Wonham, W.M.: The control of discrete event systems. Proc. IEEE 77(1), 81–98

Article *in* Proceedings of the IEEE · February 1989

DOI: 10.1109/5.21072 · Source: IEEE Xplore

2 authors:

Peter J. Ramadge
Princeton University
140 PUBLICATIONS **16,354** CITATIONS
SEE PROFILE

Walter Wonham
University of Toronto
338 PUBLICATIONS **33,379** CITATIONS
SEE PROFILE

PROC/77/1//25174

The Control of Discrete Event Systems

P. J. G. Ramadge
W. M. Wonham

Reprinted from
PROCEEDINGS OF THE IEEE
Vol. 77, No. 1, January 1989

The Control of Discrete Event Systems

PETER J. G. RAMADGE, MEMBER, IEEE, AND W. MURRAY WONHAM, FELLOW, IEEE

Invited Paper

A Discrete Event System (DES) is a dynamic system that evolves in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events. Such systems arise in a variety of contexts ranging from computer operating systems to the control of complex multimode processes. We survey a control theory for the logical aspects of such DESs. This theory was initiated by Ramadge and Wonham, and has subsequently been extended by the authors and other researchers to encompass control theoretic ideas such as controllability, observability, aggregation, and modular, decentralized, and hierarchical control. We concentrate on the qualitative aspects of control but also consider computation and the related issue of computational complexity.

I. Introduction

A Discrete Event System (DES) is a dynamic system that evolves in accordance with the abrupt occurrence, at possibly unknown irregular intervals, of physical events. For example, an event may correspond to the arrival or departure of a customer in a queue, the completion of a task or the failure of a machine in a manufacturing system, transmission of a packet in a communication system, or the occurrence of a disturbance or change of setpoint in a complex control system. DESs arise in the domains of manufacturing, robotics, vehicular traffic, logistics (conveyance and storage of goods, organization and delivery of services), and computer and communication networks. These applications require control and coordination to ensure the orderly flow of events. As controlled (or potentially controllable) dynamic systems, DESs qualify as a proper subject for control theory, a viewpoint that we shall develop in some detail in the present paper.

In the past, DESs have usually been sufficiently simple that intuitive or ad hoc solutions to various problems have been adequate. The increasing complexity of man-made systems, made possible by the widespread application of computer technology, has taken such systems to a level of complexity where more detailed formal methods become necessary for their analysis and design. Indeed, the use of VLSI has already led to the implementation of modular, hierarchical, and distributed systems on a scale never before possible. Such systems pose control and coordination problems of an unparalleled scope and complexity, and as yet there is little theory to serve as a guide in their resolution.

In this paper we survey automata and formal language models for DESs and the theory initiated by Ramadge and Wonham [56], [76], and subsequently extended by the authors and other researchers. The objective of this theory has been to examine control theoretic ideas such as controllability, observability, aggregation, and decentralized and hierarchical control for DESs from a qualitative viewpoint. The framework has proved useful in the theoretical analysis of a number of basic supervisory control problems [56], [76]; has motivated investigations using related models in database systems [26], [27] and manufacturing systems [38]; and, more recently, has been extended to cover modular [55], [95] and distributed [9], [35], [37] control. The main advantage of the model is that it separates the concept of open loop dynamics (plant) from the feedback control, and thus permits the formulation and solution of a variety of control synthesis problems. To date, the theory does not support all of the features that one would desire of a full theory of DESs. Nevertheless, the model has provided valuable concepts and insights to serve as guidelines for future work, and has contributed to our understanding of the fundamental issues involved in the analysis and control of DESs.

Computational problems for DESs are frequently complex. In our setting this manifests itself in the complexity of the computations involved in solving basic control synthesis problems. Although these have been shown to be of polynomial complexity in the number of states, the number of states in a practical system can be exponential in the number of constituent processes. To some extent this problem can be mitigated through modular synthesis [55], [75], and in certain instances can be overcome by restricting attention to processes with special structure [52], [49]. These issues are addressed in our survey.

Our coverage of recent work on DESs is of a tutorial nature, and as a result, many technical points have been omitted or glossed over. This has been necessary in order

Manuscript received July 7, 1988; revised September 13, 1988. This research was partially supported by the National Science Foundation through Grant ECS-8504584, by an IBM Faculty Development Award, and by the National Science and Engineering Research Council of Canada through Grant A-7399.

P. J. Ramadge is with the Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, USA.

W. M. Wonham is with the Department of Electrical Engineering, University of Toronto, Toronto, Canada M5S1A4.

IEEE Log Number 8825174.

to present the concepts and results which we think are important without excessively burdening the reader with notation and definitions. More detailed expositions of the topics discussed can be found in the references cited in the text of the paper.

The remainder of the paper is organized as follows. In Section II we give a brief introduction to the modeling of discrete event systems. This places the model discussed in this paper in the context of other work in the area. Section III introduces the basic DES model and contains some simple illustrative examples. Our primary interest is in the control of DESs, and this topic is introduced in Section IV. Several abstract but basic control problems are discussed, and an example is presented of their application. In view of the complexity of control problems for DESs it is of interest to investigate structured solutions to the problems of concern. We take up this theme in Section V by introducing the concept of modular synthesis. Section VI, on partial observations and observability, lays the foundation for our discussion of distributed control in Section IX. Section VIII consists of a brief discussion of modeling the infinite behavior of DESs, and how the basic control results of Section IV can be extended. In Section IX we turn to the issue of computational complexity; several of the basic problems discussed in the earlier sections are analyzed. Finally, in Section X we specialize the basic model to consider a structured class of DESs, and the complexity of some simple but interesting control problems.

II. Modeling Discrete Event Systems

In this section we give a brief overview of the modeling of DESs. To fix a context for our discussion, let us define a DES to be a dynamic system with a discrete state space and piecewise constant state trajectories; the time instants at which state transitions occur, as well as the actual transitions, will in general be unpredictable. A typical state trajectory for such a system is shown in Fig. 1.

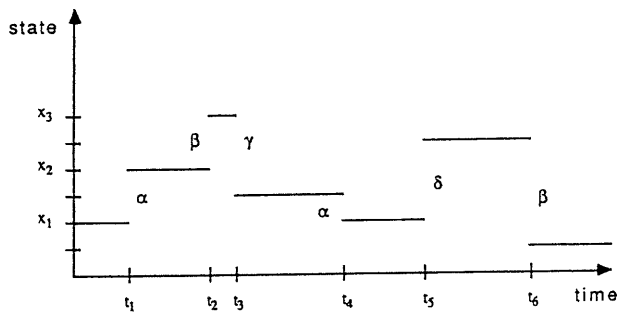


Fig. 1. State trajectory of a DES.

The state transitions of a DES are called *events* and, as indicated in the figure, these may be labeled with the elements of some alphabet. These labels usually indicate the physical phenomenon that caused the change in state. For example, in a communication protocol typical event labels are "time out," "packet received," "packet sent"; while in a manufacturing system, events of interest are "machine breakdown," "machine repaired," "part accepted," etc.

The many areas in which DESs arise and the different aspects of behavior relevant in each area have led to the development of a variety of DES models. For example, a common simplifying assumption is to ignore the times of occurrence of the events and consider only the order in which they occur. This leads to so-called *logical DES models*. In such models a system trajectory is specified simply by listing (in order) the events that occur along the original sample path. For example, in a logical model, the partial trajectory shown in Fig. 1 is reduced to the string of events $\alpha\beta\gamma\alpha\delta\beta\cdots$. This simplification is justified when the model is to be used to study properties of the event dynamics that are (or should be) independent of specific timing assumptions. On the other hand, in some applications the timing information is crucial, and must be included in the model. This leads to so-called *timed* or *performance models*. These can be further classified as *nonstochastic* (e.g., timed Petri nets, the max-algebra) or *stochastic* (e.g., Markov chains, queueing networks, generalized semi-Markov processes) according to whether the timing is known *a priori*, or is modeled by making suitable statistical assumptions. These models are intended for the study of properties explicitly dependent on interevent timing.

Logical models have been successfully used to study the qualitative properties of DESs in a variety of applications. For example, logical models have been employed in areas such as concurrent program semantics [62], [48], and communicating sequential processes [21], [23], [40], [67]; synchronization in operating systems [12]; supervisory control [55]–[57], [74]–[76]; communication protocols [9], [24], [25], [41]; logical analysis of digital circuits [16]; fault tolerant distributed computing [13], [14]; and database protocols [26], [27]. In such applications, the formulation and analysis of the model typically proceeds as follows. One first specifies the set of admissible event trajectories, i.e., the physically possible sequences of events. This may be done using some form of state transition structure (e.g., automata [56] or Petri nets [46]), by means of a set of algebraic equations [21], [23], or by a logical calculus such as temporal logic [39]. In the cases of interest the admissible event trajectories form a strict subset of the set of all (mathematically) possible event orderings. Given a property of event sequences, one then seeks to determine if each admissible trajectory has the desired property. Or, in a control context, one asks if it is possible to modify (by control action) the set of admissible trajectories so that each event trajectory has the desired property. Typical properties of interest include the following: stability (e.g., state convergence [12]), correct use of resources (e.g., mutual exclusion [3], [49]), correct event ordering (e.g., database consistency [26], [27]), desirable dynamic behavior (e.g., no deadlock/livelock [67]), and the coordination of constituent processes to achieve a desired goal (e.g., distributed consensus [13], [14]).

In addition to the study of qualitative system properties, logical models can also be used as a basis of computation, e.g., verification or synthesis of DESs. In such applications the issue of computational complexity is a key concern. For example, the number of states in a transition structure for specifying the admissible event trajectories may depend exponentially on some system parameter. In such cases simple algorithms for verification or synthesis (e.g., searching over the state space) rapidly become computationally intractable. Despite this fact, use of the models in this role has not been abandoned. Indeed in some applications (e.g., circuit verification, protocol correctness) they offer the only known means of automated analysis (see e.g., [4], [24], [25],

[61]). Typically, one tries to mitigate the complexity by the use of aggregation or modularity, or by exploiting hierarchical or the other special structures [31], [32].

Nonstochastic timed models are similar in spirit to the logical models, except of course that event timing must now be taken into account. Formulation of a model proceeds by specifying the admissible set of event trajectories together with the associated timing. This can be done using a suitable transition structure (e.g., timed Petri nets), some formal calculus (e.g., temporal logic), or a formal simulation language. Simple models of this form have been used in the design of signal processing arrays [29], the analysis of periodic behavior in manufacturing systems [10], and the specification of real-time control structures [30], [42], [43].

Stochastic performance models are somewhat different in spirit. In these models it is usually trivial to specify the set of admissible state trajectories. The difficult part of modeling and analysis is in defining a useful measure on this set, and using this to determine the distributions and moments of the variables of interest. Such models have been successfully used to study both quantitative and qualitative features of systems such as communication networks [58], [59]; simple queueing networks [6], [19], [28], [34], [60], [65]; stochastic scheduling problems [71]; and manufacturing systems [2], [5], [7], [11].

Stochastic models, like their deterministic counterparts, also have their limitations. For example, in general they are sufficiently complex to prohibit analytic treatment. As a result, for many problems of interest no closed form solutions are known. Moreover, when the qualitative form of the solution is known, a means of computing it may not be available, and even when closed form solutions are known (e.g., Jackson networks), the form of the solution may be sufficiently complicated that its use is difficult, if not infeasible. In many cases this complexity has limited results to those of a qualitative nature, and has led to the use of simulation as a tool for quantitative analysis [15], [78]. This, in turn, has motivated the work by Ho and others [20], [65], [77], on perturbation methods for estimating gradients of performance measures. More detailed accounts of these aspects of stochastic models can be found in several other papers in this special issue.

No single approach to the modeling and analysis of DESs will suffice for all problems of interest. Each of the above models has its own applications, virtues, and limitations. In all models, there is a need for higher level descriptions of system dynamics, for aggregation, for a concise means of system and problem specification, for the study of interesting subclasses of systems with special structure, and for modular and hierarchical system and controller decompositions. A variety of these issues are addressed in the context of logical models in the remainder of the paper.

III. A Logical DES Model

In a logical model of a DES, we are interested in the sequences or strings of events that the process can generate. Let Σ denote the finite set of event labels, and Σ^* denote the set of all finite strings of elements of the set Σ, including the empty string ϵ. For convenience, we often refer to an element of Σ as an event. A string, say $u = \sigma_1\sigma_2 \cdots \sigma_k \in \Sigma^*$, represents a partial event sample path. We say partial because there may be more events after σ_k. The set of all admissible, i.e., physically possible, sample paths is then a subset L of Σ^*. It is customary to call a subset of Σ^* a *language* over the alphabet Σ.

A string u is a *prefix* of a string $v \in \Sigma^*$ if for some $w \in \Sigma^*$, $v = uw$. If v is an admissible sample path, then clearly so are all the prefixes of v. If we define the *prefix closure* of $L \subseteq \Sigma^*$ to be the language

$$\bar{L} = \{ u: uv \in L \text{ for some } v \in \Sigma^* \}$$

then we require $\bar{L} = L$. In this case we say that L is *prefix closed*.

Thus we model the behavior of a DES as a prefix closed language L over the event alphabet Σ. Each $u \in L$ represents a possible (partial) event sample path of the DES. For example, a trivial DES with two events $\{\alpha, \beta\}$ that operates so that the events α and β always occur alternately, with α or β occurring first has the behavior

$$L = \{\epsilon, \alpha, \beta, \alpha\beta, \beta\alpha, \alpha\beta\alpha, \cdots\}.$$

On the other hand, if we let $|w|_\alpha$ denote the number of occurrences of the event α in the string w, and set

$$L = \{w \in \Sigma^*: \text{for each prefix } u \text{ of } w, |u|_\alpha \leq |u|_\beta\}$$

then L represents a DES in which the number of occurrences of the event α is always less than or equal to the number of occurrences of the event β. This might be the case, for example, if in a manufacturing system, α corresponds to taking a part from a buffer, and β corresponds to placing a part in the buffer.

To construct more elaborate examples, it is convenient to have a means of language representation. For our purposes, this is most conveniently done by representing a DES by its state description, or transition structure. We do not insist that these state representations be finite, or that they have a specific structure. This allows for the possibility of counters and other useful infinite state devices. Our approach here is to first investigate the qualitative structural features of the problems of interest, then later turn to the issue of computation. This is in contrast to a purely computational approach, in which some form of finite representation would be assumed from the outset. Most of the qualitative results which we survey are in fact representation independent, i.e., their validity does not depend on a specific representation. Thus having first analyzed the qualitative properties of interest using one form of representation, we can turn to other forms of representation (perhaps for specific classes of DESs) that are suitable for computation.

To represent a behavior L we proceed as follows. A *generator* G is an automaton [22] consisting of a state set Q, and initial state q_0, and transition function $\delta: \Sigma \times Q \to Q$ (in general a partial function). As we remarked above, the state set Q need not be finite, although this is clearly an interesting special case. By the set of events possible at state q we mean the set $\Sigma(q) \subseteq \Sigma$ such that for each $\sigma \in \Sigma(q)$, $\delta(\sigma, q)$ is defined. The fact that the transition function is a partial function simply reflects the fact that in general $\Sigma(q)$ is a proper subset of Σ. One can think of G as a directed graph with node set Q and an edge $q \to q'$ labeled σ for each triple (σ, q, q') such that $q' = \delta(\sigma, q)$. We interpret G as a device that starts in its initial state q_0 and executes state transitions, i.e., generates a sequence of events, by following its graph. State transitions are considered to occur spontaneously, asynchro-

nously, and instantaneously, and their occurrence is signaled by the corresponding event label $\sigma \in \Sigma$. The generator G will play the role of the "plant" in the sense of control theory. The term "generator" is nonstandard, but better suited to our interpretation than, say, "automaton" or "machine."

The transition function δ of G is extended to a (partial) function on $\Sigma^* \times Q$ by defining $\delta(\epsilon, q) = q$ and

$$\delta(w\sigma, q) = \delta(\sigma, \delta(w, q))$$

whenever $q' = \delta(w, q)$ and $\delta(\sigma, q')$ are defined. Henceforth we write $\delta(w, q)!$ as an abbreviation for the phase "$\delta(w, q)$ is defined." In terms of the graph of G, $\delta(w, q)!$ simply means that there is a path in the graph starting from q that is labeled by the consecutive elements of the string w.

The closed behavior of G is defined to be the prefix closed language

$$L(G) = \{w : w \in \Sigma^* \text{ and } \delta(w, q_0)!\}.$$

Every prefix closed language $L \subseteq \Sigma^*$ has such a representation. If L has a finite state generator representation, then it is a *regular language* [22]. These constitute a proper subset of the languages over Σ.

Example 3.1: Fig. 2 shows a simple generator. This is intended to model a machine with three states, labeled *I* (Idle), *W* (Working), and *D* (Down); the initial state is labeled with an entering arrow → (state *I*); and there are four possible state transitions, each labeled by the associated observed event from the event set $\Sigma = \{\alpha, \beta, \lambda, \mu\}$. The closed behavior of G is simply the set of all strings obtained by starting in the state *I* and following the graph. In the formalism of regular expressions this can be written as

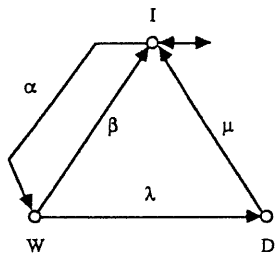


Fig. 2. A simple generator.

$$L(G) = (\alpha\beta + \alpha\lambda\mu)^*(\epsilon + \alpha + \alpha\lambda).$$

Several refinements of the above model are possible. For example, one can add to the definition of G a subset of *marker states* $Q_m \subseteq Q$, and define the marked behavior of G (with respect to Q_m) to be

$$L_m(G) = \{s : s \in L \text{ and } \delta(\tilde{w}, q_0) \in Q_m\}.$$

We interpret $L_m(G)$ as a distinguished subset of the generated sequences that could represent completed "tasks" (or sequences of tasks) carried out by the underlying DES that G is intended to model. There is no implication that the generating action halts after the completion of some marked sequence—marker states of G need not be "final" states. When G is fixed, we often abbreviate $L(G)$ and $L_m(G)$ to simply L and L_m, respectively.

It is always the case that $L_m(G) \subseteq L(G)$, and hence that $\overline{L_m(G)} \subseteq L(G)$, i.e., that every prefix of L_m is an element of L. It is natural to require (or at least desire) equality in the last expression, i.e., that every string in $L(G)$ be a prefix of a string in $L_m(G)$. In this case, every event sample path in $L(G)$ can be extended to a completed "task" in $L_m(G)$. In this case, we say that G is *nonblocking*.

For example, in Fig. 2 the marker states are indicated by exiting arrows → (state *I*). Thus the language marked by G is simply the set of all strings in the graph that begin and end at the state *I*. They represent completed work cycles of the machine. It is clear that every string in $L(G)$ can be extended to one that reaches the state *I*. Hence this generator is nonblocking. In the formalism of regular expressions we can write

$$L_m = (\alpha\beta + \alpha\lambda\mu)^*.$$

Example 3.2: Consider two asynchronous, independent users of a single resource, each modeled by the simple cyclic generators G_1, G_2 of Fig. 3.

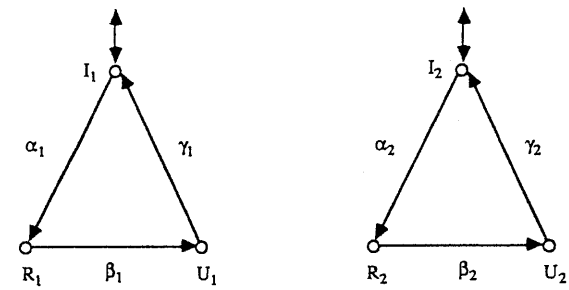


Fig. 3. Two users of a resource.

Each user has three states *I* (Idle), *R* (Request), and *U* (Use) with the transitions shown. We model the joint operation of these users by the "shuffle product" $G = G_1 \| G_2$ of G_1 and G_2. This is the DES determined by the concurrent actions of G_1 and G_2 under the assumption that these actions are asynchronous and independent. The states of G are ordered pairs (X, Y) where X is a state of G_1 and Y is a state of G_2, and the transitions of G are either of the form $(X, Y) \rightarrow (X', Y)$ where $X \rightarrow X'$ is a transition in G_1, or of the form $(X, Y) \rightarrow (X, Y')$ where $Y \rightarrow Y'$ is a transition in G_2. The graph of G is shown in Fig. 4.

$L(G)$ consists of all words over the alphabet $\Sigma = \{\alpha_1, \beta_1, \gamma_1, \alpha_2, \beta_2, \gamma_2\}$ corresponding to paths in the graph that begin at (I_1, I_2). Similarly $L_m(G)$ corresponds to all paths in the graph that both start and end in the state (I_1, I_2). Clearly the generator G is nonblocking.

This example illustrates the use of the shuffle product to model asynchronous systems. This is a special case of a more general product called the *synchronous* product that allows for the possibility of synchronous events in the argument processes. It is clear that while it is always possible (conceptually) to form the shuffle (or, more generally, the synchronous) product, and that this may be easily computed for small examples such as this one, there may be a problem in computing such models for systems with a large number of components. This follows from the observation that the number of states in the product increases exponentially with the number of components. For problems requiring computation (e.g., controller synthesis) one tries to find techniques that avoid actually carrying out this construc-

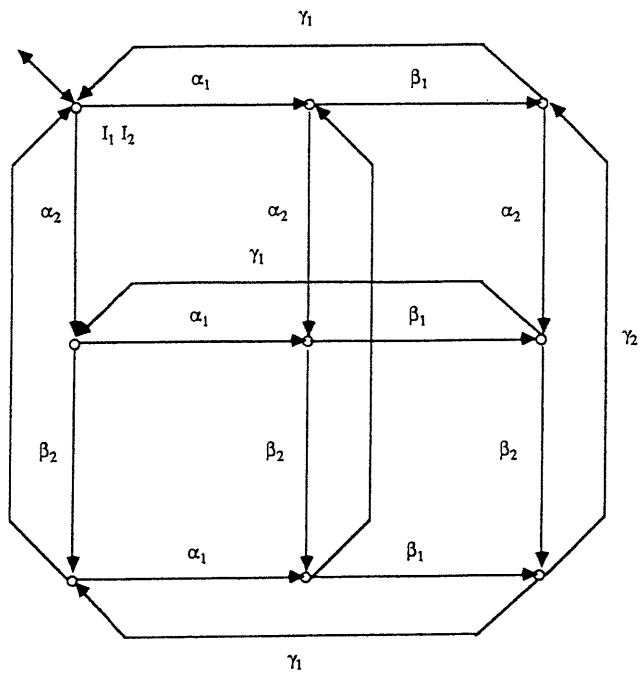


Fig. 4. The shuffle product generator.

tion, except possibly for subproblems of a reasonable size. This will be addressed in later sections of this paper.

IV. Controllability and Supervision of DESs

Our DES model as described so far is simply a spontaneous generator of event strings without a means of external control. To control a DES we postulate that certain events of the system can be disabled (i.e., prevented from occurring) when desired. This enables us to influence the evolution of the system by prohibiting the occurrence of key events at certain times. To model such control we partition the set of events Σ into *uncontrollable* and *controllable* events: $\Sigma = \Sigma_u \cup \Sigma_c$. The events in Σ_c can be disabled at any time, while those in Σ_u model events over which the controlling agent has no influence, e.g., machine breakdown in a manufacturing system, loss of a packet in a communication channel, external disturbances, etc.

A *control input* for G consists of a subset $\gamma \subseteq \Sigma$ satisfying $\Sigma_u \subseteq \gamma$. If $\sigma \in \gamma$, then σ is *enabled* by γ (permitted to occur), otherwise σ is *disabled* by γ (prohibited from occurring). The condition $\Sigma_u \subseteq \gamma$ means that the uncontrollable events are always enabled. Let $\Gamma \subseteq 2^{\Sigma}$ denote the set of control inputs. A DES represented by the generator G equipped with a set of control inputs Γ is called a *controlled DES* (CDES). In what follows, Γ is fixed, and for convenience we refer to a CDES by its underlying generator G.

Control of a CDES G consists of switching the control input through a sequence of elements $\gamma, \gamma', \gamma'', \cdots$ in Γ, in response to the observed string of previously generated events. Such a controller will be called a *supervisor*. In introducing the concept of a supervisor we follow the standard practice in control theory of distinguishing rather sharply between the "plant" (or object to be controlled), and the agent doing the controlling. While in certain instances (perhaps mainly of computer system applications) this distinction might seem artificial, it tends to simplify the problem of defining exactly what controlled behavior is required, as well as what constraints on behavior are imposed *a priori* by the underlying physical entities with which the design problem originates.

Formally, a supervisor is a map

$$f: L \to \Gamma$$

specifying for each possible string of generated events w the control input $f(w)$ to be applied at that point. Our objective will be to design a supervisor that selects control inputs in such a way that the given CDES G behaves in obedience to various constraints. Roughly, constraints can be viewed as requiring that certain undesirable sequences of events are not permitted to occur, while at the same time, certain other desirable sequences are permitted to occur.

When a CDES G is supervised by the supervisor f it operates as before, except that it obeys the additional constraint that, following the generation of a string w, the next event must be an element of $f(w) \cap \Sigma(\delta(w, q_0))$. Denote the closed loop system of G supervised by f by (G, f). The behavior of (G, f), denoted $L(G, f)$, or simply L_f when no confusion is possible, is formally defined as follows:

i) $\epsilon \in L_f$; and
ii) $w\sigma \in L_f$ iff $w \in L_f$, $\sigma \in f(w)$, and $w\sigma \in L$.

If G is equipped with a set of marker states, then the *language controlled by f in G* is also of interest. This is the language

$$L_m(G, f) = L_m(G) \cap L_f.$$

This is simply that part of the original marked language that survives under supervision. If L_m indeed represents completed tasks, then this language is clearly important, since it indicates those tasks that will be completed under supervision. When no confusion is possible we will abbreviate $L_m(G, f)$ to simply L_{mf}.

In practice one may require an alternative representation of the supervisor f. For this we can use a state realization in terms of an automaton together with an output map (this is sometimes used as the definition of a supervisor; see e.g., [56]). Let $T = (\Sigma, X, \xi, x_0)$ be an automaton, and $\phi: X \to \Gamma$. We say that the pair (T, ϕ) *realizes* the supervisor f if for each $w \in L_f$

$$\phi(\xi(w, x_0)) = f(w).$$

This simply says that the value of f on the string w can be found by first applying w to T causing T to be driven from its initial state to some state x, and then computing $\phi(x)$. Thus T is a standard automaton whose state transitions are driven by the events in Σ.

In standard control terminology G plays the role of the "plant" (object to be controlled), T functions as an "observer" or "dynamic compensator," and ϕ is the "feedback." It is possible to visualize supervision as a simple interconnection of G and T through ϕ: the outputs of G drive the state transitions of T, and in turn, the state of T determines the next control input γ through ϕ (see Fig. 5).

It is also possible to realize a supervisor simply as another DES S. In this case the control action of S on G is implicit in the transition structure of S. In detail, if $s \in L(G, f)$ then we require $s \in L(S)$, and $s\sigma \in L(S)$ only if $\sigma \in f(s)$. In addition, if $s \in L(G, f)$, $s\sigma \in L(G)$, and $\sigma \in f(s)$, then $s\sigma \in L(S)$. The first condition ensures that those transitions disabled by f do not appear in the transition structure of S; while the second

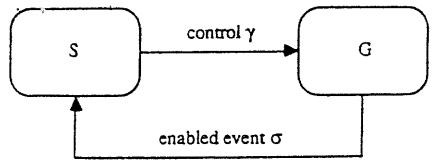


Fig. 5. Supervision of a DES.

condition ensures that those transitions enabled by f, and which are possible in G, do appear in the transition structure of S. S and G are assumed to run in parallel in the following fashion. An event σ can occur when $S \times G$ is in the state (x, q) only if σ is possible in both S and G at that point; and results in the state change $(x, q) \rightarrow (x', q')$ where $x \rightarrow x'$ and $q \rightarrow q'$ are the transitions in S and G, respectively, under σ. This form of supervisor realization can be obtained from the state realization (S, ϕ) by suitably trimming the transition structure of S (cf. [63]).

From the point of view of the theory, we do not require a supervisor to have a finite state realization. Thus counters, unbounded queues, and other useful infinite state devices can appear as part of the automaton S. However, the case when S is finite state is clearly a special case of interest. We say that f is a *finite state supervisor* if it has a finite state realization.

The basic problem in supervisory control is to modify the open loop behavior of a given DES G so that it lies (as a set) within some prescribed range. This desirable range may be specified by actually giving the desired closed loop behavior, by giving a behavior within which the closed loop behavior must be contained, or by specifying such sets indirectly through other qualitative performance objectives. One is thus led to consider the following problem: given a CDES G with behavior L, what closed loop behaviors $K \subseteq L$ can be achieved by supervision? The key to the resolution of this (and many related) questions is the concept of controllability.

Say that $K \subseteq \Sigma^*$ is *controllable* if

$$\overline{K}\Sigma_u \cap L \subseteq \overline{K}.$$

This condition requires that for any prefix of a string in K, i.e., any $w \in \overline{K}$, if w followed by an uncontrolled event $\sigma \in \Sigma_u$ is in L, i.e., $w\sigma \in L$, then it must also be a prefix of a string in K, i.e., $w\sigma \in \overline{K}$. In this sense $\overline{K}$ is conditionally invariant under the action of Σ_u. Since uncontrollable events cannot be prevented from occurring, it is intuitively clear that if such an event occurs along a sample path in K, then the extended sample path must remain in K in order for K to be a feasible closed loop behavior.

More generally, the aim of supervision is not to modify L per se, but instead to achieve a prescribed language for L_{mf}, and to do so while preserving the desired nonblocking property. Conditions under which this is possible can also be stated in terms of language controllability [56, prop. 5.1 and theorem 6.1]. We summarize both results in

Proposition 4.1: Fix a nonblocking DES G with closed behavior L and marked behavior L_m.

1) For nonempty $K \subseteq L$ there exists a supervisor f such that $L_f = K$ iff K is prefix closed and controllable.
2) For nonempty $K \subseteq L_m$ there exists a supervisor f such that $L_{mf} = K$, and the closed loop system is nonblocking iff K is controllable, and $\overline{K} \cap L_m = K$.

When K satisfies the condition $\overline{K} \cap L_m = K$ in part 2) of the above result, we say that K is L_m-*closed*. Thus achieving $L_f = K$ is possible precisely when K is closed and controllable, and achieving $L_{mf} = K$ and the nonblocking property is possible precisely when K is L_m-closed and controllable. In addition, the proof of this result provides an algorithm for constructing a realization (S, ϕ) of the required supervisor f from a generator for the controllable language K.

We can now use our characterization of the possible controlled behaviors to study the structure of this family of languages. For a given $K \subseteq \Sigma^*$ let $C(K)$ denote the family of controllable sublanguages of K. $C(K)$ is always nonempty since $\varnothing$ is controllable. Our second main result on controllability is that the family $C(K)$ is closed under set union, and has a unique supremal element under the partial order of subset inclusion, i.e., there exists a unique largest controllable language $K^\uparrow$ such that $K^\uparrow \subseteq K$. Note that $K^\uparrow$ may be the empty language. A similar result holds for language intersection when the languages are, in addition, restricted to be closed. In this case one would be interested in the family of closed and controllable languages containing a given language K.

The closure of $C(K)$ under set union indicates that if a given language K is not controllable, then there is a natural controllable approximation to K, namely the largest controllable language contained in K. This language preserves the restrictions imposed by K, while requiring the least amount of control action. It can thus be regarded as the "optimal" or "minimally restrictive" approximation to K.

We remark that the abstract existence of an optimal solution is far from being an invariable property of supervisory control problems; indeed, it fails in certain extensions of our model to accommodate communication delay and true concurrency (synchronous events) [30]. In general, however, when optimality is present the problem of computing even a *feasible* solution to a control problem may be considerably simplified because there is just one "natural" solution to look for (technically, one is not required to scan over possibly awkward partially ordered candidate solution sets in which a semilattice property of closure under "join" fails to hold). In addition, an idealized abstract model where optimality is obtained may serve as a practical guide to finding sufficient conditions for problem solvability in the more realistic, though messier, situations referred to above [30].

For the finite state case, i.e., finite state generators are provided for L (the open loop behavior) and K (the desired behavior), an algorithm for the computation of $K^\uparrow$ is described in [76]. The basis of this algorithm is the following fixpoint characterization. Let $P(\Sigma^*)$ denote the power set of Σ^*, i.e., the set of all languages over Σ^*, and define the operator

$$\Omega: P(\Sigma^*) \rightarrow P(\Sigma^*)$$

by

$$\Omega(J) = K \cap \sup \{T: T \subseteq \Sigma^*, T = \overline{T}, T\Sigma_u \cap L \subseteq \overline{J}\}.$$

Then $K^\uparrow$ is the largest fixpoint of Ω, i.e., the largest language J such that $\Omega(J) = J$. Furthermore, if we set

$$K_0 = K$$

$$K_{j+1} = \Omega(K_j)$$

then

$$\lim_{j \to \infty} K_j = K^{\uparrow}.$$

If the generators for L and K have m and n states, respectively, then this scheme converges after at most mn iterations. Since the computation of Ω is itself bounded by a polynomial in m and n, this means that the computation of $K^{\uparrow}$ is of polynomial complexity in m and n.

We have not addressed the practical problem of actually obtaining the generator for the desired closed loop behavior K. This has simply been taken as the means of task description, and hence assumed to be available. The problem of how to define or specify the control task in other terms has not been addressed.

Some supervisor synthesis problems can be recast in the framework of Markov decision theory, and dynamic programming with a minimax criterion can be used to compute the desired control [70]. Such an approach provides an alternative computation scheme, but does little to illuminate the algebraic and structural properties of interest.

The following example is rather simple but illustrates in an amusing way the application of the concepts of controllability and supervision.

Example 4.1: A cat and a mouse are placed in the maze shown in Fig. 6. Each doorway in the maze is either for the

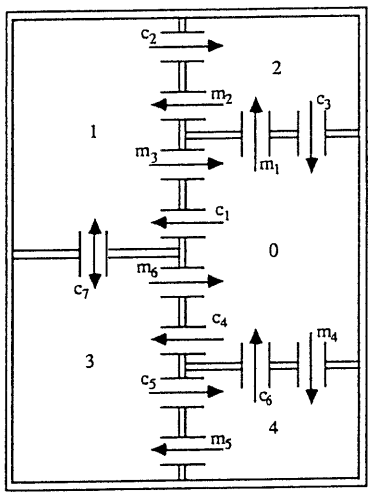


Fig. 6. Maze for cat and mouse.

exclusive use of the cat, or for the exclusive use of the mouse, and must be traversed in the direction indicated. Our cat and mouse are somewhat whimsical but could represent, for example, autonomous vehicles in an automated factory.

Let $\Sigma = \{c_i, m_j : 1 \leq i \leq 9, 1 \leq j \leq 6\}$. We model the movement of the cat and the mouse in the maze by the generators G_1 and G_2 over Σ shown in Fig. 7. Here state i corresponds to room i, and a transition $i \rightarrow j$ corresponds to traversing the door c_k between rooms i and j.

For our joint model of the cat and the mouse we adopt the shuffle product $G = G_1 \| G_2$. The states of G are ordered pairs ij where i is a state of G_1 and j is a state of G_2. G is tabulated in Fig. 8.

We assume that each door, with the exception of c_7, can be opened or closed as required in order to control the movement of the cat and the mouse. Our objective is to find

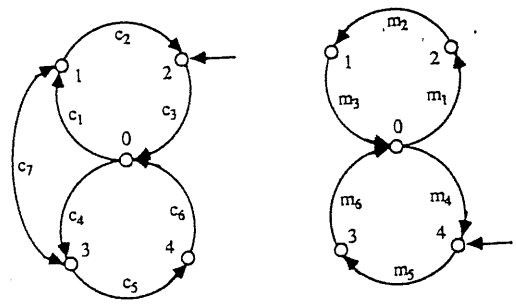


Fig. 7. Generator models for cat and mouse.

the control scheme which permits the cat and the mouse the greatest possible freedom of movement but which also guarantees that

a) the cat and the mouse never occupy the same room simultaneously, and that
b) it is always possible for the cat and the mouse to return to the initial state, i.e., the state in which the cat is in room 2, and the mouse is in room 4.

Let G be the product generator, set $Q_m = \{(2,4)\}$, and $\Sigma_u = \{c_7\}$. This is a controlled generator with $\Gamma = \{\gamma : c_7 \in \gamma\}$. A solution can be computed using the results described in this section as follows. A generator for the maximal language K that satisfies the stated contraints is obtained from G by removing states which violate constraint a), and from which one cannot reach Q_m. The supremal controllable sublanguage contained in K is then computed using the iteration method previously described. This yields a generator for the supremal controllable language $K^{\uparrow}$ contained in K. The necessary control action to implement $K^{\uparrow}$ can be computed at the same time, and a realization of the required supervisor can be based on this and the generator for $K^{\uparrow}$. This supervisor realization can be further simplified by a supervisor aggregation technique [56, sec. 8]. The result is the supervisor shown in Fig. 9. This has two states (0 and 1), and the status of each controlled event (enabled (1) or disabled (0)) is shown for each state in the accompanying table.

The control strategy which this supervisor implements can be summarized as follows. If the cat and the mouse occupy their respective initial rooms, then both are given the opportunity to move to a new room, i.e., c_3 and m_5 are open. If the cat leaves room 2, then the mouse is isolated in room 4, i.e., m_5 and c_5 are closed, and the cat is free to roam the rest of the maze. Similarly, if the mouse leaves room 4, then the cat is isolated in room 2, and the mouse is permitted access to those rooms from which it can return to room 4, i.e., rooms 0, 3, 4.

V. Modular Synthesis

One means of reducing the complexity of logical DES models is to incorporate additional structure into the model; and one means by which this can be done is through modular problem specification and supervisor construction. Modularity allows complex problems to be decomposed into simpler components, and greater structure and flexibility to be incorporated into the controller. Essentially this involves defining an algebra of DESs, and a corresponding algebra of supervisors. To be useful, such algebras must be compatible with the concepts of controllability, nonblock-

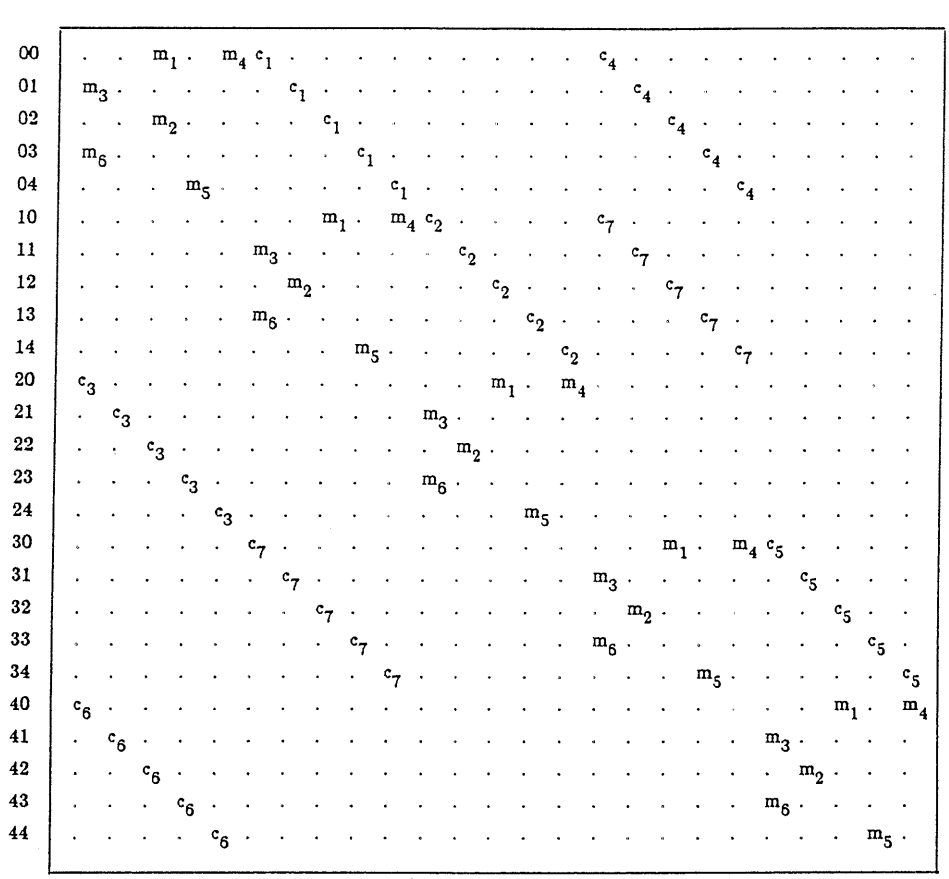


Fig. 8. Product generator for cat and mouse.

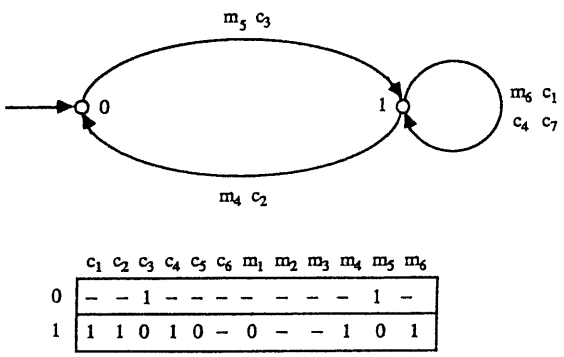


Fig. 9. Supervisor for cat and mouse.

ing supervision, and supremal controllable languages introduced in the previous section. These problems have been examined in [55], [75], and are reviewed below.

Of the two natural operations on languages, union and intersection, intersection is often the most useful in modular problem specification. For example, it may be desired to control a DES so as to satisfy two constraints simultaneously. If each constraint is specified as a desired controlled language K_i, then the overall constraint is specified by the intersection $K_1 \cap K_2$.

For prefix closed languages it is readily checked that $K_1 \cap K_2$ is controllable whenever both K_1 and K_2 are controllable. However, without the prefix closure assumption this need not be true. The key to compatibility between language intersection and controllability is the concept of nonconflicting languages. For $K_1, K_2 \subseteq \Sigma^*$ it is always the case that

$$\overline{K_1 \cap K_2} \subseteq \overline{K_1} \cap \overline{K_2}$$

i.e., a prefix of a word common to K_1 and K_2 is also a prefix of K_1 and K_2. K_1 and K_2 are said to be *nonconflicting* when we have equality in this expression:

$$\overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}.$$

In other words, two languages are nonconflicting if, whenever they share a prefix, they also share a word containing this prefix. For example, any two closed languages are nonconflicting.

To illustrate an application of this concept, consider a refinement of the control problem mentioned above. Fix a CDES G and suppose that we must find a supervisor f so that $L_{mf} = K_1 \cap K_2$, where $K_1, K_2 \subseteq L(G)$. From the previous section we know that this is possible iff $K = K_1 \cap K_2$ is controllable, and $L_m(G)$-closed. This may be tested in a modular fashion as follows:

Proposition 5.1: Let $K_1, K_2 \subseteq \Sigma^*$ be nonconflicting. If K_1 and K_2 are both $L_m(G)$-closed and controllable, then $K_1 \cap K_2$ is $L_m(G)$-closed and controllable.

Similarly, one can use the nonconflicting property to determine conditions under which the operation of taking the supremal controllable sublanguage of K commutes with language intersection:

Proposition 5.2: Let $K_1, K_2 \in \Sigma^*$. If $K_1^\uparrow$ and $K_2^\uparrow$ are nonconflicting, then $K_1^\uparrow \cap K_2^\uparrow = (K_1 \cap K_2)^\uparrow$.

Corollary 5.1: If $K_1, K_2 \subseteq \Sigma^*$ are closed languages, then $(K_1 \cap K_2)^\uparrow = K_1^\uparrow \cap K_2^\uparrow$.

These results indicate that the supremal controllable sublanguage of $K_1 \cap K_2$ can be found by first computing $K_1^\uparrow$ and $K_2^\uparrow$, checking that these languages are nonconflicting, and, if so, forming their intersection.

Assume that the generator G has m states. Then for a regular language K specified by an n state automaton, the complexity of the verification $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$ is $O(nm)$. Thus to compute $K_1 \cap K_2$ (an $O(n^2)$ computation), and then to check the controllability of K is a computation of complexity $O(n^2m)$. On the other hand, to verify that K_1 and K_2 are each controllable is a computation of complexity $O(nm)$. These upper bounds are conservative, but they strongly suggest that when the modular procedure is applicable it can offer a significant reduction in computational complexity.

If f and g are supervisors, then their *conjunction* $f \wedge g$ is the supervisor

$$f \wedge g(w) = f(w) \cap g(w).$$

If f and g have state realizations (S, ϕ) and (T, ψ), respectively, then this simply amounts to operating the automata S and T in parallel $(S \times T)$, together with a feedback map $\phi \diamond \psi$ formed by intersecting the sets of enabled events of ϕ and ψ.

It is clear from the definition that supervisor conjunction is a commutative and associative operation, and it is easy to construct a supervisor that acts as an identity. However, the conjunction of two nonblocking supervisors need not be nonblocking. Indeed, if two supervisors implement contradictory objectives, then their conjunction will yield a supervisor that permits the closed loop system to become blocked. In this respect the utility of supervisor conjunction depends on the nonconflicting property:

Proposition 5.3: Let f_1 and f_2 be nonblocking supervisors for G. Then the supervisor $f = f_1 \wedge f_2$ has

1) $L(G, f) = L(G, f_1) \cap L(G, f_2)$; and
2) $L_m(G, f) = L_m(G, f_1) \cap L_m(G, f_2)$.

Furthermore, f is nonblocking for G iff $L_m(G, f_1)$, and $L_m(G, f_2)$ are nonconflicting.

To illustrate the above ideas we consider the following example.

Example 5.1: In a factory two machines M_1, M_2 operate in parallel to feed a buffer B; a third downstream machine M_3 takes parts from the buffer (Fig. 10). Each machine M_i operates as in Example 3.1; i.e., each has three states: I_i (Idle), W_i (Working), and D_i (Down); and events $\{\alpha_i, \beta_i, \gamma_i, \mu_i\}$ as shown in Fig. 2. The buffer B is simply an automaton driven by M_1, M_2, and M_3.

The system operates as follows. Machine M_1 takes a workpiece (event α_1), and either successfully completes processing and passes the workpiece to the buffer (event β_1); or breaks down and discards the workpiece (event γ_1), but in that case may later be repaired (event μ_1). Machine M_2 operates in the same way. Machine M_3 operates essentially the same way, but takes its workpiece from the buffer, provided one is there. Aside from these constraints the machines operate asynchronously and independently.

The informal control specifications are the following:

1) The number of parts in the buffer must be kept less than a fixed integer N.
2) Machine M_3 must not attempt to take a part from the buffer if it is empty.
3) Machines M_1 and M_2 are repaired in order of breakdown.
4) Machine M_3 has priority of repair over machines M_1 and M_2.

As the generator for the system we take

$$G = M_1 \| M_2 \| M_3.$$

This has 29 states and 108 transitions.

To formalize the specifications we bring in the following DESs. First, to model constraints 1) and 2) (constraints on the buffer), we define the DES A_1 shown in Fig. 11 (here we assume $N = 3$). Similarly the breakdown/repair of machines M_1 and M_2 is modeled by the DES A_2 in Fig. 12 and the breakdown/repair of machine M_3 by the DES A_3 in Fig. 13. In each of these three figures only the events that are constrained are indicated; other events have been omitted for clarity. Omitted events can be reinserted by adjoining a selfloop labeled by the missing events to each state (in subsequent computation we assume this has been done).

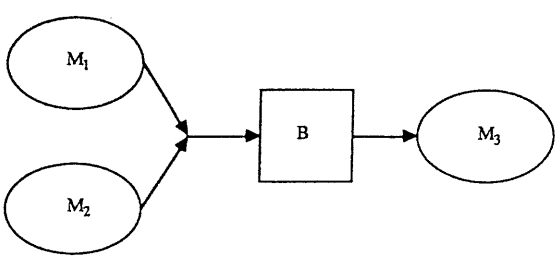


Fig. 10. A simple factory.

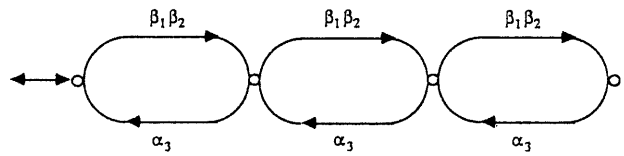


Fig. 11. Buffer constraint for $N = 3$.

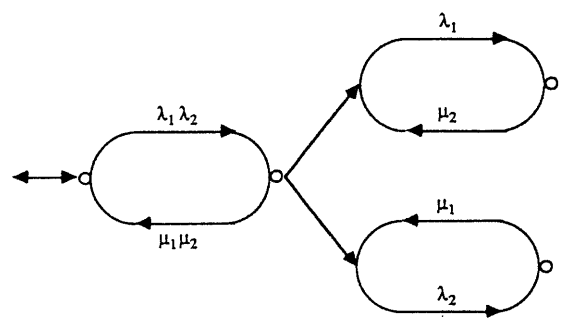


Fig. 12. Breakdown/repair of machines one and two.

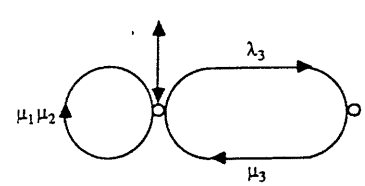


Fig. 13. Breakdown/repair of machine three.

We first consider nonmodular, or "monolithic" supervision. The specifications can be combined by taking the intersection $A = A_1 \cap A_2 \cap A_3$. This is a DES with 32 states and 248 transitions. Using the algorithm introduced in Section IV we then compute a supervisor that implements the supremal controllable sublanguage of A. This turns out to have 96 states and 302 transitions—too many to display here. Although this supervisor is guaranteed to satisfy the constraints, and to do so in a minimally restrictive fashion, it is rather cumbersome to implement and in no way reflects the modularity of the original constraints.

To develop a modular solution we proceed as follows. First, to prevent buffer overflow we compute: s_1 = number of empty buffer slots − number of feeder machines at work. This can be done by the automaton S_1 shown in Fig. 14. To

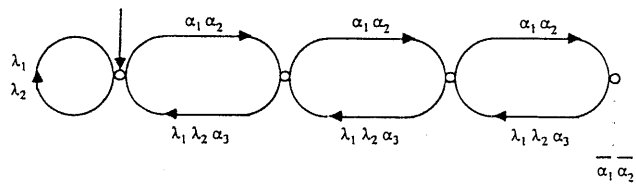


Fig. 14. Supervisor to prevent buffer overflow.

this we can add the feedback that disables α_1 and α_2 in state 0. This yields a supervisor realization that prevents buffer overflow.

To prevent buffer underflow we compute: s_2 = number of full buffer slots. This can be done using the automaton shown in Fig. 15. If, to this automaton, we add a feedback

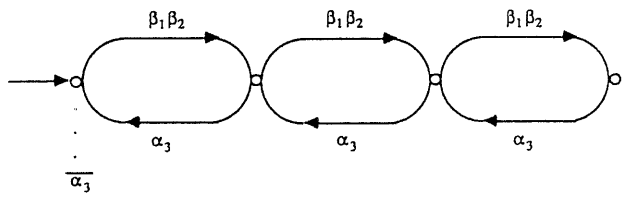


Fig. 15. Supervisor to prevent buffer underflow.

that disables α_3 when the buffer is empty (state 0), then we have a supervisor to ensure that constraint 2) is satisfied. By adjoining to A_2 and A_3 suitable feedback maps, these can also be converted to supervisors which will ensure that constraints 3) and 4), respectively, are satisfied.

Using the results of this section, it can be checked that correct and optimal supervision of the factory is enforced by the conjunction of the above supervisors. Clearly this modular supervisor is much simpler to design and implement than the corresponding monolithic supervisor, to which it is equivalent in control action.

VI. Partial Observations and Observability

Up to this point it has been assumed that all of the events generated by a CDES can be directly observed by the supervising agent. However, in situations involving decentralized or hierarchical control we usually only have local or partial observations.

To model a DES with partial observations we bring an additional alphabet Σ_o, the *observation alphabet*, and a *projection* (or *mask*) $P: \Sigma \to (\Sigma_o \cup \{\epsilon\})$. The idea is that $P(\sigma)$ is the event observed when the generator undergoes a state transition labeled by σ. Thus the events in Σ are observed through the map P. Those events $\sigma \in \Sigma$ with $P(\sigma) = \epsilon$ are not observed at all (they are erased), while events $\alpha, \beta \in \Sigma$ with $P(\alpha) = P(\beta)$ can no longer be distinguished. The special case in which P simply erases some of the events in Σ occurs frequently, and will be called a *natural projection*. In this case we can take $\Sigma_o \subseteq \Sigma$ and define P by

$$P(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_o \\ \epsilon, & \text{if } \sigma \notin \Sigma_o. \end{cases}$$

The action of a projection P is extended to strings by defining $P(\epsilon) = \epsilon$ and

$$P(s\sigma) = P(s)P(\sigma) \quad \text{for } s \in \Sigma^*, \sigma \in \Sigma.$$

Denote the equivalence kernel of this extended map by *ker P*, i.e., *ker P* is the equivalence relation on Σ^* defined by $(s, s') \in ker\ P$ (or $s \equiv s' (\text{mod } P)$) iff $P(s) = P(s')$.

If a CDES G has behavior L, then under the projection P this is observed as the language $P(L) \subseteq \Sigma_o^*$. To respect this information constraint a supervisor for G is now required to be a map $g: P(L) \to \Gamma$. Similarly, a supervisor realization becomes an automaton $S = (\Sigma_o, X, \xi, x_0)$ together with a map $\phi: X \to \Gamma$. Algebraically this is equivalent to restricting attention to supervisors $f: L \to \Gamma$ for which there exists a map g: $P(L) \to \Gamma$ such that for each $w \in L$, $f(w) = g(P(w))$. Such a supervisor is said to be a *P-supervisor*.

Given a nonempty closed language $K \subseteq L(G)$ one may now ask under what conditions does there exist a P-supervisor f such that $L(G, f) = K$? This is a simple supervisory control problem with partial information. To resolve the problem we introduce the concept of language observability. Define the binary relation act_K on Σ^* as follows. The pair $(s, s') \in act_K$ if $s, s' \in K$ implies that there does not exist $\sigma \in \Sigma$ such that either

$$s\sigma \in K \quad \text{and} \quad s'\sigma \in L(G) - K \quad \text{or} \quad s\sigma \in L(G) - K \quad \text{and} \quad s'\sigma \in K.$$

In other words $(s, s') \in act_K$ if all the one step continuations of s and s' that remain in $L(G)$ yield the same result with respect to membership of K. The relation act_K is a tolerance relation on Σ^*, i.e., it is symmetric and reflexive, but not in general transitive. Finally, define the closed language K to be *P-observable* with respect to G if

$$ker\ P \preceq act_K$$

i.e., if $P(s) = P(s')$ then $(s, s') \in act_K$. Roughly, this means that the projection P retains sufficient information to decide whether or not, after the occurrence of some event, the resultant string is in K.

We can now state [36, theorem 2.1], [cf. 9, lemma 2.3]:

Proposition 6.1: Let $K \subseteq L(G)$ be closed and nonempty. Then there exists a P-supervisor f such that $L(G, f) = K$ iff K is controllable and P-observable.

If K does not satisfy the conditions of the above proposition, then it is natural to consider the possibility of approximating K as was done in the complete information case. Unfortunately, it turns out that, in a given situation, a unique maximal controllable and observable sublanguage of K need not exist. To obtain such approximations in a reasonable and practical manner we consider the fol-

lowing subclass of languages. Say that a closed language $K \subseteq L(G)$ is *P-normal* if

$$K = L(G) \cap P^{-1}(P(K)).$$

This condition simply requires that K is the union of some cosets of *ker P* intersected with L. This is displayed conceptually in Fig. 16. In the figure each square represents a

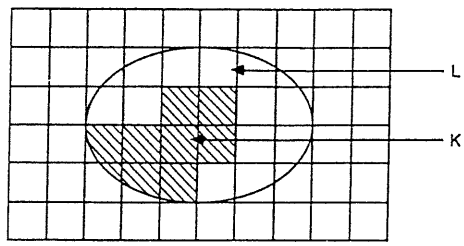


Fig. 16. A *P*-normal language.

coset of *ker P*. If a normal sublanguage contains a point of a coset, then it must contain all points of that coset that lie in L. Alternatively, K is *P*-normal iff it is the largest sublanguage of L having $P(K)$ as its projection; thus K is determined uniquely by its projection and the constraints imposed by L.

Normal languages have a number of special properties that make them of interest. The one that concerns us here is that for $s \in L(G)$ we can decide if $s \in K$ from $P(s)$ alone. This in turn gives us [36, prop. 4.1]:

Proposition 6.2: Let $K \subseteq L(G)$ be closed and *P*-normal. Then K is *P*-observable.

The converse of the above proposition need not be true, namely a *P*-observable language need not be *P*-normal. However, the normal languages form a subset of the observable languages that is algebraically better behaved. In particular, this family is closed under set union, so the largest *P*-normal sublanguage of a given closed language always exists [36, prop. 4.2]. Thus the *supremal controllable P-normal sublanguage* of a given closed language $K \subseteq L(G)$ always exists, and provides a quasi-optimal controllable and observable approximation to K should this language fail to satisfy the conditions of Proposition 6.1 [9, sec. 3], [36, sec. 4].

The above results give the flavor of supervisory control with partial information. The literature contains further discussion of more complex supervision problems with partial observations [9], [36]; as well as a discussion of some of the related computational issues [8], [9], [70], [74]. The supremal controllable *P*-normal sublanguage of a given language can be computed using algorithms similar in spirit to the lattice iteration described in Section III. Several algorithms for this and related computations are discussed in some detail in [8], [9], and [74]. It is well known that most discrete decision and control problems with partial information are computationally difficult, i.e., NP-complete or worse [44], [45]. As shown in [70] this is also the case for general supervisory control problems with partial information, and this makes suboptimal schemes, e.g., normal sublanguages, all the more interesting as a basis for computation.

Example 6.1: Two stations labeled A and B are connected by a single one-way track from A to B. The track consists of four sections, with stoplights (*) and detectors (!) installed

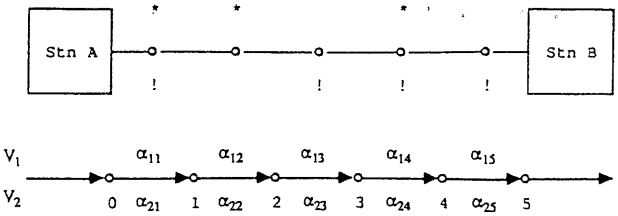


Fig. 17. Guideway with lights (*) and detectors (!).

at various junctions (Fig. 17). Two vehicles V_1, V_2 use the guideway simultaneously. V_j is in state 0 when at A, in state i when in section i, $i = 1, \cdots, 4$, and in state 5 when at B.

To avoid collisions, control of the stoplights must ensure that V_1 and V_2 never travel on the same section of track simultaneously. The generator for V_j is shown in Fig. 17; controllable events are α_{ji} for $i = 1, 2, 4$, and α_{j2} is an unobservable event. The plant is obtained by taking $G = V_1 \| V_2$. From this, a generator for the desired closed loop behavior K is obtained by removing the states (i, i), $i = 1, \cdots, 4$. This is a generator with 30 states and 40 transitions. One then sets out to compute the unique supremal controllable normal sublanguage of K. This turns out to be described by a generator with 26 states and 32 transitions. That part of the generator corresponding to the situation when V_2 starts first is shown in Fig. 18. The corresponding supervisor imple-

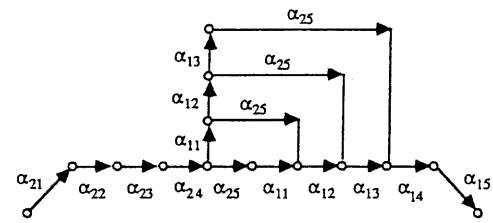


Fig. 18. Supervisor for guideway.

ments the following policy: since V_2 starts first, V_1 must wait at A until V_2 enters track section 4. V_1 may then continue into track section 3, but may not enter track section 4 until V_2 enters station B. Light #2 is not used.

VII. Decentralized Supervision

Decentralized supervision is based on the idea of local agents (supervisors) simultaneously supervising a DES G, with each agent having access only to "local" information and "local" controls (Fig. 19). Such a situation is similar to

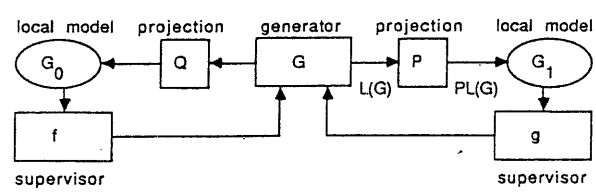


Fig. 19. Decentralized supervision.

modular supervision except that we have added the additional constraint of partial information and partial control. In the current framework the overall control task, as embodied in some constraint language $K \subset L(G)$, often

splits naturally into subtasks for which "local" supervisory controllers are fairly easy to obtain. This makes decentralized control attractive. However, the question then arises whether such local controllers acting concurrently achieve the desired control objective, and if so, whether they achieve it in an "optimal" manner.

Fix a DES G over the alphabet Σ and assume that we have a projection $P: \Sigma^* \to \Sigma_o^*$. For simplicity we shall assume that P is a natural projection (see Section VI). As before, we interpret Σ_o as the set of events that can be observed by the local supervisor.

The local observation of the behavior of G is $P(L(G))$. This can be represented by a *local model* G_o, with $L(G_o) = P(L(G))$, and $L_m(G_o) = P(L_m(G))$. G_o embodies the local agent's model of the global process G. Denote, as usual, the sets of controlled and uncontrolled events of G by Σ_c, and Σ_u, respectively. Then define

$$\Sigma_{uo} = \Sigma_u \cap \Sigma_o \quad \text{and} \quad \Sigma_{co} = \Sigma \cap \Sigma_o.$$

Thus the local agent sees the control decomposition $\Sigma_o = \Sigma_{co} \cup \Sigma_{uo}$, and only has access to the corresponding local input set Γ_o.

Now assume that we are given a nonempty closed language $E_o \subseteq \Sigma_o^*$, which we interpret as a constraint on admissible behavior at the local level, i.e., we want to find a local supervisor $f_o: P(L) \to \Gamma_o$ such that $L(G_o, f_o) \subseteq E_o$. For this we select f_o so as to synthesize the supremal (locally) controllable sublanguage K_o of $L(G_o) \cap E_o$, i.e., $L(G_o, f_o) = K_o$. At the global level this results in the closed loop behavior $\hat{K} = L(G) \cap P^{-1}(K_o)$.

On the other hand, the local specification E_o corresponds to the global constraint language $E = L(G) \cap P^{-1}(E_o)$, and we can find a global supervisor $f: L \to \Gamma$ so as to synthesize the supremal controllable sublanguage K of E, i.e., $L(G, f) = K$.

It is intuitively clear that we must have $\hat{K} \subseteq K$. But when do we have $K = \hat{K}$, i.e., when is $\hat{K}$ globally optimal? For this we have [37, theorem 3.1]:

Proposition 7.1: Local supervision is globally optimal (i.e., $\hat{K} = K$) iff K is P-normal. In that case, events in $\Sigma_c - \Sigma_{co}$ never need to be disabled.

In practice, use of the local supervisor f_o will be attractive because of its relatively simple structure and ease of synthesis. One would like to be able to justify its use without actually computing and comparing it with the global structure K or the corresponding supervisor f since, in general, these are complicated and expensive to compute. Sometimes this can be done on the basis of several sufficient conditions which do not require K to be explicitly computed [37]. However, this will not be pursued further here.

We now turn to the case of multiple local controllers. Assume that nonempty (not necessarily pairwise disjoint) local subalphabets $\Sigma_i \subseteq \Sigma$, $i \in I$, are given, and let $P_i: \Sigma^* \to \Sigma_i^*$ denote the corresponding natural projections. As before we set

$$\Sigma_{ci} = \Sigma_c \cap \Sigma_i \quad \Sigma_{ui} = \Sigma_u \cap \Sigma_i$$

and let G_i denote the local model of agent i.

A set of local supervisors $\{ f_i, i \in I \}$ acting concurrently on G is equivalent to the single global supervisor f defined by

$$f(s) = \wedge_{i \in I} f_i(s) \quad s \in L(G).$$

Of course not every supervisor can be represented in this fashion. A supervisor f for G is said to be a *decentralized supervisor* precisely when it can be represented in the above form. Decentralization imposes a restriction on the class of admissible supervisors, and hence on the class of realizable controlled behaviors.

A natural first question is the following: given a nonempty closed $K \subset L(G)$, when is it possible to find a decentralized supervisor f such that $L(G, f) = K$? For each $\sigma \in \Sigma_c$, define the index set $I_\sigma = \{i: \sigma \in \Sigma_i\}$. Say that K is $\{P_i\}$-observable if for $\sigma \in \Sigma_c$, strings $\{s_j: j \in I_\sigma\}$ in K and $s' \in K$ the conditions

1) $s_j \sigma \in K$ for all $j \in I_\sigma$;
2) $s' \sigma \in L(G)$;
3) $P(s_j) = P(s')$ for all $j \in I_\sigma$;

together imply that

$$s' \sigma \in K.$$

Then, in the spirit of Proposition 6.1, we have [9, lemma 4.2]:

Proposition 7.2: Let K be a nonempty closed sublanguage of $L(G)$. There exists a decentralized supervisor f such that $L(G, f) = K$ iff K is controllable and $\{P_i\}$-observable.

While clearly theoretically important, the above result has the computational disadvantage that it requires working with the global structure K. Consider now the situation where we specify the constraint on the controlled behavior by $E = \cap_i P_i^{-1}(E_i)$, where $E_i \subseteq \Sigma_i^*$. In other words, the global constraint can be reduced to the simultaneous satisfaction of local constraints expressed in the sublanguages Σ_i. Let K denote the supremal (globally) controllable sublanguage of E, and K_i denote the supremal (globally) controllable sublanguage of $L(G) \cap P^{-1}(E_i)$, $i \in I$. As a first step in the analysis of this situation we have [37, prop. 4.1]:

Proposition 7.3: Let $f_i (i \in I)$ be supervisors for G with $L(G, f_i) = K_i$, $i \in I$. Then $L(G, \wedge_i f_i) = K$.

The above result simply says that the concurrent operation of globally optimal decentralized supervisors is globally optimal. Of course the result is most likely to be useful just when the f_i can be designed and implemented at the local level. For this let J_i denote the supremal (locally) controllable sublanguage of $E_i \cap P_i(L(G))$, let f_i be a local supervisor with $L(G_i, f_i) = J_i$, and set

$$\hat{K}_i = L(G) \cap P_i^{-1}(J_i).$$

$\hat{K}_i$ is the global behavior resulting from local control synthesis by the ith agent. Say that G is *locally controllable*, with respect to the family of sublanguages $\{E_i\}$, if we have $K_i = \hat{K}_i (i \in I)$. With these definitions we can summarize the state of affairs as [37, theorem 4.1]:

Proposition 7.4: Let G be locally controllable. Then L(G, $\wedge_i f_i$) = K.

This result says that if G is locally controllable, then decentralized supervision of G is globally optimal.

Further details, and a number of examples of decentralized control, are discussed in [9], [37], and [74]. Since decentralized control is a partial information problem, similar comments to those given at the end of Section VI apply to the issue of computational complexity. The design and analysis of distributed protocols and controllers is known to be a computationally difficult task. While the model discussed here gives insight and qualitative results, it remains to explore issues such as aggregation and approximation which may help in computational applications.

VIII. Sequential Behavior

There are a number of possible further extensions of the basic model defined in Section II. Some of these are discussed in [17], [18], [30]–[33], [49]–[54], [68], [69], and [79]. In this section we describe an extension of the model to include infinite sequences of events as developed in [49]–[51], [67], [68]. Modeling the behavior of a DES as a set of infinite sequences can offer several advantages, including the modeling of nonterminating processes, addressing issues such as livelock and fair concurrency, and distinguishing between transient and nontransient behavior.

Let Σ^ω denote the set of all infinite sequences over the alphabet Σ. We model the behavior of a DES as a subset B of Σ^ω. A subset of Σ^ω is usually termed an *ω-language* over Σ.

The basic means of analysis is to consider the sequential behavior as the limit, in a suitable sense, of an appropriate string language. The string language represents the finite time behavior, and the sequential behavior the limiting behavior over time. For this we need the concept of the prefix of an ω-language, and the adherence of a string language. For each sequence $e \in \Sigma^\omega$ let e^j denote the string consisting of the first j elements of e. By definition $e^0 = \epsilon$. The *prefix* of $B \subseteq \Sigma^\omega$ is the set $pr(B) \subseteq \Sigma^*$ defined by

$$pr(B) = \{e^j : e \in B, j \geq 0\}$$

i.e., $pr(B)$ is the set of all strings in Σ^* that form a prefix of a sequence in B including the empty string ϵ. For $K \subseteq \Sigma^*$ the *adherence* of K is the ω-language K^∞ consisting of those sequences $e \in \Sigma^\omega$ with infinitely many prefixes in K. For example, the prefix of the ω-language $(ab)^\omega = \{abababab \cdots\}$ is the string language $(ab)^*(\epsilon + a) = \{\epsilon, a, ab, aba, \cdots\}$; and the adherence of the string language $a^* = \{\epsilon, a, aa, aaa, aaaa, \cdots\}$ is the ω-language $a^\omega = \{aaaaaa \cdots\}$.

An ω-language B can be specified in terms of a generator model as follows. To the generator $G = (\Sigma, Q, \delta, q_0)$ we adjoin a subset of states $Q_m \subseteq Q$. Roughly, G recognizes the sequence $e = \sigma_1\sigma_2\sigma_3 \cdots$ if there is a state trajectory of G under e that intersects the set Q_m infinitely often. The set of all sequences so recognized constitutes the ω-language *recognized* by G. Denote this set by $B(G)$. In the case that G is finite state, this model is called a (deterministic) Büchi automaton, and the class of languages so recognized is called the class of (deterministic) regular ω-languages.

It is clear that $pr(B(G)) \subseteq L(G)$, in general without equality. Equality implies that every string in $L(G)$ is a prefix of some sequence in $B(G)$. Roughly, this means that the system is never prevented from producing a sequence in $B(G)$. Hence when $pr(B) = L$, we say that G is *nonblocking*.

The introduction of infinite sequences brings in some interesting topological issues. There is a natural topology on Σ^* corresponding to the limiting operations mentioned above.[1] The easiest way to introduce this topology is through the metric:

$$\rho(e_1, e_2) = \begin{cases} 1/n, & \text{if } e_1^{n-1} = e_2^{n-1} \text{ and } e_1(n) \neq e_2(n); \\ 0, & \text{if } e_1 = e_2. \end{cases}$$

This measures the distance between two sequences of events as the reciprocal of the index of the first place in which they differ. The topological closure of a set $B \subset \Sigma^\omega$ with respect to the above topology is denoted $\overline{B}$, and for subsets $B \subseteq S \subseteq \Sigma^\omega$, we say that B is *closed relative to* S if $\overline{B} \cap S = B$.

As before, a supervisor for G is a map f that specifies the current control action as a function of the previously generated events. Since at any given time only a finite number of events have been generated, this means that a supervisor is a map

$$f: L(G) \to \Gamma.$$

When f supervises G the closed loop string language is $L(G, f)$, and the closed loop ω-language $B(G, f)$ is defined by

$$B(G, f) = B(G) \cap L(G, f)^\infty.$$

As before, when no confusion is possible, we abbreviate $L(G, f)$ to L_f and $B(G, f)$ to B_f. We say that f is *nonblocking for* G if the controlled system (G, f) is nonblocking.

It is natural to now inquire what controlled sequential behaviors B' can be achieved by supervision. This can be answered in terms of controllability of the finite behavior of G and an appropriate topological condition on B'. The details are stated in [49, prop. 3.1], [69, prop. 4.3]:

Proposition 8.1: If $B' \subseteq B(G)$ is nonempty, then there exists a nonblocking supervisor f for G such that $B(G, f) = B'$ iff

i) $pr(B')$ is controllable; and
ii) B' is topologically closed relative to $B(G)$, i.e., $\overline{B'} \cap B(G) = B'$.

When the two conditions of the proposition are satisfied, we say that $B' \subseteq B(G)$ is a *controllable ω-language*.

It is readily shown that the set of controllable ω-languages is closed under finite set union, but not, in general, under countable set union. Despite this fact, in certain situations it is still true that there is a unique supremal controllable sequential behavior contained in a prescribed ω-language B'. It is sufficient, for example, to impose the additional constraint that B' be closed relative to $B(G)$ [49, prop. 3.2]:

Proposition 8.2: If $B' \subseteq B(G)$ is closed relative to $B(G)$, then there exists a unique supremal controllable ω-language $B'^\uparrow$ contained in B'.

For some synthesis problems this result is adequate. However, for the analog of the general synthesis problems considered in [56] we need a stronger notion of controllability. This is discussed in detail along with related computational issues in [69].

IX. Some Complexity Issues

In computational applications of DES models, one is interested in the computational complexity of the relevant decision and synthesis problems. In our context this reduces to the complexity of verifying concepts such as controllability and observability, and the complexity of synthesizing appropriate supervisors.

The basic supervisory control problems we have discussed are described in terms of a generator G and a sublanguage K of $L(G)$. Fix the alphabet Σ, assume that G is finite state with m states, and that K has a finite state realization with n states. We measure the size of an instance of a supervisory problem by $\max(m, n)$. Then, a supervisor synthesis problem is said to be *polynomially decidable* if, given an instance of the problem, it is possible to decide in a time

[1] The closed sets of this topology are precisely the adherences of the prefix closed string languages.

bounded by a polynomial in the size of the instance whether or not it is solvable. Similarly, we shall say that the problem is *polynomially solvable* if, given a solvable instance of the problem, it is possible to synthesize a solution in a time bounded by a polynomial in the size of the particular instance.

The controllability of a given $K \subseteq L(G)$ is polynomially decidable. This can be seen by applying the algorithm for computing supremal controllable sublanguages given in [76, sec. 6]. According to this algorithm, one first takes the intersection of the generator for K with G, an operation of complexity $O(mn|\Sigma|)$. This yields a generator with at most mn states. For each state of this generator one checks a subset inclusion, each check being a computation of complexity $O(|\Sigma|\,|\Sigma_n|)$. Thus the controllability of K can be checked in $O(mn|\Sigma_u|\,|\Sigma|)$ time.

Consider the problem of synthesizing a supervisor so that the closed loop behavior is a prescribed language K. Recall that there exists a supervisor f for G such that $L(G, f) = K$ iff K is closed and controllable. Since both of these conditions can be checked in polynomial-time, the problem is polynomially decidable. Assume that K is a closed and controllable sublanguage of $L(G)$. The proof of Proposition 4.1 provides a scheme for the construction of a realization of a supervisor to implement K [56, prop. 5.1]. The time complexity of this scheme is polynomial in mn. Thus the problem is polynomially solvable.

If the closed language $K \subseteq L(G)$ is not controllable, then one can approximate K by its largest controllable sublanguage. The computation of $K^\uparrow$ using the algorithm outlined in Section IV (see [76, sec. 6]) is of time complexity $O(m^2n^2)$. So $K^\uparrow$ can be computed in polynomial-time.

Now consider the problem of synthesizing a supervisor f so that the closed loop behavior contains the language K_1, and is contained in the language K_2. This problem is solvable iff the largest controllable sublanguage of K_2 contains K_1 [56, sec. 9]. If K_1 and K_2 are specified by finite state generators with n_1 and n_2 states, respectively, then $K_2^\uparrow$ can be computed in a time bounded by a polynomial in m and n_2, and the inclusion $K_1 \subset K_2^\uparrow$ can be checked in a time bounded by a polynomial in n_1 and n_2. Thus the synthesis problem is polynomially decidable. When solvable, it reduces to the first synthesis problem with $K = K_2^\uparrow$, and hence is polynomially solvable.

Not all problems in this setting are polynomially decidable. Consider, for example, the problem of supervisor realization. Given a supervisor f it is natural to consider the problem of realizing f by an automaton with the least number of states. If we assume that one finite state realization of f, say (S, ϕ), is given, then the minimal realization problem is equivalent to the minimization of a partially specified sequential machine. There are two approaches to the problem. First, one can look for congruences on the state set X of S finer than $\ker\phi$. A *congruence* is an equivalence relation on X with the property that, for each $\sigma \in \Sigma$, if $x \equiv y$ and $\xi(\sigma, x)$ and $\xi(\sigma, y)$ are both defined, then

$$\xi(\sigma, x) \equiv \xi(\sigma, y).$$

This is a very useful construction. However, if we translate, with some minor modifications, the results of Pfleeger [47] into the current setting we have:

Proposition 9.1: Let (S, ϕ) be a finite state supervisor realization. The decision problem: does there exist a congruence on S finer than $\ker\phi$, and having at most k equivalences classes, is NP-complete.

A second, and more general approach to the minimization problem involves the use of invariant covers. It can be shown that the minimal realizations of f can be obtained from any given realization (S, ϕ) by examining the invariant covers of S finer than ϕ. This was explored in [72]. However, a simple extension of Pfleeger's previous result yields:

Proposition 9.2: Let $f: K \to \Gamma$ be a finite state supervisor. The decision problem: does there exist a finite state realization (S, ϕ) of f that has at most k states, is *NP*-complete.

These results indicate that in general minimal supervisor realization is unlikely to be feasible for problems of significant size (assuming $P \neq NP$). It is important to be aware of this fact but also important to keep it in perspective: in many supervisory control problems, structured solutions are probably of greater merit than minimal solutions.

Control problems with partial information are known to be computationally difficult. In [70] it is shown that the conditions of Proposition 6.1, i.e., the existence of a *P*-supervisor, can be checked in polynomial-time. This is a positive result. However, under the assumption that $P \neq NP$, several other control problems of interest are shown to require nonpolynomial-time algorithms. The use of *P*-normal languages to avoid this problem was discussed in Section VI.

We have seen examples of systems constructed using the shuffle product. Such product systems arise naturally when modeling the concurrent operation of several asynchronous, or partially synchronous, discrete dynamical systems. One of the principal difficulties in dealing with these systems is that the number of states increases exponentially with the number of components. Thus synthesis methods based on searching over the product state space are unlikely to be computationally feasible. However, it is also the case that product systems possess special structure, and in some instances this can be exploited in the solution of decision and synthesis problems. This is discussed further in the next section.

X. Product Systems

In this section we continue our discussion of computational complexity, but specialize to the interesting case of a DES composed of a finite set of asynchronous components. In our setting such systems have been modeled by the shuffle (or, more generally, the synchronous) product. For notational convenience let $[1, p]$ denote the interval of integers $\{i: 1 \leq i \leq p\}$.

For each $i \in [1, p]$ let $G_i = (\Sigma_i, Q_i, \delta_i, q_{0i})$ be a DES with control partition $\Sigma_i = \Sigma_{ci} \cup \Sigma_{ui}$. Assume that the event sets Σ_i are pairwise disjoint, that each G_i has a nonempty sequential behavior B_i, and that $Pr(B_i) = L(G_i)$. Let $\Sigma = \cup_i \Sigma_i$ and $P_i: \Sigma \to \Sigma_i$ be the natural projection.

The *product system* $G = \|_{i=1}^{p} G_i$ is the shuffle product of the DES $G_1, \cdots, G_p$. This DES is defined so that its sequential behavior is the ω-language

$$B(G) = \{e: e \in \Sigma^\omega \text{ and } P_i(e) \in B_i,\ i \in [1, p]\}$$

i.e., $e \in B(G)$ iff for each i, $e(t) \in \Sigma_i$ infinitely often, and if e_i is the subsequence of e consisting of all elements in Σ_i, then $e_i \in B_i$. The control partition of G is $\Sigma = \Sigma_u \cup \Sigma_c$ with $\Sigma_u = \cup_i \Sigma_{ui}$, and as usual, its set of admissible controls is denoted by Γ.

If each G_i has n states, then G has n^p states. Clearly, from the point of view of computation, this is a possible problem. If p is bounded, say $p \leq 4$, then the number of states in G is bounded by a polynomial in n, and the results of Section IX apply. However, we are interested in the case when both p and n are variable. In this case, we say that a supervision problem for a product system is *polynomially decidable* if the time required to determine whether or not it is solvable is bounded by a polynomial in n and p. Similarly, a control problem is *polynomially solvable* if we can synthesize a finite realization of a solution in a time bounded by a polynomial in n and p. It is convenient to construct this realization as a product system. In this case the dynamics of the realization are specified by a set of finite state automata S_i, $i \in [1, p]$, together with a feedback map ϕ mapping their product state space $X = \Pi_{i=1}^{p} X_i$ into Γ. Since the components $S_1, \cdots, S_p$ of the realization operate in parallel, there is no need to actually construct their product. This is in the spirit of the modular synthesis discussed in Section V. In general such a realization will be nonminimal, but this will be offset by the fact that it is structured, and more readily synthesized.

We restrict attention here to control problems requiring pure coordination. Roughly, this means that the supervisor does not modify the open loop behaviors of the individual DESs, but only constrains how they interact by controlling the relative order of events. Formally, we say that a supervisor f for a product system is a *coordinator* if, for every set of p event sequences, $e_1, e_2, \cdots, e_p$ with $e_i \in B_i$, $i \in [1, p]$, there exists a sequence e in the closed loop behavior B_f such that for all $i \in [1, p]$.

$$P_i(e) = e_i.$$

Consider, for example, the following coordination problem:

Mutual Exclusion (MEX): Let $\overline{Q}_i \subseteq Q_i$ be p given nontransient subsets, and k be a fixed integer with $1 \leq k < p$. Synthesize (if possible) a nonblocking coordinator f for G such that for each $e \in B_f$, and each $j \geq 1$, after the first j events of e at most k of the generators G_i satisfy $q_i \in \overline{Q}_i$.

The problem requires that at most k of the generators G_i are in the designated subsets of states at any one time, and this must be done without changing the open loop behavior of any generator. The assumption that the subset $\overline{Q}_i$ of the state set of the generator G_i is nontransient simply means that there exists an admissible state trajectory for G_i that visits $\overline{Q}_i$ infinitely often.

Let B' be the subset of $B(G)$ that satisfies the mutual exclusion constraint. It is easy to see that B' is closed, and hence closed relative to $B(G)$. Thus by Proposition 8.2 there exists a unique supremal controllable sequential behavior $B'^{\uparrow}$ contained in B'. It can be shown from the definitions that if an instance of MEX is solvable, then it has a minimally restrictive solution, and that this solution implements the closed loop behavior $B^{\uparrow}$. On the issue of computational complexity one can state [49, theorem 5.1]:

Proposition 10.1: MEX is polynomially decidable. Moreover, given a solvable instance of MEX it is possible to synthesize a minimally restrictive solution in polynomial-time.

That the problem is of polynomial complexity is due to the fact that it can be decoupled and analyzed in terms of the component DESs. The proof of the proposition provides a simple polynomial-time test by which the solvability of MEX can be decided. In addition, if the problem is solvable, a supervisor is provided which will ensure that the requirements of MEX are met.

As a second example of a coordination problem consider:

Uncontrolled String Exclusion (USE): Let $w_i \in \Sigma_{ui}^*$, $1 \leq i \leq p$, be p nonempty, nontransient strings of uncontrolled events, and let the string w be formed from a shuffling of the w_i. Synthesize (if possible) a nonblocking coordinator f for G such that for every $e \in B_f$ and every $j \geq 1$, $e^j \neq s_1 w s_2$ for some $s_1, s_2 \in \Sigma^*$.

The problem requires that the generated sequence of events e never contains the "illegal string" w. The length of w is an additional factor in the complexity analysis.

It is readily shown that there exists a polynomial transformation of an instance of USE to an instance of MEX. Indeed, given an instance of USE set

$$X_i = \{q_i \in Q_i : \delta_i(w_i, q_i)!\}.$$

Then the instance of USE is solvable iff the instance of MEX with $k = p - 1$ and $\overline{Q}_i = X_i$ is solvable. This leads to:

Proposition 10.2: USE is polynomially decidable. Moreover, given a solvable instance of USE it is possible to synthesize a minimally restrictive solution in polynomial-time.

It is possible to pose a string exclusion problem without the assumption that the events in the string are uncontrolled. In this case the string w contains at least one controllable event, and with the exception of a few special cases, a supervisor can be based on a simple string recognizer. This problem is again polynomial [49, theorem 5.3].

An essential feature of the above problems is that they can be "decoupled" and analyzed in terms of the system components, and as a result, the required supervisor can be synthesized in a modular fashion as a product system. For this we assumed that the components of the open loop product system were independent, and that any interaction between the components could be modeled as part of a control constraint. Alternatively, one can think of the supervisor synthesis as designing the process interaction to achieve a desired behavior. While this is possible for a variety of DESs of interest, it is a serious assumption that we would like to weaken. In [17] it is shown that with just one common shared event the MEX problem cannot be solved with a polynomial-time algorithm. However, with reasonable additional restrictions on the shared events, one can ensure that the problem remains polynomial.

XI. Conclusion

This paper has provided an overview of one trend in the development of a control theory for discrete-event systems. In view of the relatively long history of prior approaches to discrete-event control design (notably discrete-event system simulation, and analysis via Petri nets, starting in the 1960s; and investigations via stochastic models, including perturbation analysis, from the early 1970s) it is perhaps surprising that attempts to evolve a synthetic, control-theoretic overview of the problem area, especially in its qualitative, logical aspects, have been both few in number and recent in appearance. Nevertheless, the control of DESs is now an established branch of control theory.

The current studies of the qualitative aspects of the control of DESs highlight the thesis that control science is defined in terms of problems and concepts, not in terms

of techniques. Stimulated by the demands of technology and by developments in computer science, new techniques are entering the field of control theory from automaton theory, formal language and formal logic, to take their place alongside the traditional mathematics like differential equations and operator theory. For both researchers and educators in the control field, the challenges remain plentiful.

References

[1] S. Aggarwal, C. Coucoubetis, and P. Wolper, "Adding liveness properties to coupled finite state machines," preprint, AT&T Bell Laboratories, Murray Hill, NJ, 1988.

[2] R. Akella and P. R. Kumar, "Optimal control of production rate in a failure prone manufacturing system," *IEEE Trans. Automat. Contr.*, vol. 31, no. 2, pp. 116–126, Feb. 1986.

[3] J. Beauquier and M. Nivat, "Application of formal language theory to problems of security and synchronization," in *Formal Language Theory—Perspective and Open Problems*, R. V. Book, Ed. New York: Academic Press, 1980, pp. 407–454.

[4] C. Berthet and E. Cerny, "An algebraic model for asynchronous circuits verification," *IEEE Trans. Comput.*, vol. 37, no. 7, pp. 835–847, July 1988.

[5] T. Bielecki and P. R. Kumar, "Optimality of zero inventory policies for unrealiable manufacturing systems," preprint, Dept. Electrical Engineering, University of Illinois, Urbana, IL, 1986.

[6] A. D. Bovopoulos and A. A. Lazar, "Optimal routing and flow control of a network of parallel processors," preprint, Dept. Electrical Engineering, Columbia University, NY, Jan. 1985.

[7] J. A. Buzacott, "Optimal operating rules for automated manufacturing systems," *IEEE Trans. Automat. Contr.*, vol. 27, no. 2, pp. 80–86, Feb. 1982.

[8] H. Cho and S. I. Marcus, "On the supremal languages of sublanguages that arise in supervisor synthesis problems with partial observation," *Mathematics Contr., Signals Syst.*, vol. 2, no. 2, pp. 47–69, 1989.

[9] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, "Supervisory control of discrete event processes with partial observations," *IEEE Trans. Automat. Contr.*, vol. 33, no. 3, pp. 249–260, Mar. 1988.

[10] G. Cohen, D. Dubois, J. P. Quadrat, and M. Voit, "A linear system-theoretic view of discrete event processes and its use for the performance evaluation in manufacturing," *IEEE Trans. Automat. Contr.*, vol. 30, no. 3, pp. 210–220, 1985.

[11] C. A. Courcoubetis and R. R. Weber, "A bin packing system for objects with sizes from finite set: necessary and sufficient conditions for stability and some applications," in *Proc. 25th Conf. Decision and Control*, (Athens, Greece), pp. 1686–1691, Dec. 1986.

[12] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Commun. ACM*, vol. 17, no. 11, pp. 643–644, Nov. 1974.

[13] M. J. Fisher, "The consensus problem in unreliable distributed systems (a brief survey)," Research Rep. YALEU/DCS/RR-273, Dept. Computer Science, Yale University, June 1983.

[14] M. J. Fisher, N. A. Lynch, and M. S. Patterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1982.

[15] G. S. Fishman, *Principles of Discrete Event Simulation*. New York, NY: Wiley, 1978.

[16] I. Gertner and R. Kurshan, "Logical analysis of digital circuits," preprint, AT&T Bell Laboratories, Murray Hill, NJ, 1987.

[17] C. H. Golaszewski and P. J. Ramadge, "Mutual exclusion problems for discrete event systems with shared events," in *Proc. 27th IEEE Conf. Decision and Control* (Austin, TX), pp. 234–239, Dec. 1988.

[18] ——, "Discrete event processes with arbitrary controls," in *Advanced Computing Concepts and Techniques in Control Engineering*, M. J. Denham and A. J. Laub, Eds., Springer-Verlag NATO ASI Series. New York, NY: Springer Verlag, 1988, pp. 459–469.

[19] B. Hajek, "Optimal control of two interacting service stations," *IEEE Trans. Automat. Contr.*, vol. 29, no. 6, pp. 491–499, June 1984.

[20] Y. C. Ho and C. Cassandras, "A new approach to the analysis of discrete event dynamic systems," *Automatica*, vol. 19, no. 2, pp. 149–167, 1983.

[21] C. A. R. Hoare, *Communicating Sequential Processes*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

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

[23] K. Inan and P. Varaiya, "Finitely recursive process models for discrete event systems," *IEEE Trans. Automat. Contr.*, vol. 33, no. 7, pp. 626–639, July 1988.

[24] R. Kurshan, "Reducibility in analysis of coordination," in *Discrete Event Systems: Models and Applications, IIASA Conference, Sopron, Hungary, Aug. 3–7, 1987*, P. Varaiya and A. B. Kurzhanski, Eds., Lecture Notes in Control and Information Sciences, vol. 103. New York, NY: Springer-Verlag, 1988, pp. 19–39.

[25] R. P. Kurshan, "Testing containment of ω-regular languages," preprint, AT&T Bell Laboratories, Murray Hill, NJ, Oct. 1986.

[26] S. Lafortune, "Modelling and analysis of transaction execution in database systems," *IEEE Trans. Automat. Contr.*, vol. 33, no. 5, pp. 439–447, May 1988.

[27] S. Lafortune and E. Wong, "A state model for the concurrency control problem in data base management systems," Memo. UCB/ERL M85/27, Electronic Systems Laboratory, College of Engineering, University of California Berkeley, CA, Apr. 1985.

[28] A. A. Lazar and M. T. Hsiao, "Network and user optimal flow control with decentralized information," preprint, Dept. Electrical Engineering, Columbia University, NY.

[29] E. A. Lee, "Data flow programming for parallel implementation of digital signal processing systems," in *Discrete Event Systems: Models and Applications, IIASA Conference, Sopron, Hungary, Aug. 3–7, 1987*, P. Varaiya and A. B. Kurzhanski, Eds., Lecture Notes in Control and Information Sciences, vol. 103. New York, NY: Springer-Verlag, 1988, pp. 135–148.

[30] Y. Li and W. M. Wonham, "On supervisory control of real-time discrete-event systems," *Inform. Sci.*, vol. 46, pp. 159–183, 1988.

[31] ——, "Controllability and observability in the state-feedback control of discrete-event systems," in *Proc. 27th IEEE Conf. Decision and Control* (Austin, TX), pp. 203–208, Dec. 1988.

[32] ——, "A state-variable approach to the modelling and control of discrete-event systems," in *Proc. 26th Annual Allerton Conf.*, Sept. 1988.

[33] ——, "Deadlock issues in supervisory control of discrete-event systems," in *Proc. 22nd Annual Conf. Information Sciences and Systems*, (Princeton, NJ), pp. 57–63, Mar. 1988.

[34] W. Lin and P. R. Kumar, "Optimal control of a queueing system with two heterogeneous servers," *IEEE Trans. Automat. Contr.*, vol. 29, pp. 696–703, Aug. 1984.

[35] F. Lin and W. M. Wonham, "Decentralized control and coordination of discrete-event systems," in *Proc. 27th IEEE Conf. Decision and Control* (Austin, TX), pp. 1125–1130, Dec. 1988.

[36] ——, "On observability of discrete-event systems," *Inform. Sci.*, vol. 44, pp. 173–198, 1988.

[37] ——, "Decentralized supervisory control of discrete-event systems," *Inform. Sci.*, vol. 44, pp. 199–224, 1988.

[38] O. Maimon and G. Tadmore, "Efficient low level control of FMS," Draft Tech. Rep. LIDS-P-1571, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, June 1986.

[39] Z. Manna and A. Pnueli, "Synthesis of communicating processes from temporal logic specifications," *ACM Trans. Programming Languages and Syst.*, vol. 6, no. 1, pp. 68–93, Jan. 1984.

[40] G. Milne and R. Milner, "Concurrent processes and their syntax," *J. ACM*, vol. 26, pp. 302–321, 1979.

[41] K. Okumura, "Protocol analysis from language structure," preprint, IBM Research, Tokyo Research Laboratory, 5-19, Tokyo, June 1988.

[42] J. S. Ostroff and W. M. Wonham, "A temporal logic approach to real time control," in *Proc. 24th IEEE Conf. Decision and Control*, (Fort Lauderdale, Florida), pp. 656–657, Dec. 1985.

[43] J. S. Ostroff, "Real time computer control of discrete event

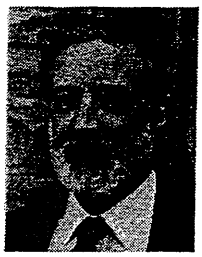
systems modelled by extended state machines: A temporal logic approach," Rep. 8618, Dept. Electrical Engineering, University of Toronto, Sept. 1986.
[44] C. H. Papadimitriou and J. N. Tsitsiklis, "On the complexity of designing distributed protocols," *Inform. Contr.*, vol. 53, pp. 211–218, June 1982.
[45] ——, "On the complexity of Markov decision processes," *Mathematics Operations Res.*, vol. 12, no. 3, pp. 441–450, Aug. 1987.
[46] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.
[47] C. P. Pfleeger, "State reduction in incompletely specified finite state machines," *IEEE Trans. Comput.*, vol. C-22, no. 12, pp. 1099–1102, Jan. 1979.
[48] A. Pneuli, "The temporal semantics of concurrent programs," in *Semantics of Concurrent Computation*, Lecture Notes in Computer Science 70. New York, NY: Springer-Verlag, 1979, pp. 1–20.
[49] P. J. Ramadge, "Some tractable supervisory control problems for discrete event systems described by Büchi automata," *IEEE Trans. Automat. Contr.*, vol. 34, no. 1, Jan. 1989.
[50] ——, "The complexity of some basic problems in the supervisory control of discrete event systems," in *Advanced Computing Concepts and Techniques in Control Engineering*, M. J. Denham and A. J. Laub, Eds., Springer-Verlag NATO ASI Series. New York, NY: Springer Verlag, 1988, pp. 171–190.
[51] ——, "Supervisory control of discrete event systems: a survey and some new results," in *Discrete Event Systems: Models and Applications, IIASA Conference, Sopron, Hungary, Aug. 3–7, 1987*, P. Varaiya and A. B. Kurzhanski, Eds., Lecture Notes in Control and Information Sciences, vol. 103. New York, NY: Springer-Verlag, 1988, pp. 69–80.
[52] ——, "Some tractable supervisory control problems for discrete event systems," to appear in *Proc. Symp. on the Mathematical Theory of networks and Systems*, (Phoenix, AZ), June 1987.
[53] ——, "Observability of discrete event systems," in *Proc. 25th IEEE Conf. Decision and Control*, (Athens, Greece), pp. 1108–1112, Dec. 10–12, 1986.
[54] ——, "A note on the fixpoint characterization of supremal controllable sublanguages," in *Proc. 21st Conf. Information Sciences and Syst.*, pp. 741–744, Mar. 1987.
[55] P. J. Ramadge and W. M. Wonham, "Modular feedback logic for discrete event systems," *SIAM J. Contr. Optimization*, vol. 25, no. 5, pp. 1202–1218, May 1987.
[56] ——, "Supervisory control of a class of discrete-event processes," *SIAM J. Contr. Optimization*, vol. 25, no. 1, pp. 206–230, Jan. 1987.
[57] ——, "Modular supervisory control of discrete event systems," in *Proc. 7th Int. Conf. Analysis and Optimization of Syst.*, (Antibes, France), pp. 202–214, June 1986.
[58] D. Raychaudhuri "Aloha with multiplacket messages and ARQ-type retransmission protocols—throughput analysis," *IEEE Trans. Commun.*, vol. 32, no. 2, pp. 148–154, Feb. 1984.
[59] ——, "Stability and optimal retransmission control of announced retransmission random access systems," preprint, RCA Laboratories, Princeton, NJ, 1985.
[60] Z. Rosberg, P. P. Varaiya, and J. C. Walrand, "Optimal control of service in tandem queues," *IEEE Trans. Automat. Contr.*, vol. 27, no. 3, pp. 600–610, June 1982.
[61] K. Sabnani, "An algorithmic technique for protocol verification," *IEEE Trans. Commun.*, vol. 36, no. 3, pp. 924–931, Aug. 1988.
[62] A. C. Shaw, "Software descriptions with flow expressions," *IEEE Trans. Software Eng.*, vol. 4, no. 3, pp. 242–254, 1978.
[63] R. Smedinga, "Using trace theory to model discrete events," in *Discrete Event Systems: Models and Applications, IIASA Conference, Sopron, Hungary, Aug. 3–7, 1987*, P. Varaiya and A. B. Kurzhanski, Eds., Lecture Notes in Control and Information Sciences, vol. 103. New York, NY: Springer-Verlag, 1988, pp. 81–99.
[64] M. Steenstrup, M. A. Arbib, and E. G. Manes, "Port automata and the algebra of concurrent processes," Computer and Information Science Tech. Rep. 81-25, University of Massachusetts, Amherst, MA, 1981.
[65] R. Suri and M. Zazanis, "Perturbation analysis is exact for the M/G/1 queue," in *Proc. 23rd IEEE Conf. Decision and Control*, (Las Vegas, NV), pp. 535–536, Dec. 1984.
[66] S. Toueg and K. Steiglitz, "Deadlock free packet switching networks," *SIAM J. Computing*, vol. 10, pp. 594–611, Aug. 1981.
[67] J. G. Thistle and W. M. Wonham, "Control problems in a temporal logic framework," *Int. J. Contr.*, vol. 44, no. 4, pp. 943–976, 1986.
[68] ——, "Supervisory control with infinite string specifications," in *Proc. 25th Ann. Allerton Conf. Communications, Control, and Computing*, pp. 327–334, Sept. 1987.
[69] ——, "On the synthesis of supervisors subject to ω-language specifications," in *22nd Ann. Conf. Information Sciences and Systems*, (Princeton, NJ), pp. 440–444, Mar. 1988.
[70] J. N. Tsitsiklis, "On the control of discrete event dynamical systems," *Math. Contr., Signals, and Syst.*, vol. 2, no. 1, pp. 95–107, 1989.
[71] P. P. Varaiya, J. C. Walrand, and C. Buyukkoc, "Extensions of the multiarmed bandit problem: the discounted case," *IEEE Trans. Automat. Contr.*, vol. 30, no. 5, pp. 426–439, May 1985.
[72] A. Vaz and W. M. Wonham, "On supervisor reduction in discrete-event systems," *Int. J. Contr.*, vol. 44, no. 2, pp. 475–491, 1986.
[73] J. Walrand, *An Introduction To Queueing Networks*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[74] W. M. Wonham, "A control theory for discrete-event systems," in *Advanced Computing Concepts and Techniques in Control Engineering*, M. J. Denham and A. J. Laub, Eds., Springer-Verlag NATO ASI Series. New York, NY: Springer Verlag, 1988, pp. 129–169.
[75] W. M. Wonham and P. J. Ramadge, "Modular supervisor control of discrete event systems," *Math. Contr., Signals, and Syst.*, vol. 1, no. 1, pp. 13–30, Jan. 1988.
[76] ——, "On the supremal controllable sublanguage of a given language," *SIAM J. Contr. and Optimization*, vol. 25, no. 3, pp. 637–659, May 1987.
[77] M. A. Zazanis and R. Suri, "Estimating second derivatives of performance measures for G/G/1 queues from a single sample path," preprint, Division of Applied Sciences, Harvard University, Cambridge, MA, June 1985.
[78] B. P. Zeigler, *Multifaceted Modeling and Discrete Event Simulation*. New York, NY: Academic Press, 1984.
[79] H. Zhong and W. M. Wonham, "On hierarchical control of discrete event systems," in *22nd Ann. Conf. on Information Sciences and Syst.*, (Princeton, NJ), pp. 64–70, Mar. 1988.

Peter J. G. Ramadge (Member, IEEE) received the B.S., B.E.E. (Hons.), and M. eng. degrees from the University of Newcastle, Australia, in 1976, 1978, and 1980, respectively, and the Ph.D. degree in electrical engineering from the University of Toronto, Ontario, Canada, in 1983.

From June–December 1978 he was a visitor in the Division of Applied Sciences, Harvard University, and from June 1983–August 1984, a postdoctoral fellow in the Systems Control Group, Department of Electrical Engineering, University of Toronto, Canada. He joined the faculty of Princeton University, Princeton, NJ, as an Assistant Professor of Electrical Engineering in September 1984. He has worked in the areas of adaptive control, stochastic control, and discrete-event systems. His current research interests are in the theoretical aspects of computer science and control theory with emphasis on applications of computers in control, learning, and signal processing.

Dr. Ramage is a member of Sigma Xi and of SIAM. In 1980 he received the Outstanding Paper Award from the Control Systems Society of the IEEE for research in the area of adaptive control; in 1982 he received the Outstanding Teaching Assistant Award from the Department of Electrical Engineering, University of Toronto; and in 1985 he was a recipient of a National Science Foundation Research Initiation Grant. He was Co-Chairman of the 22nd Con-

ference on Information Sciences and Systems, and is currently serving as an Associate Editor of the journal SYSTEMS AND CONTROL LETTERS.



W. Murray Wonham (Fellow, IEEE) received the B.S. degree in engineering physics from McGill University, Montreal, Quebec, Canada, in 1956, and the Ph.D. degree in control engineering from the University of Cambridge, Cambridge, England, in 1961.

From 1961–1969 he was associated with the Control and Information Systems Laboratory at Purdue University, the Research Institute for Advanced Studies (RIAS) of the Martin Marietta Co., the Division of Applied Mathematics at Brown University, and (as a National Academy of Sciences Research Fellow) with the Office of Control Theory and Application of NASA's Electronics Research Center. In 1970 he joined the Systems Control Group of the Department of Electrical Engineering at the University of Toronto, Canada, of which he is the current Chairman. In addition he has held visiting academic appointments with the Department of Electrical Engineering at MIT, the Department of Systems Science and Mathematics at Washington University, the Department of Mathematics of the University of Bremen, the Mathematics Institute of the Academia Sinica (Beijing), and other institutions world wide. His research interests are in the areas of stochastic control and filtering, the geometric theory of linear multivariable control, and more recently in discrete event systems from the viewpoint of formal logic and language. He has authored or coauthored about fifty research papers as well as the book *Linear Multivariable Control: A Geometric Approach*.

Dr. Wonham is a Fellow of the Royal Society of Canada. In 1987 he was the recipient of the IEEE Control Systems Science and Engineering Award.