Concurrency and Discrete Event Control

Michael Heymann

ABSTRACT: Much of discrete event control theory has been developed within the framework of automata and formal languages. This tutorial paper presents an alternative approach inspired by the theories of process-algebra as developed in the computer science literature. The framework, which rests on a new formalism of concurrency, can adequately handle nondeterminism and can be used for analysis of a wide range of discrete event phenomena.

Introduction

Traditionally, control theory has dealt with the dynamic behavior of processes whose variables are numerical and whose evolution can be modeled by differential or difference equations. With the widening use of computers as essential components of systems, increasingly complex systems have emerged that can no longer be adequately described by conventional models. Indeed, in an increasing number of processes states may have not just numerical values, but symbolic or logical values as well. State changes may then occur in response to the occurrence of discrete events that take place at discrete times, frequently asynchronously and nondeterministically. The control of such systems is of great practical importance and theoretical interest, and poses a wide range of new and intriguing intellectual challenges.

The simplest processes that exhibit such discrete behavior are discrete event processes. These are processes whose behavior can be modeled entirely within a state-event framework, that is, processes whose states are discrete and state changes take place only in response to events that occur at discrete and irregular intervals. Some of the more common and familiar examples of such processes are computer operating systems, manufacturing systems, communication networks, traffic systems, resource (such as power or water) management systems, and computer-based supervisory control systems of complex plants.

A state transition and its associated event constitutes the basic fragment of a discrete event process. (Finite) state machines and their associated state transition diagrams are the simplest formal mechanism for collecting much fragments into a whole. State machine models are conceptually appealing because of their inherent simplicity and the fact that they can be described adequately by finite automata and the theory of regular languages.

Recently, Ramadge and Wonham [45, 49, 50] initiated a pioneering effort of developing a control theory of discrete event processes within the framework of state machines and formal languages. In their framework all events are spontaneous and process-generated. Some of the events, called controllable events, possess a disablement mechanism accessible to the environment, and the control problem is to suitably interact with the process, by disabling of controllable events, so as to confine its behavior to within specified legal bounds. The mechanism examined in the work of Ramadge and Wonham for such interaction is called feedback control and consists of certain mappings between the process under consideration and a suitably formulated supervisor. Process behavior is modeled by its language, i.e., the set of event-strings that the process can generate. Various control-theoretic questions such as controllability [45, 49], observability [28, 42, 37, 12], decentralized and hierarchical control [29, 51, 38] and stabilization [8, 36], as well as such questions as computational complexity [43, 44] and others were studied in the Ramadge-Wonham framework. Their research had a profound impact on the control systems research community and generated a growing interest in control of discrete event processes as evidenced by the expanding number of research contributions to this subject (e.g., [11, 12, 22, 23, 25, 26, 48, 7, 8]).

In spite of their inherent simplicity and corresponding attractiveness, state machines have a weakness as models of complex processes because they suffer from an exponential explosion in the number of their states. To be effective and useful, it is desirable that a state/event modeling formalism have the capability to somehow relax the requirement that all states as well as all event sequences be present explicitly in the model at all times. Thus, one would like to be able to suppress in such a model all aspects of its description that are irrelevant in a particular context. This can be achieved by event-internalization, or partial observation, which leads to nondeterminism in process behavior (in the automata-theory sense) and to inadequacy of formal languages as models of behavior. A further aspect of effective modeling is the ability to construct a process description from individual components, thus introducing as an integral element of the modeling framework modularity and hierarchy. Also, to obtain an effective description tool, it is important to have the capability of describing behavior recursively. Finally, since all modules of the process must interact and correctly synchronize when operating in parallel, a suitable mechanism for communication and interaction between the various process components must be formulated, one that includes a suitable formalism for control of discrete event processes.

The importance of developing a framework for modeling, specification, verification and synthesis of discrete event processes, with particular emphasis on computer operating systems, data-base management, concurrent programs, and distributed computing, has been recognized in the computer science community for well over a decade, and a diverse and extensive literature has developed on this subject. Notable among the various approaches that have been developed are Petri-Net Theory [39], linear-time and branching-time temporal logics [13, 31, 40, 24], and, of particular interest in the context of the present paper, a number of (closely related) algebras of concurrent processes that were inspired by Hoare's Communicating Sequential Processes (CSP) [20] and Milner's Calculus of Communicating Systems (CCS) [33], and became widely known as the theory of concurrency [9, 10, 18, 32, 34, 4, 5]. (The reader is referred to the two recent volumes [2] and [3] for a broad over-

Presented at the 1989 IEEE Conference on Decision and Control, December 13-15, 1989. Michael Heymann is with the Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel. This article was written while he was on leave as a NRC-Senior Research Associate at NASA-Ames Research Center, Moffett Field, CA 94035.

0272-1708/90/0600-0103 $01.00 © 1990 IEEE

view of the current literature.)

In spirit and in general philosophy, the theory of concurrency is well suited for modeling, analysis, and synthesis of discrete event control systems. A central theme in that theory is the description of the interaction between discrete event processes and their environment. Such interaction is modeled by parallel composition with a specified degree of event synchronization. While various formalisms of parallel composition have been defined and investigated in the literature, they all rely on some framework of strict synchronization. That is, specific events of distinct processes must either strictly synchronize or be completely independent and interleave. These formalisms are inherently inadequate for modeling the interaction of discrete event processes in which sponteneity of events is an essential behavioral feature.

This article is a tutorial introduction to the theory of concurrency and to the associated process-algebra, and the suitability of such a methodology for modeling and control of discrete event processes is examined. It is shown that the existing formalisms of synchronization are inadequate for modeling the interaction of (dynamic) discrete event processes with the environment. Accordingly, a new parallel composition operator, called prioritized synchronous composition, that can model a wide range of interactions among discrete event processes, is introduced. Aspects of the corresponding process-algebra are examined. Finally, some comments are made about aspects of controllability within the framework of the new methodology. A more detailed and formal account of the new algebra of discrete event processes can be found in [19].

Process Components and Operators

Following standard notation, let Σ be a finite set of event labels and let Σ^* denote the set of all finite strings of elements of Σ, including the empty string ϵ. A discrete event process (or DEP) P with events in Σ is then a device that undergoes state transitions in response to events in Σ. A local description of P can be given in terms of individual state transitions as follows. If p and p' are states of P, and σ is an event in Σ, then we shall use the notation shown to express the possibility for process P to undergo transition from state p to state p' in response to the event σ:

$$P{:}p \overset{\sigma}{\rightarrow} p'.$$

Similarly, we shall use the following notation to express the fact that when the process P is at state p, no state transition is possible in response to the event σ.

$$P{:}p \overset{\sigma}{\rightarrow} \backslash.$$

At this stage we do not concern ourselves with the mechanism of event generation.

We shall also find it convenient to refer formally to P as the global process structure consisting of its complete state transition tree, or graph, and its designated initial state p_0. This allows us to introduce the important prefix operator (or prefix construction) by defining the process Q as

$$Q := \sigma \rightarrow P. \tag{1}$$

That is, Q is the process that starts at its initial state (say q_0) and, in response to event σ, undergoes transition to P. For example, if $P = \Delta$, the deadlock-process (that cannot undergo any state transitions), then (1) means that Q is the process that can execute (or respond to) event σ and then deadlock.

Another important process-operator is the controlled alternative operator $+$, which is defined as follows. Let $Q_1 = \sigma_1 \rightarrow P_1$ and $Q_2 = \sigma_2 \rightarrow P_2$ and assume that $\sigma_1 \neq \sigma_2$. Then the following is the process that in its initial state can either respond to σ_1 and undergo transition to P_1, or respond to σ_2 and undergo transition to P_2. The choice of the initial event is at the disposal of the environment:

$$Q := Q_1 + Q_2 = (\sigma_1 \rightarrow P_1) + (\sigma_2 \rightarrow P_2). \tag{2}$$

An important element of nondeterministic process behavior is provided by the uncontrolled alternative operator $\oplus$. A simple illustration of this operator is provided by the following situation. If $Q_1 = \sigma \rightarrow P_1$ and $Q_2 = \sigma \rightarrow P_2$, then

$$Q := Q_1 \oplus Q_2 = (\sigma \rightarrow P_1) \oplus (\sigma \rightarrow P_2)$$
$$:= (\sigma \rightarrow P_1 \oplus P_2). \tag{3}$$

This is the process that, in response to the initial event σ, undergoes transition either to P_1 or to P_2, but the choice is completely nondeterministic.

Next we introduce the event-internalization operator. Let P be a process with event set Σ. By the internalization of an event $\sigma \in \Sigma$, we refer to the removal of all occurrences of the event σ from external view so that all state transitions associated with σ become silent, or unobserved, (denoted by ϵ). We denote the resultant process by $P\backslash_\sigma$.

Example 1 Consider the process $P\backslash_a$ in Fig. 1 where P is given by

$$P = (a \rightarrow b \rightarrow \Delta) + (c \rightarrow \Delta). \tag{4}$$

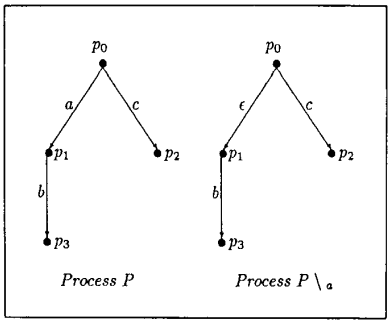


Fig. 1.

Notice that the process $P\backslash_a$ possesses nondeterministic behavior in that the internalized event can occur at any time without the explicit knowledge of the observer. Thus we may not know whether the process is at state p_0 or at p_1.

Definition 1 A DEP P is called deterministic if it has no silent transitions and for every state p of P and every event $\sigma \in \Sigma$ there is at most one state p' such that $P{:}p \overset{\sigma}{\rightarrow} p'$.

An interesting and important question is how the process $P\backslash_a$ of Example 1 differs from the deterministic process $\hat{P} := (b \rightarrow \Delta) + (c \rightarrow \Delta)$ which generates the same event-strings (or traces). We shall return to this and related questions in some more detail later but in the meantime we shall only note that the following identity holds true:

$$((a \rightarrow b \rightarrow \Delta) + (c \rightarrow \Delta))\backslash_a$$
$$= ((b \rightarrow \Delta) + (c \rightarrow \Delta)) \oplus (b \rightarrow \Delta). \tag{5}$$

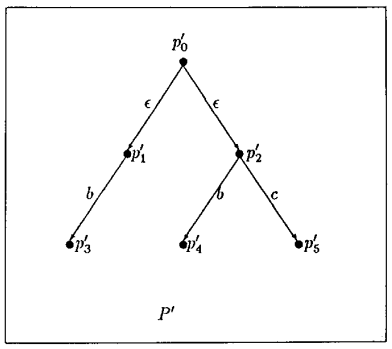


Fig. 2.

Equation (5) means that the process $P\backslash_a$ can be identified in some sense with the process P' whose state transition graph is depicted in Fig. 2. In the process P' there is an initial nondeterministic (unobserved) transition

from p_0' to either p_1' or p_2' after which it becomes deterministic. The identification of two distinct processes like that in (5) is at the heart of a process-algebra and we shall return to this issue later.

We shall conclude this section with a brief discussion of recursive equations for process description. Consider an equation of the following form, where f is a function of (or an operator on) P:

$$P = f(P). \tag{6}$$

Such an equation is a fixed-point equation, which, under suitable conditions (see e.g., [30, 20]), has a recursive solution (for P). Under appropriate restrictions this solution is even unique. Fixed point equations are a convenient way for process formulation. A simple illustration is given by the following example.

Example 2 The (recursive) solution to the following fixed point equation is the process whose transition graph is given in Fig. 3.

$$P = (a \rightarrow b \rightarrow P) + (c \rightarrow P). \tag{7}$$

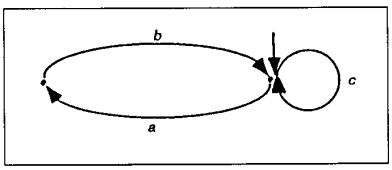


Fig. 3.

Formalisms of Concurrency

Processes interact with their environment through communication. That is, they operate in parallel with a specified degree of event synchronization. Thus we speak of parallel composition or concurrency of DEPs. Various formalisms of concurrency have been studied in the computer science literature. The simplest form of concurrency is parallel composition without synchronization, which is modeled by the interleaving behavior of the component processes. We shall denote this parallel composition by $(\cdot \parallel_{\varnothing} \cdot)$, where the subscript $\varnothing (\subseteq \Sigma)$ denotes the fact that the set of synchronized events is empty. Thus, if P and Q are DEPs, then the DEP $P \parallel_{\varnothing} Q$ is the process obtained from operating P and Q in parallel completely independently. The only assumption that is generally made about this parallel operation is that events of P and Q never coincide in time. (An exception to this assumption can be found, e.g., in [32].) Using our notational convention, we can thus define the operation of parallel composition without synchronization, formally, by

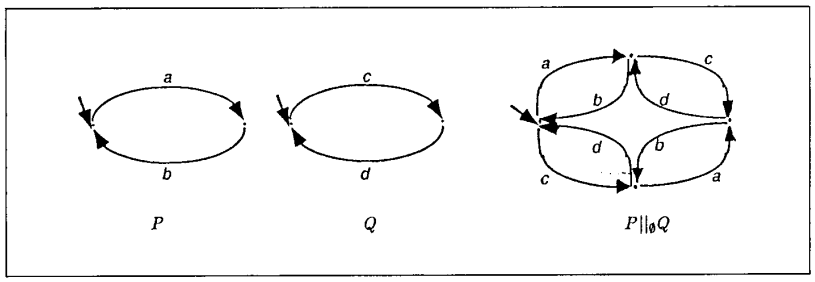


Fig. 4.

$$P:p \xrightarrow{\sigma} p' \Rightarrow P \parallel_{\varnothing} Q:(p, q) \xrightarrow{\sigma} (p', q) \tag{8}$$

$$Q:q \xrightarrow{\mu} q' \Rightarrow P \parallel_{\varnothing} Q:(p, q) \xrightarrow{\mu} (p, q'). \tag{9}$$

As an example of process interleaving consider the simple processes P and Q in Fig. 4. At the other extreme of the range of possible synchronizations, is the parallel composition with full synchronization, denoted $(\cdot \parallel_{\Sigma} \cdot)$. In this case the synchronization of events is complete in that all events in the event set Σ must be synchronized. Thus, if P and Q are Σ-processes, i.e., processes over the event set Σ, then an event in $P \parallel_{\Sigma} Q$ can take place if and only if it can take place simultaneously (and synchronously) in both processes. If one of the processes cannot participate in an event initiated by the other, the event will not take place in either process. If no common events exist at a given time, the composite process $P \parallel_{\Sigma} Q$ deadlocks. Parallel composition with full synchronization is sometimes also called parallel composition by intersection because the trace set of $P \parallel_{\Sigma} Q$ is easily seen to be precisely the intersection of the trace sets of P and of Q. The operator can be defined formally by expressions similar in form to those in equations (8) and (9), but take into account event synchronization and the possibility of deadlock.

An example of parallel composition with full synchronization is given in Fig. 5. A generalization of the synchronization convention, that includes both parallel composition by interleaving and parallel composition with full synchronization as special cases, is given by the operator $P \parallel_A Q$, where $A \subseteq \Sigma$ is an arbitrary subset called the synchronization set. Informally, this is the process obtained when P and Q run independently in parallel, except that they must fully synchronize their events in A. This operator is defined formally by

$$P:p \xrightarrow{\sigma} p' \;\&\; Q:q \xrightarrow{\sigma} q' \Rightarrow P \parallel_A Q:(p, q) \xrightarrow{\sigma} (p', q') \tag{10}$$

$$P:p \xrightarrow{\sigma} p' \;\&\; Q:q \xrightarrow{\sigma} \backslash \Rightarrow P \parallel_A Q:(p, q) \xrightarrow{\sigma} \begin{cases} (p', q), & \text{if } \sigma \notin A \\ \backslash, & \text{otherwise} \end{cases} \tag{11}$$

$$Q:q \xrightarrow{\sigma} q' \;\&\; P:p \xrightarrow{\sigma} \backslash \Rightarrow P \parallel_A Q:(p, q) \xrightarrow{\sigma} \begin{cases} (p, q'), & \text{if } \sigma \notin A \\ \backslash, & \text{otherwise.} \end{cases} \tag{12}$$

In the above operator the case $A = \varnothing$ corresponds to parallel composition by interleaving and $A = \Sigma$ corresponds to parallel composition by intersection. It is noteworthy that equations such as the above are not being

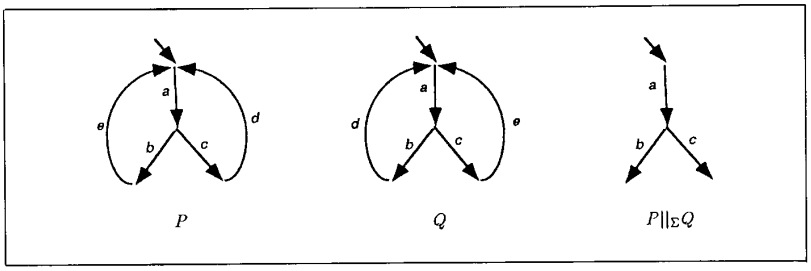


Fig. 5.

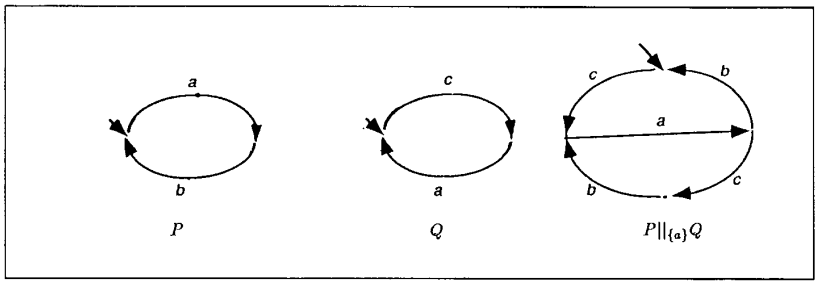


Fig. 6.

presented unnecessarily. Indeed, one of the main points of the present paper is that DEPs can be described by such (and other) equations and that these equations can be manipulated, simplified and solved using the process algebra.

An example of parallel composition with partial synchronization is given in Fig. 6.

Process Models and Language-Congruence

In the present section we discuss certain questions regarding DEP modeling. The main purpose of a mathematical model of a DEP is to describe its behavior. We must, therefore, require of a model to capture enough detail about the DEP's structure, so as to ensure that its behavior is fully exhibited in all circumstances. A model can be regarded as efficient if it captures just enough detail (for our purposes) but no more detail than necessary. Thus, an efficient model must not distinguish between DEPs that, in a given framework, exhibit identical behavior. Next we proceed to make these ideas somewhat more precise.

As we have already seen earlier, in a DEP modeling environment, DEPs are given by algebraic expressions whose arguments are also DEPs. The range of such algebraic expressions is determined by the range of algebraic operators that are defined in the given framework. Let us denote such a framework by $\mathcal{A} = \mathcal{A}(O_1, \cdots, O_k)$, where $O_1, \cdots, O_k$ are the operators under consideration. In the context of the framework exhibited thus far, the operators include the prefix operator, the alternative operators, the internalization operator, the recursion and, most importantly, the operator of strict concurrency.

By the behavior of a process P, we refer to the language $\mathfrak{L}(P) \in \Sigma^*$, consisting of all event strings, or traces, that P generates. Let $\mathfrak{M}$ denote a modeling framework for DEPs, so that $\mathfrak{M}(P)$ denotes a model for a DEP P. Then $\mathfrak{M}$ induces an equivalence relation, denoted $\mathcal{E}_{\mathfrak{M}}$, on the class of all DEPs under consideration. Specifically, we then say that DEPs P and Q are equivalent, denoted $P\mathcal{E}_{\mathfrak{M}}Q$, whenever $\mathfrak{M}(P) = \mathfrak{M}(Q)$. Clearly then for the modeling framework $\mathfrak{M}$ to be adequate, we must require that if $\mathfrak{M}(P) = \mathfrak{M}(Q)$, then P and Q must exhibit the same behavior under all circumstances. This leads us to the following.

Definition 2 An equivalence relation $\mathcal{E}_{\mathfrak{M}}$ on the class of DEPs is called a language-congruence (with respect to $\mathfrak{A}$) if for every $f \in \mathfrak{A}$ and any two DEPs P and Q:

$$P\mathcal{E}_{\mathfrak{M}}Q \Rightarrow \mathfrak{L}(f(P)) = \mathfrak{L}(f(Q)). \quad (13)$$

In the above definition $f(P)$ denotes an expression with P as an argument. (We do not preclude the possibility that f is an expression in more than a single argument, in which case our notation implies that the other arguments are held fixed.)

Thus, in terms of the above definition, an adequate modeling framework must induce a language-congruence. But this, of course, does not guarantee that the modeling framework is efficient. Let $\mathcal{C}$ denote the set of all equivalence relations on the class of DEPs over a fixed event-alphabet Σ. If $\mathcal{E}_1, \mathcal{E}_2 \in \mathcal{C}$ are two equivalence relations, we say that $\mathcal{E}_1$ is coarser than $\mathcal{E}_2$, denoted $\mathcal{E}_1 \geq \mathcal{E}_2$, if for any pair of DEPs P and Q,

$$P\mathcal{E}_2Q \Rightarrow P\mathcal{E}_1Q.$$

It is easily seen that $\geq$ constitutes a complete partial order on DEPs [30]. We now have the following.

Definition 3 A DEP modeling framework is called efficient if it induces the coarsest language-congruence with respect to $\mathfrak{A}$.

Thus, a modeling framework is efficient if it includes in the model of a DEP the least amount of detail necessary to distinguish DEPs that differ in behavior, but identifies all processes that cannot be distinguished behaviorally. It is important to realize that the detail needed in the model is crucially dependent on the operators that are included in $\mathfrak{A}$. As their expressiveness increases, the complexity of the models must, in general, increase as well.

Definition 4 A framework $\mathfrak{A}$ is called deterministically closed if for each $f \in \mathfrak{A}$, $f(P)$ is deterministic whenever P is deterministic.

It can be shown that (see, e.g., [35]), if $\mathfrak{A}$ is deterministically closed, then $\mathfrak{L}(P)$ is an adequate model for P. That is, $\mathfrak{L}$ itself constitutes a language congruence. Obviously $\mathfrak{L}$ is then the coarsest language congruence. The reader can convince himself without too much difficulty that $\mathfrak{A}_l = \mathfrak{A}(\sigma \rightarrow \cdot, +, \cdot\|_A\cdot,$ recursion) is deterministically closed. Thus, the behavior of deterministic processes that interact only through strict synchronization, can be adequately modeled by their languages. (This fact has been of key importance in the interesting work of Smedinga [46] on control of discrete events.)

We turn now to the case $\mathfrak{A}_f = \mathfrak{A}(\sigma \rightarrow \cdot, +, \oplus, (\cdot)\backslash_\sigma, \cdot\|_A\cdot,$ recursion). That is, $\mathfrak{A}_f$ includes also the operators of uncontrolled alternative and event internalization. Nondeterminism is now included in our framework.

It is of interest, at this stage, to return to the question raised in Example 1 of comparing the processes $\hat{P} = (b \rightarrow \Delta) + (c \rightarrow \Delta)$ and $P\backslash_a$ where $P = (a \rightarrow b \rightarrow \Delta) + (c \rightarrow \Delta)$, both of which generate the same languages. To this end, let us consider the following example that shows that processes $\hat{P}$ and $P\backslash_a$ are not language congruent.

Example 3 Consider the process $R := \hat{P}\|_\Sigma Q$, where $Q = (c \rightarrow \Delta)$. Using the definition of parallel composition with full synchronization as given by (10)–(12) with $A = \Sigma$, we obtain

$$R = (c \rightarrow \Delta).$$

Next, consider the process $R' := P\backslash_a\|_\Sigma Q$. While in this simple example the computation of R' can be performed directly without difficulty, we shall take the opportunity to demonstrate the use of process-algebra in computational simplification. First we shall use (5) to obtain

$$R' = (((b \rightarrow \Delta) + (c \rightarrow \Delta)) \oplus (b \rightarrow \Delta))\|_\Sigma(c \rightarrow \Delta). \quad (14)$$

Next we use the following identity (see, e.g., [10]):

$$(P \oplus Q)\|_A R = (P\|_A R) \oplus (Q\|_A R)$$

which together with (14) gives the following where the last equality is obtained with the aid of (10)–(12) (with $A = \Sigma$):

$$R' = ((b \to \Delta) + (c \to \Delta)\|_\Sigma(c \to \Delta)) \oplus ((b \to \Delta)\|_\Sigma(c \to \Delta)) = (c \to \Delta) \oplus \Delta.$$

Comparing R with R', we see that R' can deadlock initially, while R cannot. Indeed, the choice of whether R' will initially deadlock or not, is completely nondeterministic. This nondeterminism can best be understood upon noting that $P\backslash_a$ can undergo a silent transition from p_0 to p_1 (see Fig. 1), and there is no observable mechanism to guarantee that the event c be offered by Q prior to such transition.

The above example illustrates the fact that the language model cannot adequately express the possibility of deadlock. This fact motivated the introduction by [10] (see also [9, 20, 34, 35]) of the more sophisticated failures-model. This model, which is obviously more detailed than the language model, represents a process by its failures set $\mathfrak{F} = \{(s, X)\}$, where a failure (s, X) consists of a trace s, i.e., a string of events that the process can execute, and a refusal set X that consists of the events that the process can reject (or refuse) after the execution of s. We shall not elaborate here on the failures model except for giving a simple illustrative example.

Example 4 The failures set of the process $P\backslash_a$ of Example 1 is given by

$$\mathfrak{F}(P\backslash_a) = \{(\epsilon, \varnothing), (\epsilon, \{c\}), (b, \{b, c\}), (c, \{b, c\})\}.$$

We give here only the failures with maximal refusals.

Process Algebra

By process algebra we refer to a set of algebraic identities between process expressions. Such an algebra can then be used to manipulate, combine and simplify process expressions and perform a variety of computations with processes symbolically rather than explicitly. We have already encountered in the foregoing several algebraic process identities, and a simple example of their use in computational simplification was seen in Example 3. The chief utility of a behavioral (or semantic) modeling framework of processes is in establishing the algebraic identities. It is for this reason that we must guarantee that the modeling framework constitutes a behavioral (in our case, a language) congruence. The derivation of these identities are beyond the scope of the present paper

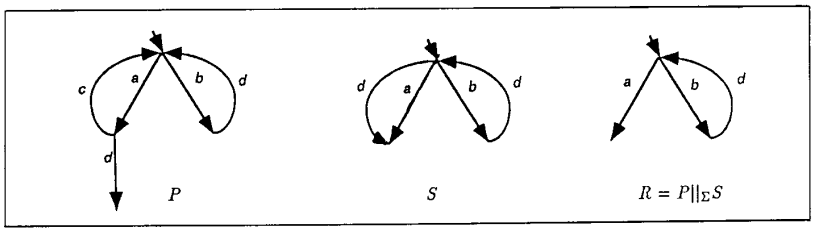


Fig. 7.

but for illustrational purposes we give below some representative examples of the type of algebraic identities that are valid for the failures model with respect to $\mathfrak{A}_f$ (see e.g., [35] for details):

$$(P + Q) + R = P + (Q + R) \quad (15)$$

$$(P + Q) \oplus R = (P \oplus R) + (Q \oplus R) \quad (16)$$

$$P\|_A(Q \oplus R) = (P\|_A Q) \oplus (P\|_A R) \quad (17)$$

$$((a \to P) + Q)\backslash_a = P\backslash_a \oplus (P + Q)\backslash_a. \quad (18)$$

The Ramadge-Wonham Discrete-Event Control Formalism

In their pioneering work on the control of DEPs, Ramadge and Wonham (RW) [45, 49, 50] introduced the following formalism. A DEP is modeled as a deterministic state-machine or automaton, called generator, which is given by a 4-tuple

$$G = (\Sigma, Q, \delta, q_0). \quad (19)$$

Here Q is a set of states, Σ is a set of events, $\delta: \Sigma \times Q \to Q$ is a partial function called the transition function, and q_0 is the initial state. The statement that δ is a partial function means that it need not be defined for all pairs $(\sigma, q) \in \Sigma \times Q$. (Actually, Ramadge and Wonham have a somewhat more general setting where a DEP is a 5-tuple that includes also marker states, but these are inessential to the present exposition.)

Control is introduced as follows. It is assumed that all events occur in the process spontaneously and asynchronously, but some of the events have a mechanism for their disablement at any time. Thus the event set Σ is partitioned into two disjoint subsets

$$\Sigma = \Sigma_u \cup \Sigma_c. \quad (20)$$

Here Σ_c is the subset of events that can be disabled, called controllable events, and Σ_u is the subset of events whose occurrence cannot be disabled, called uncontrollable. A control input for G is now defined as a subset $\Gamma \subseteq \Sigma_c$ of events that are disabled at any instant of time. Control of a DEP consists of switching the disablement set Γ as the process progresses in its run. With this event-set partition and associated disablement mechanism the DEP is called a controlled DEP, or CDEP.

The control execution is performed by a supervisor which can abstractly be thought of as a map

$$h: \mathfrak{L}(G) \to \Gamma. \quad (21)$$

Concretely, this means that after every event that takes place in the process, a new event set is supplied to the process for disablement. Thus when the CDEP is supervised by a supervisor h, the generator G must be modified by redefining the transition map δ as $\hat{\delta}$, where

$$\hat{\delta}(\sigma, q) := \begin{cases} \delta(\sigma, q), & \text{if } \sigma \in \Gamma \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Let $\mathfrak{L}_c(G)$ denote the language generated by G under control, i.e., in closed loop. Then it is clear that the domain of the map h can be restricted to $\mathfrak{L}_c(G)$. In practice, it is convenient to use a state machine realization for $\mathfrak{L}_c(G)$. Thus one defines $S = (\Sigma, X, \xi, x_0)$ as the automaton realizing $\mathfrak{L}_c(G)$, and the map h is replaced by a feedback map $\phi: X \to \Gamma$ such that for $s \in \mathfrak{L}_c(G)$

$$\phi(\xi(s, x_0)) = h(s). \quad (22)$$

Here $\xi(s, x_0)$ is the standard extension of the transition map to strings [21].

Strict Concurrency and Discrete Event Control

A key element in the Ramadge-Wonham control problem formulation is the introduction of what may be thought of as discrete-event dynamics, where by dynamics we refer to the presence of spontaneity, that is, the existence of events whose occurrence cannot be influenced by the environment.

Let us next examine the possibility of modeling the control of discrete event processes using the formalism of strict concurrency as described in Section III. To this end consider first the simple control problem described in Fig. 7.

Here all events of the process P are controllable, that is $\Sigma_c = \Sigma = \{a, b, c, d\}$ and $\Sigma_u = \varnothing$. The process S is to be thought of as the supervisor for P, with supervision achieved through concurrency with full synchronization. Specifically, when P is at state p_i and S is at state s_j, then the possibility of occurrence of an event, say a, in $R = P\|_\Sigma S$ at state $r_k = (p_i, s_j)$, means that the event is enabled by S and possible (subject to enablement) in P. An event in R is, thus, interpreted as enabled by S and occurring in P, and the participation of both processes in an event is essential for its occurrence. Thus, when all events are controllable, we can model control by strict concurrency with full event synchronization.

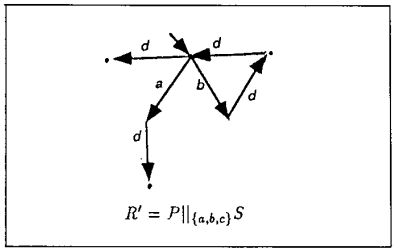


Fig. 8.

As it turns out, this is not quite as straightforward when we introduce dynamics, or uncontrollable events. Let us first try to clarify the synchronization status of the various events. Clearly, the controllable events must belong to the synchronization set as before, because it takes the supervisor to enable an event and the process to execute it. But what about the uncontrollable events? If an uncontrollable event is possible both in P and in S (at their respective states), its occurrence in the concurrent process must be given the physical interpretation as having been executed in S in response to its (spontaneous) occurrence in P. If it is possible only in P, but not in S, it will still occur in P, and hence in the concurrent process, because of its uncontrollability. But if an uncontrollable event is possible only in S, it will not occur because S cannot initiate the event.

Let us reconsider the above example, but this time assume that the event d is uncontrollable. Thus let $\Sigma_c' = \{a, b, c\}$ and $\Sigma_u' = \{d\}$. Let us reexamine the process $R = P\|_\Sigma S$ of Fig. 7. The event d appears in R after the occurrence of b but not after a. This is physically incorrect because once a occurs, the event d cannot be blocked by its absence in the supervisor. If, on the other hand, we remove the uncontrollable events from the synchronization set, and try to model controlled behavior by $R' = P\|_{\Sigma_c} S$, we would obtain the process R' as in Fig. 8, which is unsatisfactory because it permits the occurrence of the event d without participation of P, which is impossible in the physical process.

Thus, it is clear that concurrency with strict synchronization cannot be used as a satisfactory framework for modeling the interaction of dynamical discrete-event processes with their environment. More specifically, strict concurrency is an inadequate formalism for modeling control of discrete event processes within the Ramadge-Wonham framework (unless we impose special restrictive conditions on the supervisor such as the condition of supervisor completeness [45]).

Concurrency by Prioritized Synchronization

In the present section we introduce a new concurrency operator, called prioritized synchronous composition that is suitable for modeling a wide range of practical control formalisms.

Let the event set be denoted by Σ and consider two processes P and Q with events in Σ. With each process we associate a subset of special events, called its set of priority (or blocking) events. These are events in whose execution the given process must participate; otherwise they cannot take place. Thus, let $A, B \subseteq \Sigma$ be the priority sets of P and Q, respectively, and define the prioritized synchronous composition of P and Q, denoted $P_A\|_B Q$, as follows:

$$P{:}p \xrightarrow{\sigma} p' \;\&\; Q{:}q \xrightarrow{\sigma} q' \Rightarrow P_A\|_B Q{:}(p, q) \xrightarrow{\sigma} (p', q') \tag{23}$$

$$P{:}p \xrightarrow{\sigma} p' \;\&\; Q{:}q \xrightarrow{\sigma} \backslash \Rightarrow P_A\|_B Q{:}(p, q) \xrightarrow{\sigma} \begin{cases} (p', q), & \text{if } \sigma \notin B \\ \backslash, & \text{if } \sigma \in B \end{cases} \tag{24}$$

$$Q{:}q \xrightarrow{\sigma} q' \;\&\; P{:}p \xrightarrow{\sigma} \backslash \Rightarrow P_A\|_B Q{:}(p, q) \xrightarrow{\sigma} \begin{cases} (p, q'), & \text{if } \sigma \notin A \\ \backslash, & \text{if } \sigma \in A. \end{cases} \tag{25}$$

Expression (23) states that if, at their respective states, both processes P and Q can execute a given event σ, then it will be executed concurrently (i.e., in synchronization) in both processes. Both processes will then undergo simultaneously their respective state transitions. Notice that, when both processes can execute an event concurrently, the mathematical model does not distinguish which process initiates the event. Indeed, as we shall see shortly, this is a matter for the physical interpretation. Expressions (24) and (25) define the concurrency operator in case that an event is possible in (initiated by) one of the processes but is not possible in the other: In this case, the initiating process will execute the event without participation of the other, unless the event is in the priority set of the latter. In this case the execution of the event is blocked.

It is not difficult to see that the prioritized synchronous composition operator partitions the event set Σ into four distinct (and disjoint) subsets:

1) The set $A \cap B$ of strict-synchronization events. These events are either executed by both processes concurrently or by none.
2) The set $\Sigma - A \cup B$ of broadcast synchronization events. Each process can offer these events for execution and the other process will participate in their execution synchronously if it can. But if it cannot (i.e., if the event is impossible in its current state), the initiating process will execute the event by itself.
3) The set $A - A \cap B$ of priority events of process P. The execution of these events will take place if and only if the process P participates. The participation of the process Q in these events will take place whenever possible, i.e., whenever Q can in its respective state. But lack of participation by Q cannot block execution by P.
4) The set $B - A \cap B$ of priority events of process Q. (Similar to 3) above.)

To illustrate the prioritized synchronous composition, consider the following simple example:

Example 5 Let $\Sigma = \{a, b, c\}$ and consider the parallel composition of processes P and Q as described in Fig. 9, where $A = \{a, c\}$ and $B = \{a, b\}$. Observe that the event a occurs only when both processes P and Q participate in the execution taking R from state $r_4 = (p_1, q_1)$ to $r_3 = (p_2, q_2)$. However the event c occurs in R either by participation of both processes, for example in transition from $r_1 = (p_0, q_1)$ to $r_2 = (p_1, q_0)$, or by execution of P alone, in case the event is not available in Q, as for example in transition from $r_0 = (p_0, q_0)$ to $r_2 = (p_1, q_0)$. Notice also that the transition of process P from p_2 to p_0 never takes place when it runs concurrently with Q because the event b is in the priority set of Q, but Q is never at state q_0 when P is at p_2. The important property that is demonstrated above and that distinguishes prioritized synchronous composition from

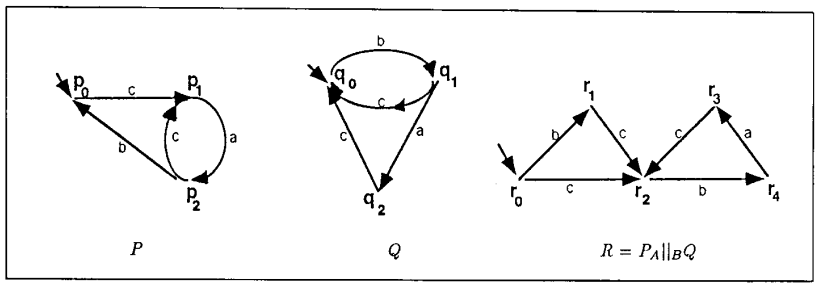


Fig. 9.

other concurrency operators, is the fact that the behavior of the concurrent process with respect to a given event, depends not just on the event, but also on the context and event availability.

We turn now to an examination of how our prioritized synchronous composition can model control of DEPs. First we remark that DEPs frequently have other mechanisms for interaction with the environment than the one investigated by Ramadge and Wonham. For example, the process may possess also driven events that must be forced or triggered by the environment in order to take place. Driven events are then distinguished from controllable events in that when they are possible in the process and triggered by the controller, they are guaranteed to take place immediately and instantaneously.

We shall let P denote the process under consideration, and let S denote the supervisor. The controlled, or closed-loop, process is then given by

$$R = (S/P) := P_A\|_B S. \tag{26}$$

The priority sets A and B are of course suitably chosen so as to correctly model the physical environment. The event set Σ will be partitioned into three disjoint subsets

$$\Sigma = \Sigma_u \cup \Sigma_c \cup \Sigma_d. \tag{27}$$

The subset Σ_u is the set of uncontrollable events, Σ_c is the set of controllable events, and Σ_d is the set of driven events. Next we consider an example in the Ramadge-Wonham framework, that is, $\Sigma_d = \varnothing$.

Example 6 Let the event set be $\Sigma = \{a, b, c, d, e\}$, and consider the simple processes P and S of Fig. 10.

Suppose P is the controlled process and let the subset of controllable events be $\Sigma_c = \{a, b, c\}$, and the subset of uncontrollable events be $\Sigma_u = \{d, e\}$. Thus, the events in Σ_c can be disabled while the events in Σ_u cannot. Let us now see how we can model control of process P by supervisor S through prioritized synchronous composition of P and S. Since all events are assumed to be spontaneous events of the process P, its priority set A must include them all, that is, $A = \Sigma$. The uncontrollable events cannot be influenced by the environment. Thus, the priority set B (of the supervisor S) must not include any uncontrollable event of P. This means that uncontrollable events of P cannot be blocked by the supervisor, but the supervisor may (if the designer so wishes) execute concurrently state transitions in response to their occurrence in P. The controllable events, however, will not occur in the process unless they are enabled by the supervisor. Thus, the controllable events must be in the supervisor's priority set B, and we have $B = \{a, b, c\}$. Notice that in this case $\Sigma - A \cup B = \varnothing$, $A \cap B = \Sigma_c$, $A - A \cap B = \Sigma_u$, and $B - A \cap B = \varnothing$.

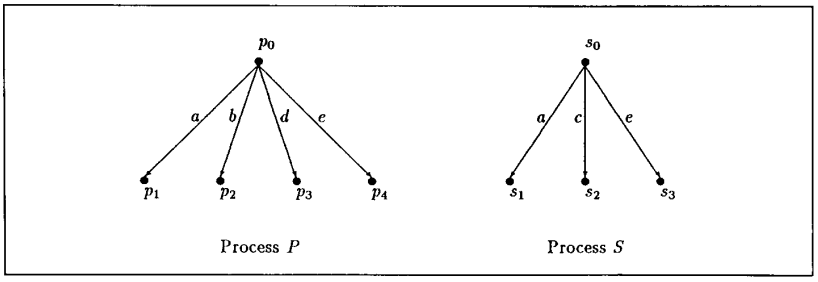


Fig. 10.

In our example the controlled process $R := P_A\|_B S = P_\Sigma\|_{\Sigma_c} S$ is then obtained as shown in Fig. 11 where $r_0 = (p_0, s_0)$, $r_1 = (p_1, s_1)$, $r_2 = (p_3, s_0)$, and $r_3 = (p_4, s_3)$. Notice that the controllable events a and c are both enabled by S while the event b is not. Hence in the controlled process R, the event a is present and will occur if it occurs in P. The events b and c do not appear in R; the first because it is not enabled by S and the second because it is not possible in P. The events d and e appear in R and occur whenever they occur in P regardless of their possibility (or lack of it) in S. Thus, if e happens in P, then S participates synchronously while if d happens in P, then S remains in its initial state s_0.

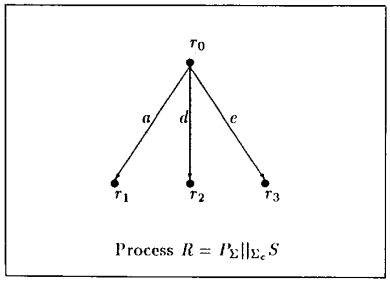


Fig. 11.

We conclude this section with some remarks on control with driven events. Let P and S be the process and supervisor with priority sets A and B, respectively. The set of driven events Σ_d must then be included in B in view of the fact that they will not take place unless triggered by the supervisor. However, they may or may not be included in A, depending on the specific control mechanism employed. Specifically, we shall say that driven events are synchronized in closed-loop, if they are included in the priority set A. In this case the supervisor S, that initiates the driven events, waits for an acknowledgement that the triggered event is actually possible (and executed) before it proceeds with further state transitions of its own. Driven events that are not included in A are said to be executed in open-loop. In open-loop mode, the forcing process does not wait for acknowledgement.

We can summarize the discussion of this Section with a formal classification of the events with respect to the priority sets A and B as follows. First, we have the requirement that

1. $A \supseteq \Sigma_u \cup \Sigma_c$,
2. $B = \Sigma_c \cup \Sigma_d$.

The subset $\Sigma_{dc} := \Sigma_d \cap A$ is then the set of closed-loop driven events and the set $\Sigma_{do} := \Sigma_d - \Sigma_{dc}$ is the set of open-loop driven events.

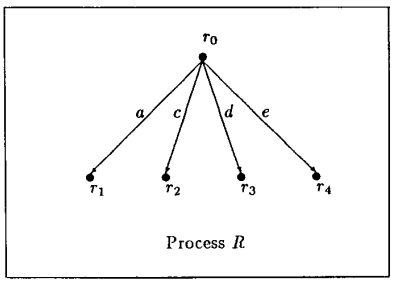


Fig. 12.

The Trajectory Model and Associated Algebra

As was stated earlier, we identify process behavior with the language that it generates. Thus, we must guarantee that we have a behavioral model for DEPs that constitutes a language-congruence with respect to an algebraic framework that includes prioritized synchronous composition. We shall concern ourselves with this question in the present section.

We have seen in the previous section how the prioritized synchronous composition operator $\cdot_A\|_B\cdot$ can be used to model a wide range of parallel composition formalisms and is, in particular, suitable for modeling dynamics and discrete-event control. We have also mentioned that the failures model captures adequately deadlock phenomena in nondeterministic behavior. It turns out, however, that in general, the failures model cannot adequately account for the range of possible interleavings that can occur in the framework of the operator $\cdot_A\|_B\cdot$ when nondeterminism is also present. This is illustrated in the following simple example.

Example 7 Consider the two processes

$$P = (a \rightarrow ((c \rightarrow \Delta) + (b \rightarrow \Delta))) \oplus (a \rightarrow b \rightarrow d \rightarrow \Delta)$$

$$P' = (a \rightarrow ((c \rightarrow \Delta) + (b \rightarrow d \rightarrow \Delta))) \oplus (a \rightarrow b \rightarrow \Delta).$$

It is easily seen that P and P' have the same failures set which is given by (listing again just the failures with maximal refusals):

$$\mathfrak{F} = \{(\epsilon, \{b, c, d\}), (a, \{a, d\}), (a, \{a, c, d\}), (ac, \{a, b, c, d\}), (ab, \{a, b, c\}), (ab, \{a, b, c, d\}), (abd, \{a, b, c, d\})\}.$$

To see that they do not behave the same way under prioritized synchronous composition, let them have priority set $A = \{a, b, d\}$ and run them in parallel with the process Q with priority set $B = \{a, b, c\}$, where

$$Q = a \rightarrow c \rightarrow b \rightarrow \Delta.$$

We obtain distinct results:

$$R = P_A\|_B Q = (a \rightarrow c \rightarrow \Delta) \oplus (a \rightarrow c \rightarrow b \rightarrow d \rightarrow \Delta),$$

$$R' = P'_A\|_B Q = (a \rightarrow c \rightarrow \Delta) \oplus (a \rightarrow c \rightarrow b \rightarrow \Delta).$$

In view of the above, it is clear that the failures model is not a language congruence with respect to $\mathfrak{A}_t = \mathfrak{A}(\sigma \rightarrow \cdot, +, \oplus, (\cdot)\backslash_\sigma, \cdot_A\|_B\cdot$, recursion).

We turn now to a brief discussion of a model, called the trajectory-model, that is a language-congruence with respect to $\mathfrak{A}_t$, and which has been examined in detail in [19] in the framework of a complete axiomatic theory. In the trajectory-model the process is specified by its set of trajectories. A process trajectory is a record of an "experiment" that describes an execution of a string of events, and records in addition to the executed events, also the events that the process can reject (or refuse) after each successful event. A typical trajectory is then an object of the form

$$(X_0, \sigma_1, X_1, \cdots, X_{k-1}, \sigma_k, X_k).$$

Here σ_i denotes the ith successful event, X_i denotes the set of events that can be refused after the ith executed event and X_0 denotes the set of events that can be refused initially. The following is an example of the trajectory set of a process (listing only the trajectories of maximal length with maximal refusal sets).

Example 8 The set of trajectories of maximal length with maximal refusals of the process P of Example 7 is given by

$$\mathfrak{I}(P) = \{(\{b, c, d\}, a, \{a, d\}, c, \{a, b, c, d\}), (\{b, c, d\}, a, \{a, d\}, b, \{a, b, c, d\}), (\{b, c, d\}, a, \{a, c, d\}, b, \{a, b, c,\}, d, \{a, b, c, d\})\}.$$

The set of trajectories of maximal length (with maximal refusals) of the process P' of Example 7 is given by

$$\mathfrak{I}(P') = \{(\{b, c, d\}, a, \{a, d\}, c, \{a, b, c, d\}), (\{b, c, d\}, a, \{a, c, d\}, b, \{a, b, c, d\}), (\{b, c, d\}, a, \{a, d\}, b, \{a, b, c,\}, d, \{a, b, c, d\})\}.$$

Notice that the trajectory sets for the processes P and P' (both of which have the same failures set) are distinct. This distinction accounts for their different behavior under parallel composition that was evidenced in Example 7.

It has been shown in [19] that a variety of useful algebraic identities similar in form to the examples given previously also hold for the trajectory model with respect to $\mathfrak{A}_t$. These are beyond the scope of the present paper.

Aspects of Controllability

We conclude this paper with some remarks about controllability in discrete event control viewed within the framework of concurrency.

A behavioral specification for a DEP is, typically, a statement about languages. If $\Sigma_s \subseteq \Sigma$ is some event subset, than a local specification consists of a pair of languages $\underline{\mathcal{K}_s}$, $\overline{\mathcal{K}_s} \subseteq \Sigma_s^*$ such that $\mathcal{L}(P\backslash_{\Sigma-\Sigma_s})$, the language of the process localized to Σ_s, satisfies the constraint

$$\underline{\mathcal{K}_s} \subseteq \mathcal{L}(P\backslash_{\Sigma-\Sigma_s}) \subseteq \overline{\mathcal{K}_s}. \qquad (28)$$

Sometimes $\underline{\mathcal{K}_s} = \varnothing$, and the specification consists of the upper-bound constraint only. If $\Sigma_s = \Sigma$, we call the corresponding specification global.

We shall assume that $\Sigma = \Sigma_u \cup \Sigma_c$, $A := \Sigma_u \cup \Sigma_c = \Sigma$ and $B = \Sigma_c$. If S is a supervisor for a process P, then

$$\mathcal{L}(S/P) = \mathcal{L}(P_\Sigma\|_{\Sigma_c} S) \subseteq \mathcal{L}(P). \qquad (29)$$

We can now introduce within our framework the concept of controllable languages.

Definition 5 Let $\mathcal{K}$ be a closed sublanguage of $\mathcal{L}(P)$. (A language is closed if it includes all its prefixes [21].) $\mathcal{K}$ is said to be controllable if and only if there exists a supervisor S such that

$$\mathcal{K} = \mathcal{L}(P_\Sigma\|_{\Sigma_c} S). \qquad (30)$$

A characterization of controllability (that was used as definition by Wonham and Ramadge in [45]) is the following easily proved theorem that essentially states that a sublanguage $\mathcal{K}$ of $\mathcal{L}(P)$ is controllable if every

event that is physically possible after a given event-string in P but is not possible after the same event-string in $\mathcal{K}$ is a controllable event (since it must be eliminated by disablement):

Theorem 1 *A closed sublanguage $\mathcal{K} \subseteq \mathcal{L}(P)$ is controllable if and only if for all strings $t\sigma \in \mathcal{L}(P)$ such that $t \in \mathcal{K}$ and $\sigma \in \Sigma$:*

$$t\sigma \notin \mathcal{K} \Rightarrow \sigma \in \Sigma_c. \tag{31}$$

A further result that can be proved using elementary process identities is the following.

Proposition 1 *Let P be a process. The class of controllable sublanguages of $\mathcal{L}(P)$ is closed under set union.*

The significance of the proposition is that it implies the existence of a unique supremal controllable sublanguage of $\mathcal{L}(P)$ (Theorem 2 below).

Consider now a process P, and let Δ, the deadlock process, serve as supervisor. The controlled process is then given as

$$(\Delta/P) = P_\Sigma \|_{\Sigma_c} \Delta \tag{32}$$

and it can be shown that if S is any supervisor for P, then

$$\mathcal{L}(\Delta/P) \subseteq \mathcal{L}(S/P). \tag{33}$$

Thus, the language $\mathcal{L}(\Delta/P)$ is the smallest controllable sublanguage of P. We denote this sublanguage by $\mathcal{U}_P$ and call it the uncontrollable or spontaneous language of P.

Theorem 2 *Let P be a process. Let $\mathcal{K} \subseteq \mathcal{L}(P)$ be a nonempty closed sublanguage. If $\mathcal{U}_P \subseteq \mathcal{K}$, then $\mathcal{K}$ contains a unique (nonempty) supremal controllable sublanguage.*

Concluding Remarks

In this article we surveyed aspects of the theory of concurrency and process-algebra and showed how the theory can be adapted to deal with issues of DEP modeling and control. The proposed framework is suitable for modeling a wide range of process-interaction formalisms and is capable of dealing adequately with aspects of nondeterminism.

We believe that the algebraic approch to DEP modeling and control proposed here can alleviate some of the computational difficulties caused by high dimensionality of practical DEPs.

References

[1] A. Benveniste, B. Le Goff, and P. Le Guernic, "Hybrid dynamical systems theory and the language 'SIGNAL'," IRISA/INRIA, Rennes Cedex, France, Int. Pub. 403, Apr. 1988.

[2] J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., "Current trends in concurrency," *Lect. Notes Comput. Sci.*, Vol. 224, 1986.

[3] J. W. de Bakker, W.-P. de Roever, and G. Rozenberg, Eds., "Linear time, branching time and partial order in logics and models for concurrency," *Lect. Notes Comput. Sci.*, Vol. 354, 1989.

[4] J. A. Bergstra and J. W. Klop, "Process theory based on bisimulation semantics," *Lect. Notes Comput. Sci.*, Vol. 354, pp. 50–122, 1989.

[5] J. A. Bergstra, J. W. Klop, and E.-R. Olderog, "Readies and failures in the algebra of communicating processes," *SIAM J. Comput.*, Vol. 17, pp. 1134–1177, 1988.

[6] G. Berry and I. Cosserat, "The ESTEREL synchronous programming language and its mathematical semantics," *Lect. Notes Comput. Sci.*, Vol. 197, 1985.

[7] Y. Brave and M. Heymann, "Formulation and control of a class of real-time discrete-event processes," Dept. of Electrical Engineering, Technion, Haifa, Israel, EE Pub. 714, Feb. 1989; see also *Proc. 27th IEEE Conf. Decision and Control*, pp. 1131–1132, Dec. 1988.

[8] Y. Brave and M. Heymann, "On stabilization of discrete-event processes," Dept. of Electrical Engineering, Technion, Haifa, Israel, EE Pub. 724, Apr. 1989; also to appear in *Int. J. Control*.

[9] S. D. Brooks and A. W. Roscoe, "An improved failures model for communicating processes," *Lect. Notes Comput. Sci.*, Vol. 197, pp. 281–305, 1985.

[10] S. D. Brooks, C. A. R. Hoare and A. W. Roscoe, "A theory for communicating sequential processes," *J. Assoc. Comput. Mach.*, Vol. 31, pp. 560–599, 1984.

[11] H. Cho and S. I. Marcus, "On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation," *Math. Control Signals Systems*, Vol. 2, pp. 47–69, 1989.

[12] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaya, "Supervisory control of discrete event processes with partial observations," *IEEE Trans. Autom. Control*, Vol. 33, pp. 249–260, 1988.

[13] E-A. Emerson and J. Srinivasan, "Branching time temporal logic," *Lect. Notes Comput. Sci.*, Vol. 354, pp. 123–172, 1989.

[14] C. H. Golaszewski and P. J. Ramadge, "Control of discrete event processes with forced events," in *Proc. 26th IEEE Conf. Decision and Control*, Los Angeles, CA, Dec. 1987, pp. 247–251.

[15] 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*, Dec. 1988, pp. 234–239.

[16] C. H. Golaszewski and P. J. Ramadge, "Discrete event processes with arbitrary controls," in *Advanced Computing Concepts and Techniques in Control Engineering*, M. J. Denham and A. J. Laub, Eds. New York: Springer-Verlag, 1988, pp. 459–469.

[17] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Prog.*, Vol. 8, pp. 231–274, 1987.

[18] M. Hennessy, *Algebraic Theory of Processes*. Cambridge, MA: M.I.T. Press, 1988.

[19] M. Heymann and G. Meyer, "An algebra of discrete event processes," NASA TM, 1989.

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

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

[22] K. Inan and P. Varaya, "Finitely recursive process models for discrete event systems," *IEEE Trans. Automatic Control*, Vol. 33, pp. 626–639, 1988.

[23] K. Inan and P. Varaya, "Algebras of discrete event models," *Proc. IEEE*, Vol. 77, pp. 24–38, 1989.

[24] F. Kroger, *Temporal Logic of Programs*. New York: Springer Verlag, 1987.

[25] S. Lafortune, "Modeling and analysis of transition execution in database systems," *IEEE Trans. Autom. Control*, Vol. 33, pp. 439–447, 1988.

[26] S. Lafortune and E. Wong, "A state model for the concurrency control problem in database management systems," in *Proc. 24th IEEE Conf. Decision and Control*, 1985.

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

[28] F. Lin and W. M. Wonham, "On observability of discrete event systems," *Inf. Sci.*, Vol. 44, pp. 173–198, 1988.

[29] F. Lin and W. M. Wonham, "Decentralized supervisory control of discrete event systems," *Inf. Sci.*, Vol. 44, pp. 199–224, 1988.

[30] J. Loeckx and K. Seiber, *The Foundations of Program Verification*, 2nd ed. New York: Wiley, 1987.

[31] Z. Manna and A. Pnueli, "The anchored version of the temporal framework," *Lect. Notes Comput. Sci.*, Vol. 354, pp. 201–284, 1989.

[32] G. J. Milne, "CIRCAL and the representation of communication, concurrency, and time," *ACM TOPLAS*, Vol. 7, pp. 270–298, 1985.

[33] R. Milner, "A calculus for communicating systems," *Lect. Notes Comput. Sci.*, Vol. 92, 1980.

[34] E.-R. Olderog, "Process theory: Semantics, specification and verification," in *Lect. Notes Comput. Sci.*, Vol. 224, pp. 442–509, 1986.

[35] E.-R. Olderog and C. A. R. Hoare, "Specification-oriented semantics for communicating

processes," *Acta Inf.*, Vol. 23, pp. 9–66, 1986.

[36] C. M. Ozveren, A. S. Willsky, and P. J. Antsaklis, "Stability and stabilizability of discrete event dynamic systems," 1989, Preprint.

[37] C. M. Ozveren and A. S. Willsky, "Observability of discrete event dynamic systems," 1989, Preprint.

[38] C. M. Ozveren and A. S. Willsky, "Aggregation and multi-level control in discrete event dynamic systems," 1989, Preprint.

[39] J. L. Peterson, *Petry Nets and Modeling of Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[40] A. Pnueli, "Application of temporal logic to the specification and verification of reactive systems: A survey of current trends," *Lect. Notes Comput. Sci.*, Vol. 224, pp. 510–584, 1986.

[41] A. Pnueli, "Linear and branching structures in the semantics and logics of reactive systems," *Lect. Notes Comput. Sci.*, Vol. 194, pp. 15–32, 1985.

[42] P. J. Ramadge, "Observability of discrete event systems," in *Proc. 25th IEEE Conf. Decision and Control*, pp. 1108–1112, 1986.

[43] P. J. Ramadge, "Some tractable problems in the supervisory control of discrete event systems described by Buchi automata," *IEEE Trans. Auto. Control*, AC-34, pp. 10-19, 1989.

[44] P. J. Ramadge, "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. New York: Springer-Verlag, 1988, pp. 171–190.

[45] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Control and Optimization*, Vol. 25, pp. 206–230, 1987.

[46] R. Smedinga, "Control of discrete systems," University of Groningen, doctoral thesis, 1989.

[47] G. Thistle and W. M. Wonham, "Control problems in temporal-logic framework," *Int. J. Control*, Vol. 44, pp. 943–976, 1986.

[48] J. Tsitsiklis, "On the control of discrete event dynamical systems," *Math. Control, Signals Systems*, Vol. 2, pp. 95–107, 1989.

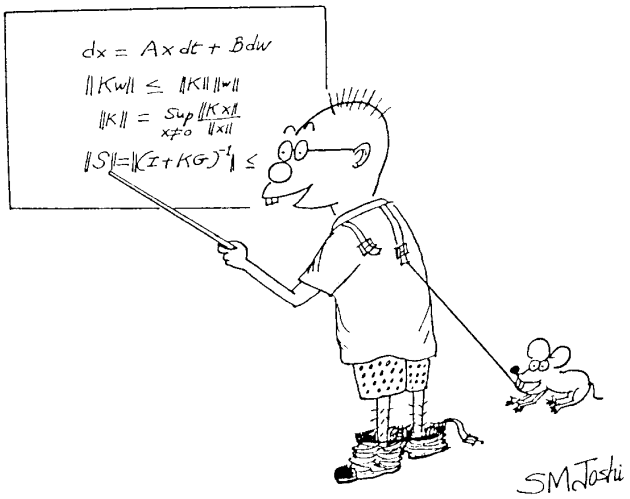
[49] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Control Optimization*, Vol. 25, pp. 637–659, 1987.

[50] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete event systems," *Math. Control, Signals Systems*, Vol. 1, pp. 13–30, 1988.

[51] H. Zhong and W. M. Wonham, "On hierarchical control of discrete event systems," in *Proc. 22nd Ann. Conf. Information Sci. Systems*, Princeton, NJ, 1988, pp. 64–70.

Michael Heymann received the B.Sc. and M.Sc. degrees from the Technion, Haifa, Israel, in 1960 and 1962, respectively, and the Ph.D. degree from the University of Oklahoma, Norman, in 1965, all in chemical engineering. During 1965–1966 he was on the Faculty of the University of Oklahoma. From 1966 to 1968 he was with Mobil Research and Development Corporation, engaged in research in control and systems theory. From 1968 to 1970 he was with the Ben-Gurion University of the Negev, Beer-Sheva, where he established and headed the Department of Chemical Engineering. Since 1970 he has been with the Technion where he is currently Professor in the Department of Computer Science holding the Carl Fechheimer Chair. He has previously been with the Department of Applied Mathematics of which he was Chairman and the Department of Electrical Engineering. He held visiting positions at various institutes, including the University of Toronto, the University of Florida, the University of Eindhoven, Concordia University, CSIR, Yale University, and the University of Bremen. During 1983–1984, 1988–1989, as well as during the summers of 1985, 1986 and 1987 he was at NASA-Ames Research Center as an NRC-Senior Research Associate. His research covered topics in the areas of linear system theory, differential games, optimization and adaptive control. His current interest is chiefly in the area of discrete event systems and the theory of concurrent processes.

Out of Control



"...and so we come to the bottom line!"