RECENT DEVELOPMENTS IN DISCRETE EVENT SYSTEMS

K. Inan

Electrical and Electronic Engineering Department
Middle East Technical University
Ankara, Turkey

Abstract This article is a brief exposure of the process approach to a newly emerging area called "discrete event systems" in control theory and summarizes some of the recent developments in this area. Discrete event systems is an area of research that is developing within the interstices of computer, control and communication sciences. The basic direction of research addresses issues in the analysis and design of distributed intelligent systems operating within real time constraints. The article focuses on the concepts of an algebraic process approach and formulates supervisory control problems in this framework.

چکیده دراین مقاله، روش فرآیندی یک رشته جدید درحال توسعه بنام «سیستم های وقایع منفعل» بطور خلاصه معرفی میگردد. درین رابطه، مخصوصاً خلاصه پیشرفت ها و دست آوردهای جدید ارئه میشود. سیستم های وقایع منفعل، یک رشته پژوهشی است که با علاقه دست اندرکاران و پژوهش گران علوم کامپیوتر، مخابرات و کنترل در حال توسعه می باشد. جهت اصلی پژوهش، مسائل و موضوعاتی را که در تحلیل و طراحی سیستم های هوشمند پراکنده تحت عملیّات زمان واقعی مطرح هستند در نظر میگیرد. این مقاله روی مفاهیم روش فرآیندی جبری تأکید نموده و مسائل کنترل ثانویه را در این چهار چوب فرموله می کند.

INTRODUCTION

The concept of a *discrete event* models an instance that causes a qualitative change in a system. For example in a data communication network the instance of requesting to send data or the instance of transmitting a single packet of data may be represented by the events *send-data-request* and *packet-transmitted* respectively. Similarly in a flexible manufacturing system:

part-a-arrived; part-b-arrived; start-process

is a possible sequence of events. A logical model of a discrete event system is a formal description of possible sequence of events that can occur. There are two factors that constrain the event sequences. The first one is the physical possibility of a sequence. For example the event *packet-a-transmitted* must precede the event *packet-a-received* in a possible event sequence. The second factor is logical design constraints. For example in a communication protocol it is usually desired that the receiver receives the data packets in the order demanded by the transmitting side. Therefore the logic of the protocol must impose this constraint so that if the event *packet-n-received* is followed by *packet-m-received* then $m = n + 1$. In more complex instances both the system description and the specification require formal logical models. The problem of *verification* is to check whether the formal description of the system satisfies the formal specification. In realistic applications the complexity of underlying logical models may be intractable due to a large amount of parallelism involved in performing numerous tasks simultaneously. Therefore unlike conventional

dynamical problems, the complexity of logical discrete event systems is not due to the size of the arithmetic operations involved, instead it stems from the requirement of generating and observing events in appropriate sequences.

The origins of logical discrete event systems lie in efforts to build semantic models for parallel computer languages. Among various approaches to this problem two well-known ones are the "Communicating Sequential Processes" (CSP) of Hoare [1] and "A Calculus for Communicating Systems" (CCS) of Milner [2]. The data flow approach of [3] addresses issues in parallel signal or data processing problems. Various logical models have been used for discrete event systems. Finite state machines, Petri-nets [4], temporal logic [5] and process models of Hoare and its derivatives such as finitely recursive marked processes [6,7] are among these. Some of these models have been used as tools for solving the verification problem discussed above. Recently Ramadge and Wonham have introduced a control dimension to this problem in their supervisory control formulation of disctete event systems [8]. Here a controller is introduced to force a system (plant) to obey a given logical specification. Among basic extensions to this approach are supervision under partial observations and decentralized supervision problems [9,10].

The literature on stochastic and timed discrete event systems is much more developed. Here the question addressed is the performance of such systems in terms of efficiency measures such as throughput and delay in various services offered to users. To a large extent these models coincide with the Markov models used in queueing theory and its extensions in various directions. A relatively recent area of interest is deterministic timed models which could serve as a bridge between the logical and stochastic models. The algebraic approach in [11] extends some linear-system-theoretic concepts to a restricted class of timed petri-nets with applications in parallel signal processing and manufacturing systems.

In this paper we focus the discussion on a recent approach in modeling and control of discrete event systems based on the algebra of processes. In particular we develop the relevant concepts of the process approach and formulate supervisory control problems in terms of these concepts.

ALGEBRA OF PROCESSES

Before we proceed with the process definition we first define the finite state machine (FSM) model for a discrete event system. An event driven state machine is given by $S:=(A, X, \delta, x_o, X_f)$ where A is a finite set of events (event alphabet), X is the set of states, $\delta: X \times A \rightarrow X$ is a partial function called the *transition function*, x_o is the initial state and X_f is a subset of X usually referred to as the final set of states. S is called a finite state machine if X is a finite set. If δ is defined on the pair (x, a) then the event transition $< a >$ is said to be allowed at the state x and the next state of the system is given by $\delta(x, a)$. The partial function δ can be extended to event strings s by applying δ consecutively to the events in s starting from the initial state. The resulting partial function is called the transitive closure of δ and is denoted by δ^*. Associated with a state machine S is a set of event strings L such that a string s belongs to L iff $\delta^*(x_o, s)$ is defined. L is called the *language* generated by S. The subset of L denoted by L_f such that $s \in L_f$ iff $\delta^*(x_o, s) \in X_f$ is called the language recognized by S. It is well known that the set of languages generated or recognized by finite state machines is the set of *regular languages* [12].

Now let A^* denote the set of all finite event sequences with events from A. Following Hoare's terminology we call each element (string) in A^* a trace. If s and t are traces then $s \hat{} t$ denotes the trace obtained by concatenating s and t. In order to distinguish the event a from a trace consisting of that single event we use $<a>$ to denote the latter. Let $< >$ denote the empty trace and let $C(A)$ denote the set of all

prefix-closed subsets K (i.e $s \hat{} t \in K \Rightarrow s \in K$) of A^*. A (deterministic) *process* P is defined as a triple $(trP, \alpha P, \tau P)$where trP is an element of $C(A)$ and αP, and τP are functions mapping trP into the sets 2^A and $\{0,1\}$, respectively. The functions αP and τP are called the *event control* and *termination* functions, respectively. Let Π denote the set of all such processes. We define a partial order on Π by letting:

$$P \leq Q \Rightarrow \begin{cases} trP \subseteq trQ \\ \alpha P(s) = \alpha Q(s) \quad \forall s \in trP \\ \tau P(s) = \tau Q(s) \quad \forall s \in trP \end{cases}$$

P is called a *subprocess* of Q iff $P \preceq Q$. A *chain* in Π is an infinite nondecreasing sequence of processes and the limit of the chain is defined as the least upper bound of the sequence. The partial order defined above is complete in the sense that every chain in Π has a limit in Π. In fact if $\{P_k\}$ is a chain then its least upper bound denoted $\cup_k P_k$ is given by

$$tr(\cup_k P_k) = \cup_k tr\, P_k$$

and if $s \in tr\, P_j$ for some then

$$\alpha(\cup_k P_k)(s) = \alpha\, P_j(s)$$
$$\tau(\cup_k P_k)(s) = \tau\, P_j(s)$$

The calculus of processes consists of two basic operations on Π. The *post-process* of a process P denoted by P/s is defined as:

$$tr(P/s) := (t \in A^* \mid s \hat{} t \in tr\, P)$$
$$\alpha(P/s)(t) := \alpha P(s \hat{} t)$$
$$\tau(P/s)(t) := \tau P(s \hat{} t)$$

provided that $s \in tr\, P$. The converse operation is called the choice function. Let $P_1, \ldots, Pn$ be processes in Π, let $a_1, \ldots, a_n$ be distinct events in A and let α_0 and τ_0 be elements in 2^A and $\{0,1\}$, respectively. Then the *choice* operation denoted by

$$Q = (a_1 \rightarrow P_1 | \cdots | a_n \rightarrow P_n)_{(\alpha_o, \tau_o)}$$

is defined as follows:

$$trQ := \{<>\} \cup_{j=1}^{n} (<a_j> \hat{} t_j \mid t_j \in tr\, P_j)$$
$$\alpha Q(<a_j> \hat{} t_j) := \alpha P_j(t_j)$$
$$\tau Q(<a_j> \hat{} t_j) := \tau P_j(t_j)$$

and the initial condition requirement is satisfied, that is,

$$\alpha Q(<>) := \alpha_o$$
$$\tau Q(<>) := \tau_o$$

The fundamental relation of process calculus relates the post-process and choice operations as the converses of each other as given by the following formula

$$P = (a_1 \rightarrow P/<a_1> | \cdots | a_n \rightarrow P/<a_n>)_{(\alpha P(<>), \tau P(<>))} \quad (1)$$

where the set $\{<a_1>, \ldots, <a_n>\}$ denotes the set of all single event traces of P.

The principle advantage of process formulation lies in the possibility of defining funcitions on processes. For example the post-process operation P/s is a partial function defined on those proceses in Π that have s as a trace. On the other hand the choice function maps Π^n into Π for a given initial condition pair (α_o, τ_o). The trace-length-projection operation $P \uparrow n$ is another basic function on Π defined by

$$tr(P \uparrow n) := (s \in tr\, P \setminus \# s \leq n)$$

where $\# s$ denotes the length (no. of events) of the trace s and the event control and termination functions of $P \uparrow n$ are the restrictions of those of P to the traces $P \uparrow n$.

A function $F: \Pi \rightarrow \Pi$ is called *continuous* if for every chain $\{P_j\}$ in Π, $\{F(P_j)\}$ is also a chain in Π and

$$\cup_j F(P_j) = F(\cup_j P_j) \quad (2)$$

F is said to be *strictly causal if*

$$F(P \uparrow n) \uparrow (n+1) = F(P) \uparrow (n+1)$$

and *causal* if

$$F(P \uparrow n) \uparrow n = F(P) \uparrow n$$

These definitions according to Hoare [1] are the discrete event counter-parts of the usual causality definitions for conventional dynamical systems. If F has several arguments then the above definitions are extended by applying them to each argument one at a time. The properties of continuity, causality and strict causality are all preserved under functional compositions. In addition if only a single stage in a composed chain is strictly causal, the others being plain causal, then the composed function is strictly causal.

Next we relate the process definition above to the state machine definition by constructing an abstract state machine $Sp := (A, Xp, \delta, x_o, X_f)$ corresponding to a given process P by letting Xp to be the set of all post-processes of P and defining the transition function as

$$\delta(x,a) := x/\langle a \rangle$$

A post-process $x \in X_f$ *iff* $\tau x\ (\langle\ \rangle) = 1$. It can be shown that the language generated by Sp is equal to trP under this construction.

In order to use processes as discrete event system models one has to construct finite representations for processes. The basic tool in doing this is recursion coupled to algebraic functions. Consider the simple recursive equation

$$Y = (a \rightarrow Y)_{(\alpha_o, \tau_o)}$$

A process Y_{sol} that satisfies this equation is given by:

$$trY_{sol} = \{a\}^* := (a^n \mid n \geq 0)$$
$$\alpha Y_{sol}(s) = \alpha_o$$
$$\tau Y_{sol}(s) = \tau_o$$

In order to arrive at this solution via a recursive recipe first define the process $HALT(\alpha_o, \tau_o)$ as the process with only the empty trace $\langle\ \rangle$ and

$$\alpha HALT_{(\alpha_o,\tau_o)}(\langle\ \rangle) := \alpha_o$$
$$\tau HALT_{(\alpha_o,\tau_o)}(\langle\ \rangle) := \tau_o$$

Then we construct the following fixed point iteration:

$$Y_{j+1} = (a \rightarrow Y_j)_{(\alpha_o, \tau_o)}$$

with $Y_o := HALT(\alpha_o, \tau_o)$. It can be shown that the sequence $\{Y_j\}$ is a chain and it converges to Y_{sol} given above as its limit where this limit is a solution to the recursive equation. Moreover it can be shown that the solution Y_{sol} is unique. The following theorem is a generalization of the simple example given above.

Theorem

Consider the recursive equation

$$X = F(X) \tag{3}$$

where $X \in \Pi^n$ *and* $F: \Pi^n \rightarrow \Pi^n$. *Suppose that* $Z \in \Pi^n$ *satisfies the initial condition requirement:*

$$Z_k = F_k(Z) \uparrow 0 \tag{4}$$

for $k = 1,\ldots,n$ *where the subscript k refers to the kth component of the vectors Z and F. There exists a unique minimal solution* $X = \bar{X}$ *to equation (3) subject to the initial condition constraint* $X_k \uparrow o = Z_k$, $k = 1,\ldots,n$ *provided that each* F_k *is a continuous function. By minimal it is meant that if X' is another solution to the same equation and subject to the same initial condition constraint then for each k,* $\bar{X}_k \preceq X'_k$.

If, in addition, each F_k *is strictly causal then the solution* $\bar{X}$ *is unique*

In order to enrich the algebraic structure of recursively represented processes we define two further operators on them. The first is the parallel operator operating on two processes P and Q and yielding a new process denoted by $P \parallel Q$. The definition proceeds inductively by first letting

$$\alpha(P \parallel Q)(\langle\ \rangle) := \alpha P(\langle\ \rangle) \cup \alpha Q(\langle\ \rangle)$$
$$\tau(P \parallel Q)(\langle\ \rangle) := Min(\tau P(\langle\ \rangle), \tau Q(\langle\ \rangle))$$

and by defining the one-step progress rule for a possible transition $\langle a \rangle$ as follows:

$$(P \parallel Q)\langle a \rangle \begin{cases} := (P/\langle a \rangle \parallel Q/\langle a \rangle), \text{if } \langle a \rangle \in trP \cap trQ \\ := (P/\langle a \rangle \parallel Q), \text{if } \langle a \rangle \in trP \setminus trQ \wedge a \notin \alpha Q(\langle\ \rangle) \\ := (P \parallel Q/\langle a \rangle), \text{if } \langle a \rangle \in trQ \setminus trP \wedge a \notin \alpha P(\langle\ \rangle) \\ \textit{undefined otherwise} \end{cases}$$

By using the transitivity of the post-process operation, that is $(P/s)/t = P/(s \hat{} t)$, the definition above can be completed inductively to cover all the traces of $P \| Q$. According to the definition above each argument process of the parallel operator can progress individually unless they synchronize on a common transition or one process is blocked by the other one. The parallel operator is associative, commutative and causal but not necessarily continuous. As an example consider the processes P, Q and P' where

$$trP' = \{<>, <a>\}$$
$$trP = \{<>, <a>, <b>\}$$
$$trQ = \{<>, <b>\}$$

where

$$\alpha P' \equiv \alpha P \equiv \{a\};\ \alpha Q(<>) \equiv \{b\}$$

and termination functions are all identically 0. Clearly $P' \preceq P$, yet

$$tr(P \| Q) = \{<>, <a>, <b>, ab\}$$

whereas

$$tr(P' \| Q) = \{<>, <a>, <b>, ab, ba\}$$

so that

$$(P' \| Q) \not\preceq (P \| Q)$$

and the monotonocity requirement of continuity is violated. There is an important class of processes on which the parallel operator is continuous. Define a process P to be *proper* if it satisfies

$$(s \hat{} <a> \in trP) \Rightarrow a \in \alpha P(s)$$

and let Π_P denote the set of all proper processes in Π. Then the parallel operator can be shown to be continuous on the set $\Pi_P \times \Pi_P$. Note that the process P in the example above was not proper. Therefore in view of the theorem above if the parallel operator appears on the right side of a recursive definition its arguments must be proper processes for the recursion to have a solution.

Another important operator is the sequential operator. Let P, Q be processes and define the sequential composition $P;Q$ by first letting

$$\alpha(P;Q)(<>) := \begin{cases} \alpha P(<>), \text{if } \tau P(<>) = 0 \\ \alpha Q(<>), \text{otherwise} \end{cases}$$

and

$$\tau(P;Q)(<>) := \begin{cases} 0, \text{if } \tau P(<>) = 0 \\ \tau Q(<>), \text{otherwise} \end{cases}$$

Traces are inductively defined by

$$(P;Q)/(<a>) := \begin{cases} (P/<a>);Q, \text{if } \tau P(<>) = 0 \wedge <a> \in trP \\ Q/<a>, \text{if } \tau P(<>) = 1 \wedge <a> \in trQ \\ \text{undefined otherwise} \end{cases}$$

The sequential operator is an associative, continuous and causal operator. In order to exhibit the expressive power of the sequential operator consider the following recursion:

$$P = (a \rightarrow P;Q \backslash b \rightarrow HALT_{(\{c\},1)})(\{a,b\},0)$$
$$Q = (c \rightarrow HALT_{(\{\},1)})(\{c\},0)$$

It can be shown that the recursion has a unique solution and the solution process P has traces given by:

$$trP = (a^n bc^n \mid n \geq 0)$$

It is well-known that the language corresponding to trP above is a language generated by a context-free grammar [12] and therefore is non-regular and cannot be generated by a finite state machine.

We conclude this section by summarizing the idea behind a finitely recursive process over a given algebra. Consider the following recursive representation

$$X = F(X)$$
$$Y = G(X) \qquad (5)$$

where each F_k is of the following form

$$F_k(X) = (a_{1k} \rightarrow f_{1k}(X) \mid \cdots \mid a_{n_k k} \rightarrow f_{n_k k}(X))_{(\alpha_{ok}, \tau_{ok})}$$

Each function f_{jk} consists of arbitrary number of composition of functions from a given fixed collection. For example if the collection consists only of the sequential and parallel operators defined above then each f_{jk} consists of arbitrary sequence of parallel and sequential operations

operating on any of the argument processes $X_1,...,X_n$. This idea can be formalized by a set whose elements consist of functions that are arbitrary compositions of operators from a given fixed collection C. We call this set of functions an algebra A generated by the collection C. A *finitely recursive process* (FRP) over an algebra A is a solution Y of the representation given by (5) where each f_{jk} as well as G are members of the algebra A. The basic result on FRP's states that if Y is an FRP over an algebra A then for any $s \in trY$ the process $Y/s = H(X)$ where $H \in A$ [6]. Therefore elements of the algebra A form a state set for the FRP process.

In [7] the ideas summarized above are generalized to *marked processes*. Omitting certain technicalities, a marked process is a pair $(trP, \mu P)$ where trP is as defined above and μP is an abstract function mapping trP into a marking set M. For the special case discussed above $\mu = \alpha P \times \tau P$ and $M = 2^A \times \{0, 1\}$. It was shown in [7] that the construct of a finitely recursive process over an algebra can be extended to marked processes. In paricular known discrete event models such as Petri-nets and others are special cases of such a construct where the algebraic structure and recursive nature of representation is implicit in these models. Examples of FRP as an expressive modeling tool is given in [6].

SUPERVISORY CONTROL

In this section we formulate the supervisory control problem introduced by Ramadge and Wonham [8] in terms of the process concepts of the previous section. In supervisory control one tries to force a given process (uncontrolled plant) by means of blocking some of its transitions (supervisory control action) such that the controlled process behaves in accordance with a given specification process. The decision to block a transition depends on the observations made by the supervisor in question. Moreover the supervisor may lack the authority to block some of the event transitions (uncontrollable events) of the plant. In the decentralized supervision more than one supervisor may be in action each with its own observation and control authority. Below we model these concepts and formulate the problem.

In order to formulate the partial observation capacity of the supervisors we first define trace and process projections on a given process. First define the projection of a trace s on a process P inductively as follows:

$$\begin{aligned} &\langle\rangle\downarrow P := \langle\rangle \\ &\langle a\rangle\downarrow P := \begin{cases} \langle a\rangle, \text{if } \langle a\rangle \in trP \\ \langle\rangle, \text{otherwise} \end{cases} \\ &(s^\frown\langle a\rangle)\downarrow P := s\downarrow P^\frown\langle a\rangle\downarrow P/(s\downarrow P) \end{aligned} \tag{6}$$

As an example take trP: = {< >, < a >, ab, <c>, ca} and s = dcab then $s \downarrow p$ = ca.
It follows from this definition that $s \downarrow P \in$ trP for any trace s and if $s \in trP$ then $s \downarrow P = s$.
Using the definition of trace projection we define the *projection* of a process Q on a process P denoted $Q.P$ as follows:

$$\begin{aligned} &tr(Q.P) := (s \in trP \mid \exists t \in trQ, s = t\downarrow P) \\ &\alpha(Q.P)(s) := \alpha P(s), \forall s \in tr(Q.P) \\ &\tau(Q.P)(s) := \tau P(s), \forall s \in tr(Q.P) \end{aligned} \tag{7}$$

We now proceed with the supervisory control formulation. The plant and the supervisor are modelled as processes with appropriate constraints. The mechanism of supervision is the parallel composition of the plant and the supervisor processes.

The plant process P is assumed to be a *proper* process and has an event control function that is constant and is equal to the universal alphabet set A.

For the supervisor we define a set of processes on which we seek a solution to the algebraic equation that defines supervisory control. The generic *supervisory process space* is denoted by $S(A_c, Y_o)$ where the parameters A_c and Y_o are called the *controllable events set* and the *observation process* respectively. Formally A_c is a subset of the set of plant events A and Y_o is any process.

Elements of A_c are called the *controllable events* relative to the supervisor. The event control and_termination functions of Y_o do not play any role in the analysis therefore we assume them to be arbitrary constant functions. The process Y_o represents the fact that the supervisor can only observe the filtered traces $s\downarrow_{Y_o}$. Typically $trY_o = A_o{}^{*2}$ where $A_o \subseteq A$ and its elements are called the *observable events* of the supervisor. The filtering for this special case corresponds to hiding all event transitions that are not in A_o. Under the parallel operator the supervisor process is allowed to make transitions synchronized to the observable event transitions of the plant but it can only block events in its event control set. Therefore, unlike the plant process, the supervisor process is an improper process since it can make transitions of pure observation events without anv authority to block them.

Formally the parametrized supervisory subspace of processes $S = S(A_c, Y_o)$ is defined as follows:

$$S \in S \Longleftrightarrow \begin{matrix} trS \subseteq trY_o \\ \alpha S(s) \subseteq A_c \; \forall s \in trS \end{matrix} \qquad (8)$$

The general supervisory control problem is to find a solution to the equation.

$$P \| S_1 \quad \| S_n = K \qquad (9)$$

where K is called the *specification process* and satisfies the constraint

$$K \preceq P$$

and each S_j belongs to a supervisory space of the type explained above

The neccessary and sufficient conditions for the problem above to have a solution is given in [14]. Here we present a simple example. For a more realistic example the reader is referred to [14]. Consider the plant process P and the specification process K given in the figure above. In this figure we represent processes by finite state machines. The events are given by $A:=\{a, b, c, d, e, f\}$. There are two supervisors in

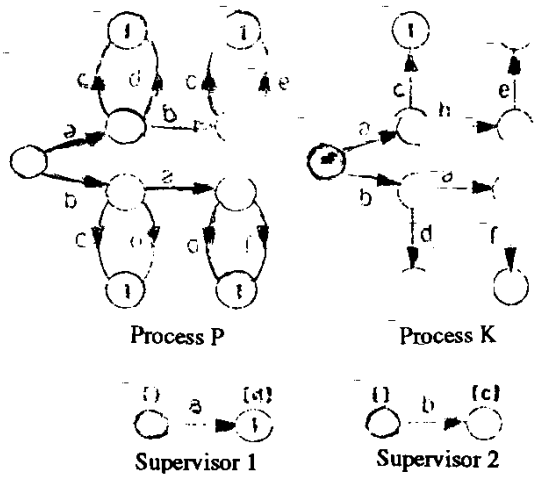

Figure 1

this example where $Y_{oj} := A^*_{oj}$ for j= 1,2 and

$$A_{c1} = A_{c2} := \{c, d\}$$
$$A_{o1} := \{a\}$$
$$A_{o2} := \{b\}$$

The initial state is the darkened one for each process. For the processes P and K the event control sets are constant and equal to A at each state by definition, therefore are not specified in state diagrams. On the other hand the event control sets for the supervisors are marked in the figure for each state. For example, for supervisors 1 and 2 the event control sets at the initial state are both empty and become {d} and {c} after one transition respectively. The termination marks for each state is marked 1 or left empty corresponding to a 0. After observing a transition <a> supervisor 1 blocks all <d> transitions whereas after observing <b> supervisor 2 blocks <c> transitions. The resulting parallel composition of the plant together with the supervisors yields the specification process K.

REFERENCES

1. C.A.R. Hoare "*Communicating Sequential Processes;*" Prentice-Hall International, Herts, England (1985).
2. R. Milner. "*Calculus of Communicating Systems.*" Springer,New York, NY, (1980).
3. J.B. Dennis. In M.Broy, editor, "*Control Flow and Data*

Flow." Springer-Verlag (1985).

4. J.L. Peterson "*Petri Net Theory and the Modeling of Systems.*" Prentice-Hall, Englewood Cliffs, NJ (1981).
5. J.G Thistle and W.M. Wonham. *International Journal on Control,* 44(4), 943. (1986).
6. K. Inan and P. Varaiya. *IEEE Trans. Automatic Control,* 33(7), 626 (1988).
7. K Inan and P. Varaiya. *IEEE Proceedings,* 77(1), 24 (1989).
8. P.R. Ramadge and W.M. Wonham. *SIAM Journal on Control and Optimization,* 25(1), 206. (1987).
9. R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. *IEEE Trans. Automatic Control,* 33(3), 249 (1988).
10. F. Lin and W.M. Wonham. *Information Sciences,* 44,199 (1988).
11. G. Cohen, D. Dubois, J.P. Quadrat, and M. Voit. *IEEE Trans. Automatic Control,* 30 (3), 210. (1985).
12. J.E. Hopcroft and J.D.Ullman. "*Introduction to Automata Theory, Languages and Computation.*" Addison-Wesley (1979).
13. M. Hennessy. "*Algebraic Theory of Processes.*" MIT Press, (1988).
14. K. Inan, "An Algebraic Approach to Supervisory Control." to be published.