emphasized upon the automaton model of a DEDS and described some ideas regarding controlling and observing the behaviour of such systems. We also mentioned perturbation analysis as a performance measure analysis method. As a future extension, more powerful models could be used instead of finite automata, for example, Grammars, Pushdown Automata, Turing Machines and/or μ-recursive functions. Applications related to fields other than communication and manufacturing systems could be exploited. Many dynamic tasks can be modeled as DEDS and thus they can be analyzed and controlled efficiently using the ideas discussed in this report.

References

[1] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, 1979.

[2] G. E. Révész, *Introduction to Formal Languages*, McGraw-Hill, 1985.

[3] Zvi Kohavi, *Switching and Finite Automata Theory*, McGraw-Hill, 1979.

[4] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, 1981.

[5] C. M. Özveren, *Analysis and Control of Discrete Event Dynamic Systems : A State Space Approach*, Ph.D. Thesis, Massachusetts Institute of Technology, August 1989.

[6] Y. Ho, "Performance Evaluation and Perturbation Analysis of Discrete Event Dynamic Systems", *IEEE Transactions on Automatic Control*, July 1987.

[7] Rajan Suri, "Perturbation Analysis : The State of the Art and Research Issues Explained via the GI/G/1 Queue", *Proc. of the IEEE*, January 1989.

[8] P. J. Ramadge and W. M. Wonham, "Modular Feedback Logic for Discrete Event Systems", *SIAM Journal of Control and Optimization*, September 1987.

[9] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", *SIAM Journal of Control and Optimization*, January 1987.

[10] Yong Li and W. M. Wonham, "Controllability and Observability in the State-Feedback Control of Discrete-Event Systems", *Proc.* 27^{th} *Conf. on Decision and Control*, 1988.