DISCRETE EVENT SYSTEMS

A discrete event system (DES) can be defined as a dynamic system for which the state changes in response to the occurrence of discrete events. The discrete events take place at possibly irregular or unknown points in time (i.e., asynchronously and nondeterministically) but are the result of interactions within the system itself. The acronym DES, or frequently DEDS (for discrete event dynamic systems), has been used extensively in many different fields of mathematics and applications to designate apparently widely different systems. Nevertheless, all these systems have in common the property of being driven by events, rather than by time. The conceptual structure of a DES is deceptively simple. It is a system composed of multitudes of "jobs" that require various services from a multitude of "resources." The limited availability of the resources determines the interactions between the jobs, whereas the start and the completion of the jobs, as well as the changes of the resources, generate the events that govern the dynamics of the system. But this conceptually simple model encompasses scores of event-driven, mostly human-made, overwhelmingly complex systems: large international airports, automated manufacturing plants, military logistic systems, emergency hospital wards, offices, services and spare parts operations of multinational companies, distributed computing systems, large communication and data networks, very large scale integrated circuits (VLSI), electronic digital circuits and so on. Typical examples of events that can trigger the response of a DES and the possible change of its state are the arrival or the departure of a customer in a queue, the arrival or the departure of a packet in the node of a communication network, the completion of a task, the failure or the repair of a machine in a factory, the opening or the closing of a switch in an electrical network, the pressing of a key on the keyboard of a personal computer (PC), the accessing or the leaving of a resource, and so on.

System theory has traditionally been concerned with continuous variable dynamic systems (CVDSs) described by differential equations, possibly including random elements. The essential feature of CVDSs is that they are driven by time, which governs their dynamics. The discrete-time systems, for which the time instances are elements of a sequence, are described by difference equations instead of differential equations, but they essentially belong to the CVDS approach as long as their variables can take numerical values and are time-driven. In most cases, the discrete-time systems can be considered merely computational models, obtained by the sampling of the continuous-time systems. The CVDS approach is a powerful paradigm in modeling real-world "natural" systems. Currently, CVDSs are the main objects of what forms the core of our scientific and technical knowledge, ranging from Galileo's and Newton's classical mechanics to relativist and quantum mechanics, thermodynamics, electrodynamics and so on. CVDS models have also been highly successful in most engineering fields to describe low- or medium-complexity man-made systems and are still the main objects of control theory.

With the continuous and rapid increase in complexity of the systems to be modeled, analyzed, designed, and controlled, especially of the human-made systems that include computer and communication subsystems as essential components, systems too complex to allow a classical CVDS description have emerged. For such systems, the variables attached to the states and to the processes can have not only numerical values, but also symbolic or logical values. This motivates the interest in DESs in domains as different as manufacturing, robotics, vehicular traffics, conveyance and storage of goods, organization and delivery of services, and computer and communication networks, with particular emphasis on database management, computer operating systems, concurrent programming, and distributed computing. In all these domains, control is necessary to ensure the orderly flow of events in highly complex systems. Significant efforts have been made in the last two decades to develop a comprehensive framework to handle DESs. The DES theory, even if still in its infancy when compared to the differential/difference equations paradigm underlying the CVDS theory, is fast growing at the confluence of artificial intelligence, operations research, and control system theory. Notable among the various approaches that have been used to represent DESs are the state machines and formal languages models (1–19), Petri nets (20–29), timed marked graphs (30–33), Markov chains (34,35), and generalized semi-Marcov processes (GSMP) (36,37).

These models allowed the analysis of DES qualitative properties, the quantitative evaluation of DES performances by methods as perturbation analysis (38–40) and likelihood ratio method (41,42), as well as progress in the design and control of DESs. Even if a general theory of DESs does not yet exist, the previously mentioned partial approaches have provided valuable concepts and insights and have contributed to the understanding of the fundamental issues involved in the analysis, design, and control of DESs. Discrete system simulation methods, algorithms, and software are now commercially available for both qualitative behavior analysis and quantitative performance evaluation (43–57).

Because of the complexity and the heterogeneity of the domain, as well as its fast growth, only some of the very basic aspects will be presented in the rest of this article. The main attention is focused on the modeling of DESs, which allows one to grasp the basic features and the behavior of DESs. Some elements of the control of DESs are presented, and some examples of DES application are given.

MODELS OF DISCRETE EVENT SYSTEMS

The increased complexity of human-made systems, especially as an effect of the widespread application of information technology, has made the development of more detailed formal methods necessary to describe, analyze, and control processes observed in environments such as digital communication networks and manufacturing plants. As opposed to the continuous time-driven evolution of a CVDS [Fig. 1(a)], the evolution of a DES is piecewise-constant and event-driven [Fig. 1(b)]. The state variables of a DES may have not just numerical values, but also symbolic or logical values, so that the set of states Q does not have the vector space structure typical for CVDS. The elements $q_j \in Q$, $j \in \mathbb{N}$, may be seen as labels attached to the various distinct states of the DES. The state transitions may occur in response to the occurrence of discrete events σ_k, belonging to a set of events Σ and taking place at discrete time instances t_k. From the point of view of the timing information, DESs can be classified into two main categories: (1) untimed (logical) and (2) timed.

Untimed Discrete Event Systems

Untimed or logical DES models ignore time as a variable that specifies the moments when the events occur. Only the order of the events is relevant for these models. The untimed DES models have been used for the deterministic qualitative analysis of control issues such as the reachability of states (18,58,59) or deadlock avoidance (23,60). Finite-state machine and Petri nets are the formal mechanisms mostly used for the representation of untimed DESs. Other untimed frameworks, such as the trace theory, have also been explored. Nevertheless, finite-state machines and their associated state transition graphs are still the most widely used models because of their inherent simplicity and because they can be described adequately by finite automata and regular languages. The simplest untimed DES model is a deterministic state-machine or automaton, called *generator*, described by the 4-tuple

$$G = (Q, \Sigma, \delta, s) \tag{1}$$

where Q is the (countable) set of states of the system, Σ is the (countable) set of events, $\delta\colon Q \times \Sigma \to Q \cup \{\Lambda\}$ is the transition function, and $s = q_0$ is the initial (start) state of the system. By a reminiscence of the classical system theory, the set of states is sometimes called the state space, even if it does not have the structure of a vector space, typical for the CVDSs. The function δ describes the transition from a state $q \in Q$ to a new state $q' = \delta(q, \sigma)$, in response to the occurrence of an event $\sigma \in \Sigma$. The symbol Λ denotes the null element, which is used to indicate that the transition is not defined for some pairs $(q, \sigma) \in Q \times \Sigma$. For this reason, $\delta\colon Q \times \Sigma \to Q$ is called a *partial function.* It is convenient to designate by $\Sigma^f(q)$ the set of all feasible events for a given state q, i.e., $\Sigma^f(q) = \{\sigma \in \Sigma | \delta(q, \sigma) \neq \Lambda\}$. As usual in the regular expressions formalism, we denote by Σ^* the set of all finite strings of elements of Σ, including the empty string ϵ. A sample path (trajectory) of a DES, starting from the specified initial state $q_0 = s$ [see Fig. 1(a)], is given by the state-(event-state) sequence $q_0\sigma_1q_1\sigma_2, \ldots, \sigma_nq_n$. The set of all (physically) possible such sequences is called the behavior $B(G)$ of the generator G:

$$B(G) = \{q_0\sigma_1q_1\sigma_2, \ldots, \sigma_nq_n | n \in \mathbb{N}^*, 1 \le k \le n, q_k = \delta(q_{k-1}, \sigma_k)\} \tag{2}$$

For a deterministic DES, the sample trajectory can be described equivalently by the event string $\{\sigma_k\}_{k=1,2,\ldots,n}$, or by the state string $\{q_k\}_{k=0,1,2,\ldots,n}$. In the formalism of regular languages, an event string corresponding to a sample trajectory is called a *word* w built with the symbols σ taken from the alphabet Σ. Correspondingly, the set of all the (physically) possible words is called the *language* $L(G) \subset \Sigma^*$ generated by G over the alphabet Σ. Sometimes, the language is also called the behavior of the DES, or the behavior of its generator. In the framework of automata theory, an automaton is described by a 5-tuple, which includes as a fifth element a set of marker states $Q_m \subseteq Q$. A marker state usually represents the completion of a task. This is not essential in this context, so it is deferred for the following section.

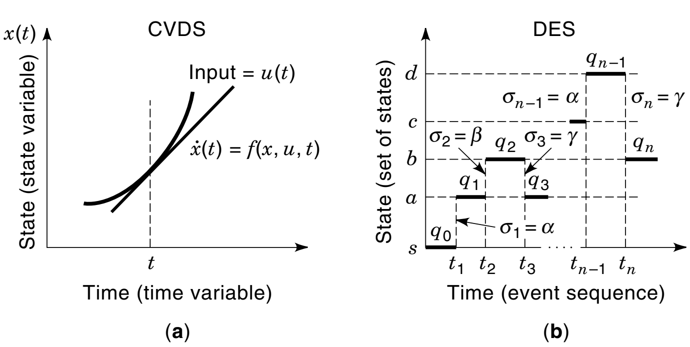

Figure 1. Comparison of generic trajectories of continuous variable dynamic systems and of discrete event systems: (a) Example of an illustrative one-dimensional CVDS trajectory. (b) Example of a DES trajectory ($\alpha, \beta, \gamma, \delta \in \Sigma$; $a, b, c, d, s \in Q$).

Example 1. Consider a DES generator G that models a simple generic machine. The state set $Q = \{I, W, D,\}$ is composed of the states: I—Idle, W—Working, and D—Down, whereas the event set $\Sigma = \{S, C, B, R\}$ is composed of the events: S—Start of a task, C—Completion of the task, B—Breaking

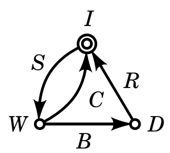

Figure 2. The transition graph of a simple generic machine model. The system can be in the states: I—Idle, W—Working, and D—Down, and the transitions are induced by the events: S—Start of a task, C—Completion of the task, B—Breaking down, and R—Repair.

down, and R—Repair. Figure 2 shows the transition function of the system. The states are designated by nodes, and the events by oriented arcs connecting the nodes. The initial state $s = I$ is marked with an entering arrow. The language generated by G, i.e., the set of all the (physically) possible sequences of events is

$$L(G) = \{\epsilon, S, SD, SC, SCS, SCSD, SDR, SDRS, SDRSD, \ldots\}$$

which can be written in the formalism of regular expressions as $L(G) = (SC + SDR)^*(\epsilon + S + SD)$.

Example 2. Let us now consider the case of two machines of the type given in Example 1 working in parallel. Each machine has a generator of the previously considered type. The transition graphs of the two machines working as independent entities are represented in Fig. 3. The system composed of the two machines working in parallel, even without conditioning each other, has the state set $Q = Q_1 \times Q_2 = \{(I_1, I_2), (W_1, I_2), (D_1, I_2), \ldots, (D_1, D_2)\}$, the set of events $\Sigma = \Sigma_1 \cup \Sigma_2 = \{S_1, C_1, B_1, R_1, S_2, C_2, B_2, R_2\}$, and the transition graph shown in Fig. 4. The combinatorial growth in complexity of a DES with the increase of the number of components is obvious.

Since untimed models contain no quantitative timing information, they cannot be used to obtain performance measures involving time, such as holding times or event occurrence rates. Nevertheless, logical DES models have successfully been used to represent and study qualitative aspects in areas such as concurrent program semantics, communicating sequential processes, synchronization in operating systems, supervisory control, communication protocols, logical analysis of digital circuits, and fault-tolerant distributed computing and database protocols. The control theory of discrete event systems has been initiated by Ramadge and Wonham

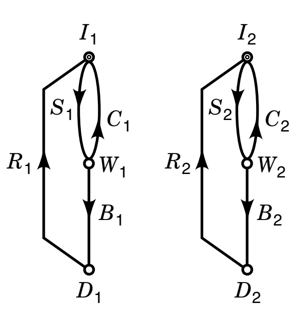

Figure 3. The transition graphs of two instances of the simple machine model in Fig. 2, operating independently.

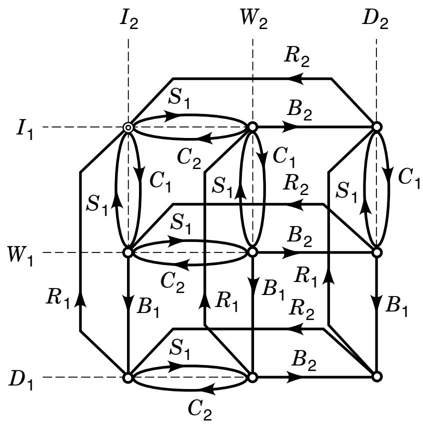

Figure 4. The transition graph of a system made up of the two instances of the simple machine model shown in Fig. 2, operating as elements of the system.

(see Refs. 16–19,61,62) in the framework of untimed DESs. The analysis of an untimed DES model typically proceeds as follows. By using some state transition structure (e.g., automata or Petri nets), a set of algebraic equations, or a logical calculus approach, one specifies the set of all admissible event trajectories, that is, enumerates all the sequences of events that do not contradict various physical restrictions inherent to the modeled system. On this basis, the behavior of the system—usually expressed by the generated language L, that is, by the set of all the possible finite sequences of events that can occur in the system—is found as a strict subset of all event orderings Σ^*. In the control context, one has to further restrict the language so that each system trajectory has some desired property such as stability (e.g., state convergence), correct use of resources (e.g., mutual exclusion), correct event ordering (e.g., data base consistency), desirable dynamic behavior (e.g., no deadlock/livelock), or the achievement of some goal (e.g., distributed consensus).

The difficulties in applying logical DES models to real-life size problems are caused by the computational complexity. Even if problems like establishing controllability or designing a supervisor to control the behavior of a DES are polynomially decidable or polynomially solvable in the number of states of the DES, the number of states itself grows in a combinatorial manner when a complex system is built from simpler component subsystems. As a consequence, the number of the states of a logical DES increases exponentially with respect to the system size. This motivates the efforts to state/event formalisms that have the capability to suppress the aspects of the system description irrelevant in a given context. One modality is *event internalization,* or *partial observation,* which leads to nondeterministic process behavior and, consequently, to inadequacy of formal languages as models of behavior. The complexity issues are also talked with by using modularity, hierarchy, and recursivity when building the system descriptions from the individual component features. Since all the components of a complex process must interact and synchronize when operating in parallel, a suitable mechanism for communication and interaction between modules is an important component of DES modeling.

Markov Chain Model of an Untimed DES. One way of modeling the random behavior of discrete event systems is by using the Markov chain formalism. As pointed out earlier, the nondeterministic behavior of a system can be the result of its incomplete (partial) description. Either some of the events are aggregated into complex events that can yield multiple outcomes (event internalization) or the states of the system are defined in a space of lower dimension than would be required for their complete specification (hidden variables) so that the states aggregate and actually correspond to classes of states. Partial description can be necessary and desirable in order to reduce the computational difficulties—to make complex systems tractable—or can result from incomplete knowledge about the modeled system. On the other hand, randomness can be an irreducible feature of some of the processes in the system itself. The Quantum Mechanics approach is the first example at hand. The problem of whether or not such built-in randomness does exist is still open to philosophical debate. From the engineering point of view, this is irrelevant because the behavior of the system is similar in both cases.

A Markov chain model of a nondeterministic DES is defined by the set of states Q and the transition probability matrix $P^S = [P^S_{ij}]$, where

1. $P^S_{ij} = P(q_j|q_i)$, for $i \neq j$, is the conditional probability that the system passes into the state $q_j \in Q$, i.e., the probability of occurrence of event $\sigma_{ij} = (q_i, q_j)$, provided that the current state is $q_i \in Q$.
2. $P^S_{ij} = 1 - \sum_{j \neq i} P^S_{ij}$ is the probability of remaining in the state q_i, which is the probability of occurrence of event $\sigma_{ii} = (q_i, q_i)$, if $\sigma_{ii} \in \Sigma^f$, or the probability that no event occurs in the state q_i, if $\sigma_{ii} \notin \Sigma^f$.

The probability that, starting from the initial state s $=$ $q(0) = q_i$, the system arrives after n steps into the state $q(n) = q_j$ is denoted by $P^S_{ij} = P[q(n) = q_j|q(0) = q_i]$. Thus, the entries of the transition probability matrix give the probabilities of paths of length one:

$$P^S_{ij} = P[q(n+1) = q_j|q(n) = q_i]$$

Markov chains can be used to represent the "closed loop" behavior of a controlled DES. In this case, the probabilities of the enabled transitions (events) are strictly positive, whereas the probabilities of the disabled transitions are zero. The control of a DES modeled by a Markov chain consists thus in changing the transition probabilities, according to the commands issued by the supervisor, to achieve a certain controlling task.

Timed DES Models

Timed DES models were developed primarily to allow the quantitative evaluation of DESs by computing performance measures like holding times or event occurrence rates, which imply counting events in a given time interval or measuring the time between two specific event occurrences and obtaining the appropriate statistics. The timed event trajectory of a DES is specified by the sequence $\{\sigma_k, t_k\}_{k \in N^*}$, whereas the timed state trajectory is $\{q_k, t_k\}_{k \in N}$, where t_k gives the moment of the kth event occurrence. Significant analytical results have been obtained in the special case of queuing theory. For the systems that do not satisfy the specific hypotheses of the queuing theory, timed DES models have been studied by using simulation and statistical analysis, which is computationally costly and has little potential for real-time control. Both approaches were used for the evaluation of performances related to resource contention and allocation, based on the oversimplifying assumption that a manufacturing process can be described adequately by using only timing considerations. For instance, the problem of the yield percentage in semiconductor wafer manufacturing is more closely related to the properties of the materials and to the technological aspects than to resource contention.

Another approach is based on the fact that sample paths of parametric DESs contain a considerable amount of information that allows to predict the behavior of the system when the values of the parameters are perturbed. Both infinitesimal perturbation analysis (IPA) and likelihood ratio (LR) methodology have been used in conjunction with various gradient-based stochastic optimization schemes. These techniques yielded significant results in problems like routing in communication networks or load balancing in distributed processing systems.

In order to define a timed DES, a mechanism for generating the event time instance sequence $\{t_k\}_{k \in N}$ has to be added to the untimed model. This mechanism should also take into account the randomness of the event lifetime τ_σ, $\sigma \in \Sigma$. Cassandras and Strickland (36) have introduced a model to study the properties of the sample paths of a timed DES. The generator

$$G = \{Q, \Sigma, \delta, s, F\} \tag{3}$$

contains, in addition to the components of an untimed DES [Eq. (1)], the *event lifetime generator:*

$$F = \{F_\sigma(\cdot), \sigma \in \Sigma\} \tag{4}$$

which is a set of probability distribution functions (pdfs) associated with the events.

The basic simplifying hypothesis is that all events are generated through renewal processes, i.e., each pdf $F_\sigma(\cdot)$ depends only on the event σ, not on other factors such as the states before and after the event σ occurs and the count of how many events of type σ have already occurred.

Figure 5 shows a typical sample path of a timed DES. In the general case, the set of events Σ contains several types of events and it is possible that for some states q there are nonfeasible events σ, i.e., $\sigma \notin \Sigma^f(q)$. In the simplest case, when there is only one type of event in Σ and this event is feasible for all the states in the path, the kth lifetime $\tau_{k,i}$ of the event of type i characterized by the pdf $F_i(\cdot)$ gives the interval between two successive occurrences of the event $t_{k+1} - t_k = \tau_{k,i}$ where $k = 1,2,\ldots$. A certain time instant t in

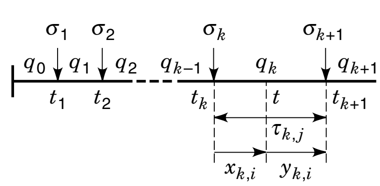

Figure 5. Generic sample path of a timed DES with one event type. The moment t divides the kth *lifetime* $\tau_{k,i}$ of event of type i into the *age* $x_{k,i}$ and the *residual lifetime* $y_{k,i}$.

this interval, $t \in [t_k, t_{k+1}]$, divides it into two parts that define the *age* $x_{k,i} = t - t_k$ of the event i (the time elapsed since its most recent occurrence), and the *residual lifetime* $y_{k,i} = t_{k+1} - t = \tau_{k,i} - x_{k,i}$ of the event of type i (the time until its next occurrence). When several types of events are possible, the next event occurrence is determined by the currently feasible event with the smallest residual lifetime $\sigma_{k+1} = \arg\min_{\sigma_i \in \Sigma^f(q_k)} \{y_{k,i}\}$, where $y_{k,i}$ is a random variable generated with the pdf:

$$H_{k,i}(u, x_{k,i}) = P[y_{k,i} \leq u | x_{k,i}] = P[\tau_{k,i} \leq x_{k,i} + u | \tau_{k,i} > x_{k,i}] = \frac{F_i(x_{k,i} + u) - F_i(x_{k,i})}{1 - F_i(x_{k,i})} \tag{5}$$

The dynamic model of a timed DES, allowing the step-by-step construction of a sample path, is thus given by

$$q_k = \delta(q_{k-1}, \sigma_k) \tag{6}$$

$$t_{k+1} = t_k + \min_{\sigma_j \in \Sigma^f(q_k)} \{y_{k,j}\} \tag{7}$$

$$\sigma_{k+1} = \arg\min_{\sigma_j \in \Sigma^f(q_k)} \{y_{k,j}\} \tag{8}$$

$$x_{k+1,i} = \begin{cases} x_{k,i} + \min\limits_{\sigma_j \in \Sigma^f(q_k)} \{y_{k,j}\}; & \text{if } \sigma_i \in \Sigma^f(q_k), \quad \sigma_i \neq \sigma_k \\ 0; & \text{otherwise} \end{cases} \tag{9}$$

for $k \in \mathbb{N}^*$ and for initial conditions specified by some given $s = q_o$ and $t_1 = \min_{\sigma_j \in \Sigma^f(q_0)} \{y_{1,j}\}$, $\sigma_1 = \arg\min_{\sigma_j \in \Sigma^f(q_0)} \{y_{1,j}\}$, and $x_{1,i} = \min_{\sigma_j \in \Sigma^f(q_0)} \{y_{1,j}\}$, where $y_{1,j}$ are random variables drawn from $F_j(\cdot)$ for all j. This stochastic dynamic model generates a generalized semi-Markov process. For such processes, a state is actually defined by two components: the *discrete state* $q_k \in Q$ and the so-called *supplementary variables* $x_{k,i}$ (or, equivalently, $y_{k,i}$), for all $i \in \Sigma$. A GSMP offers a convenient framework for representing timed DESs. The deterministic mechanism for the state transitions, defined here by the function $\delta(q, \sigma)$, can also be replaced by a probabilistic state transition structure. Even more than in the case of untimed DESs, despite the conceptual simplicity of the dynamics, the exhaustive analysis of a stochastic timed DES model can be of prohibitive computational complexity, not only because of the large number of states but also because of the nonlinearity of the equations and the age-dependent nature of the pdfs $H_{k,i}(u, x_{k,i})$. On the other hand, if the dynamic equations are seen as a sample path model, than the timed trajectories of DES can be generated relatively simple when the lifetime distributions $F_i(\cdot)$ are known for all $i \in \Sigma$. This allows the use of techniques like perturbation analysis (38) or the likelihood ratio method (39,40) for performance evaluation, control, or optimization purposes.

The stochastic model can be reduced to a deterministic one if the lifetimes are considered to be constants for all $i \in \Sigma$. The residual lifetimes of events are determined by $y_{k,i} = \tau_i - x_{k,i}$, for all i, k, whereas the event ages $x_{k,i}$ result from the state equation

$$x_{k+1,i} = \begin{cases} x_{k,i} + \min\limits_{j \in \Sigma^f(q_k)} \{\tau_j - x_{k,j}\}; & \text{if } i \in \Sigma^f(q_k) \backslash \{\sigma_k\} \\ 0; & \text{otherwise} \end{cases} \tag{10}$$

with $x_{1,i} = 0$. The event time instances are given by the time Eq. (7)

$$t_{k+1,i} = t_{k,i} + \min_{j = \Sigma^f(q_k)} \{\tau_j - x_{k,j}\} \tag{11}$$

The model is similar to the one used by the *min-plus* dioid algebra approach presented in a later section of this article.

Formal Languages and Automata DES Models

Formal Languages—Regular Expressions. Using the previous notation, let the generator G of an untimed (logical) DES have the finite state set Q, the finite set of events Σ, and the behavior described by the set of all (physically) possible finite event strings $L(G) \subset \Sigma^*$, a proper subset of Σ^*—the set of all finite strings built with elements of the alphabet Σ, including the empty string ϵ. In the formal language approach, let us consider that each event is a symbol, the event set Σ is an alphabet, each sample event path $w = \sigma_1\sigma_2 \ldots \sigma_n$ of the DES is a word, and the (event) behavior $L(G)$ is a language over Σ. The length $|w|$ of a word w (i.e., of a sample path) is the number of symbols σ from the alphabet Σ (i.e., events) it contains. The length of ϵ is zero.

Given two languages, L_1 and L_2, their *union* is defined by

$$L_1 + L_2 = L_1 \cup L_2 = \{w | w \in L_1 \quad \text{or} \quad w \in L_2\} \tag{12}$$

whereas their *concatenation* is

$$L_1 L_2 = \{w | w = w_1 w_2, \quad w \in L_1 \quad \text{or} \quad w \in L_2\} \tag{13}$$

The *Kleene (iterative) closure* of a language L is

$$L^* = \{w | \exists k \in \mathbb{N} \quad \text{and} \quad w_1, w_2, \ldots, w_k \in L \quad \text{so that} \quad w = w_1 w_2 \cdots w_k\} \tag{14}$$

The union, concatenation, and Kleene closure are *regular operators.*

A string u is a *prefix* of $w \in \Sigma^*$, if there is some $v \in \Sigma^*$ so that $w = uv$. If $w \in L(G)$, then so are all its prefixes. A prefix is called proper if $v \notin \{\epsilon, w\}$.

The *prefix closure* of $L \subset \Sigma^*$ is

$$\overline{L} = \{u | uv \in L \quad \text{for some } v \in \Sigma^*\} \tag{15}$$

A language L is prefix closed if $\overline{L} = L$, i.e., if it contains the prefixes of all its words.

The (event) behavior of a DES can be modeled as a prefix closed language L over the event alphabet Σ. In the following, the main relevant propositions will be stated, but the proofs will be omitted for briefness. We will write v^*, $u + v$, and so on, instead of $\{v\}^*$, $\{u\} + \{v\}$, when no confusion is possible.

A regular expression in $L_1, L_2, \ldots, L_m \subset \Sigma^*$ is any expression in $L_1, L_2, \ldots, L_m$ containing a finite number of regular operators. A language is called regular if it can be defined by a regular expression in a finite set of symbols, i.e., events.

The set $\mathscr{R}$ of regular languages over an alphabet Σ is the smallest set of languages satisfying:

$$\begin{aligned} &1.\ \phi = \{\ \} \in \mathscr{R}, \{\epsilon\} \in \mathscr{R}, \\ &2.\ \{a\} \in \mathscr{R}, \quad \text{for} \quad \forall a \in \Sigma, \\ &3.\ \forall A, B \in \mathscr{R}, A \cup B, AB, A^* \in \mathscr{R}. \end{aligned} \tag{16}$$

Regular expressions are notations for representing the regular languages, constructed with these rules:

1. ϕ, ϵ, and the elements of the alphabet Σ are regular expressions.
2. If α and β are regular expressions, then $\alpha \cup \beta$, $\alpha\beta$, α^* are also regular expressions.

Obviously, a regular expression can be considered itself a word (a string of symbols) over the alphabet $\Sigma' = \Sigma \cup \{), (, \phi, \cup, *, \epsilon\}$.

A language $L(\xi)$, represented by a regular expression ξ, is defined by

1. $L(\phi) = \phi$, $L(\epsilon) = \{\epsilon\}$,
2. $L(a) = \{a\}$, $\forall a \in \Sigma$,
3. $L(\alpha \cup \beta) = L(\alpha) \cup L(\beta)$, (17)
4. $L(\alpha\beta) = L(\alpha)L(\beta)$,
5. $L(\alpha^*) = L(\alpha)^*$.

It can be shown that a language is regular if it is represented by a regular expression. The set of all the words constructed with the symbols from an alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \alpha_n\}$, including the empty word ϵ, is represented by the regular expression $\Sigma^* = \{\sigma_1 + \sigma_2 + \cdots + \sigma_n\}^*$. The set of all the nonempty words constructed with symbols from Σ is given by the regular expression $\Sigma^+ = \Sigma\Sigma^*$.

DES Deterministic Generators. Consider the generator of a DES modeled by a *finite deterministic state machine* (automaton) defined now by the 5-tuple

$$G = \{Q, S, d, s, Q_{\mathrm{m}}\} \tag{18}$$

where Q is a (finite) state set, Σ is the (finite) alphabet recognized by G, δ: $Q \times \Sigma \to Q$ is the transition function, $s = q_0$ is the initial state, and $Q_{\mathrm{m}} \subseteq Q$ is the set of marker states. For sake of simplicity, we considered here $\Sigma^f(q) = \Sigma$, $\forall q \in Q$ [see comments on Eq. (1)]. As already mentioned, the marker states have been introduced by Ramadge and Wonham (see Ref. 16) to represent the completed tasks of a DES by the state trajectories that end in (or contain a) marker state. Therefore, along with $B(G)$, the previously defined unmarked behavior of a DES [Eq. (2)], we define the marked behavior

$$B_{\mathrm{m}}(G) = \{q_0\sigma_1 q_1\sigma_2, \ldots, \sigma_n q_n \in B(G) | q_n \in Q_{\mathrm{m}}\} \tag{19}$$

which includes all the system trajectories that end in a marked state, i.e., result in the accomplishment of a certain task. Correspondingly, in addition to the *language generated by G,* the subset $L(G) \subset \Sigma^*$ of all the (physically) possible words generated by G over the alphabet Σ,

$$L(G) = \{w = \sigma_1\sigma_2, \ldots, \sigma_n \in \Sigma | q_0\sigma_1 q_1\sigma_2, \ldots, \sigma_n q_n \in B(G)\} \tag{20}$$

we define the language marked or accepted by G, as the restricted subset $L_{\mathrm{m}}(G) \subseteq L(G)$

$$L_{\mathrm{m}}(G) = \{w = \sigma_1\sigma_2, \ldots, \sigma_n \in \Sigma^* | q_0\sigma_1 q_1\sigma_2, \ldots, \sigma_n q_n \in B_{\mathrm{m}}(G)\} \tag{21}$$

which is composed the words that start from the specified initial state $s = q_0$, and lead to a marked state $q_n \in Q_{\mathrm{m}}$. Because the marked language $L_{\mathrm{m}}(G)$ is a subset of the language $L(G)$, so is its prefix closure [see Eq. (15)] $\overline{L}_m(G) \subseteq L(G)$, i.e., every prefix of $L_{\mathrm{m}}(G)$ is also an element of $L(G)$. A generator G is called *nonblocking* if the equality $\overline{L}_{\mathrm{m}}(G) = L(G)$ holds, meaning that every word in $L(G)$ is a prefix of a word in $L_{\mathrm{m}}(G)$. In this case, every sample path of events in $L(G)$ can be extended to include a marker state or—in other words—can be continued to the completion of a task.

The links between the states $q \in Q$ and the words $w \in \Sigma^*$ can be put on a more formal basis using the concept of configuration. The configuration of a finite automaton is defined by the ordered pair $(q, w) \in Q \times \Sigma^*$, which makes up a state q and a word w applied in this state.

A configuration (q', w') can be derived from a configuration (q, w) by the generator G, the relation of which is denoted by $(q, w) \overset{*}{\underset{G}{\mapsto}} (q', w')$, if there is a finite number $k \geq 0$ and a sequence $\{(q_i, w_i)|0 \leq i \leq k - 1\}$ so that $(q, w) = (q_0, w_0)$, $(q', w') = (q_k, w_k)$, and (q_i, w_i), $\mapsto (q_{i+1}, w_{i+1})$, for every i, $0 \leq i \leq k$, i.e., $w_i = \sigma_{i+1} w_{i+1}$, $q_{i+1} = \delta(q_i, \sigma_i)$. Each word w_i is composed of the first symbol σ_{i+1} and the remaining word w_{i+1}, so that the words in the sequence are related by

$$w = w_0 = \sigma_1 w_1 = \sigma_1\sigma_2 w_2 = \cdots = \sigma_1\sigma_2\cdots\sigma_k w_k = \sigma_1\sigma_2\cdots\sigma_k w' \tag{22}$$

The execution of an automaton on a word w is $(s, w) \mapsto (q_1, w_1) \mapsto \cdots \mapsto (q_n, \epsilon)$, with

$$w = \sigma_1 w_1 = \sigma_1\sigma_2 w_2 = \cdots = \sigma_1\sigma_2\cdots\sigma_n \tag{23}$$

For a deterministic automaton, each word w defines a unique execution, thus a unique trajectory of the system.

Using this formalism, a word w is accepted or marked by a generator (automaton) G if the execution of the automaton on the given word leads to a marker state $q_n \in Q_{\mathrm{m}}$:

$$(q_0, w) \overset{*}{\underset{G}{\mapsto}} (q_n, \epsilon);\, q_n \in Q_{\mathrm{m}} \tag{24}$$

The language $L_{\mathrm{m}}(G)$ accepted or marked by the automaton G is the set of words accepted by G:

$$L_{\mathrm{m}}(G) = \{w \in \Sigma^* | (q_0, w) \overset{*}{\underset{G}{\mapsto}} (q_n, \epsilon);\, q_n \in Q_{\mathrm{m}}\} \tag{25}$$

DES Nondeterministic Generators. A *finite nondeterministic state machine* (automaton) is the 5-tuple

$$G = \{Q, \Sigma, \Delta, s, Q_{\mathrm{m}}\} \tag{26}$$

where Q, Σ, $s = q_0$, Q_{m} retain the meanings defined for deterministic generators [Eq. (18)], whereas the evolution law is given by the transition relation $\Delta \subset Q \times \Sigma \times Q$, which generalizes the previously defined transition function δ. For a given state $q \in Q$, an event $\sigma \in \Sigma$ can induce a transition of the system to a state $p \in Q$, with $(q, \sigma, p) \in \Delta$. The set of states reachable in one step from the state q, after a transition induced by the event σ, is

$$Q(q, \sigma) = \{p \in Q | (q, \sigma, p) \in \Delta\} \tag{27}$$

The set $\Sigma^f(q)$ of all feasible events for a given state q can be expressed as

$$\Sigma^f(q) = \{\sigma \in \Sigma | \exists p \in Q, (q, \sigma, p) \in \Delta\} = \{\sigma \in \Sigma | Q(q, \sigma) \neq \varnothing\} \tag{28}$$

The *deterministic generator* can be seen as a special case of the nondeterministic generator with the property that, for all $q \in Q$ and $\sigma \in \Sigma$, there exist at most one state $p \in Q$ such that $(q, \sigma, p) \in \Delta$. In this case, a transition function δ: $Q \times \Sigma \to Q \cup \{\Lambda\}$ can be defined such that $\delta(q, \sigma) = p \in Q$, when $(q, \sigma, p) \in \Delta$, and $\delta(q, \sigma) = \Lambda$, when $(q, \sigma, p) \notin \Delta$, i.e., when $\sigma \notin \Sigma^f(q)$.

It is convenient to extend further the definition of the evolution law to a relation $\Delta^* \subset Q \times \Sigma^* \times Q$, by stating $(q_0, w, q_n) \in \Delta^*$ if there exist the sequences $\{q_k | q_k \in Q, k = 0, 1, \ldots, n\}$ and $w = \{\sigma_k | \sigma_k \in \Sigma, k = 1, 2, \ldots, n\} \in \Sigma^*$, such that $(q_{k-1}, \sigma_k, q_k) \in \Delta$, for all $k = 1, 2, \ldots, n$.

Using the relation Δ^*, the language generated by G can be expressed as

$$L(G) = \{w \in \Sigma^* | \exists q \in Q: (q_0, w, q) \in \Delta^*\} \tag{29}$$

and the language accepted or marked by G as the restricted subset $L_m(G) \subseteq L(G)$

$$L_m(G) = \{w \in \Sigma^* | \exists q_m \in Q_m; (q_0, w, q_m) \in \Delta^*\} \tag{30}$$

A configuration (q', w') is derivable in one step from the configuration (q, w) by the generator G, the relation of which is denoted by $(q, w) \underset{G}{\mapsto} (q', w')$, if $w = uw'$, (i.e., the word w begins with a prefix $u \in \Sigma^*$) and $(q, u, q') \in \Delta^*$.

A class of equivalent states is a set of states that have the property that the system can pass from one state in the class to another without the occurrence of any event, i.e., by transitions on the empty word ϵ. The equivalence class $E(q)$ of a state q is defined as an equivalence class comprising the state, q, i.e., the set of states reachable from the state q by transitions on the empty word

$$E(q) = \{p \in Q | (q, w) \overset{*}{\underset{G}{\mapsto}} (p, w)\} = Q(q, \epsilon) \tag{31}$$

Two generators G_1 and G_2 are called equivalent if $L(G_1) = L(G_2)$.

For any nondeterministic finite generator $G = \{Q, \Sigma, \Delta^*, q_0, Q_m\}$, it is possible to build formally an equivalent deterministic finite generator $G' = \{\boldsymbol{Q}', \Sigma', \delta', \boldsymbol{q}'_0, \boldsymbol{Q}'_m\}$, for which the states are replaced with classes of equivalent states. Correspondingly, the state set becomes the set of equivalence classes $\boldsymbol{Q}' \subseteq 2^Q$ (the set of the subsets of the state set Q), the initial state is replaced by the set $\boldsymbol{q}'_0 = E(q_0) = Q(q_0, \epsilon)$ of states in which the generator can be before any event occurs, the transition function is defined by $\delta'(\boldsymbol{q}, \sigma) = \cup_{p \in Q} \{E(p) | \exists q \in \boldsymbol{q} : (q, \sigma, p) \in \Delta^*\}$, and the set of marker equivalence classes is $\boldsymbol{Q}'_m = \{\boldsymbol{q} \subset \boldsymbol{Q}' | \boldsymbol{q} \cap Q_m \neq \phi\}$. The last equation shows that a "state" of G' is a marker state if it contains a marker state of G.

Regular Languages and Finite Automata Representation. As stated earlier, regular expressions and finite automata are formalisms adequate for representing regular languages, as well as for representing the behaviors of DESs, which are languages over some alphabets of events. The following propositions express the fundamental links between regular languages and finite automata:

- A language is regular if it is accepted by a finite automaton.
- If a language can be constructed by a regular expression, then it is accepted by a finite nondeterministic automaton.
- For each basic regular expression ϕ, ϵ, $\sigma \in \Sigma$, there is an automaton that accepts the corresponding language as shown in Fig. 6.
- For each composed regular expression $\alpha_1\alpha_2$, $\alpha_1 + \alpha_2$, α_1^*, an automaton accepting the same language can be built based on the automata A_1 and A_2 that accept the languages described by α_1 and α_2, respectively: $\alpha_1 \Leftrightarrow A_1 = \{Q_1, \Sigma, \Delta_1^*, q_0^{(1)}, Q_m^{(1)}\}$, $\alpha_2 \Leftrightarrow A_2 = \{Q_2, \Sigma, \Delta_2^*, q_0^{(2)}, Q_m^{(2)}\}$. For instance, the automaton A corresponding to the regular expression $\alpha_1\alpha_2$ is $\alpha_1\alpha_2 \Leftrightarrow A = \{Q, \Sigma, \Delta^*, q_0, Q_m\}$, where $Q = Q_1 \cup Q_2$, $\Delta = \Delta_1 \cup \Delta_2 \cup \{(q, \epsilon, q_0^{(2)}) | q \in Q_m^{(1)}\}$, $q_0 = q_0^{(1)}$, $Q_m = Q_m^{(2)}$.

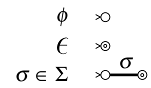

Figure 6. Elementary automata that accept the languages corresponding to the basic regular expression ϕ, ϵ, and $\sigma \in \Sigma$.

Algorithm for Constructing the Marked Language of a Generator *G*. Consider again the generator of a DES $G = \{Q, \Sigma, \Delta, s, Q_m\}$, with the finite set of states $Q = \{q_1, q_2, \ldots, q_n\}$, where the order is arbitrary. Let us find the language $L_m(G)$ marked by G, i.e., the set of words over the alphabet Σ that end in a marker state [see comments on Eq. (30)].

Let us denote by $R(i, j, k)$ the partial language made up of the set of words allowing the transition from the state q_i to the state q_j, passing either directly, or only through states with indices lower than k. Then

$$R(i, j, 1) = \begin{cases} \{w | (q_i, w, q_j) \in \Delta^*\}, & i \neq j \\ \{\epsilon\} \cup \{w | (q_i, w, q_j) \in \Delta^*\}, & i = j \end{cases} \tag{32}$$

and the following recurrence relation holds:

$$R(i, j, k+1) = R(i, j, k) \cup R(i, k, k) \cup R(k, k, k)^* R(k, j, k), \quad k = 1, 2, \ldots, n \tag{33}$$

Choosing the initial state $s = q_1$, the language $L_m(G)$ marked by G results:

$$L_m(G) = \bigcup_{q_j \in Q_m} R(1, j, n+1) \tag{34}$$

Both the partial languages R and the language $L_m(G)$ are regular languages.

Example 3. Consider a simple DES, having the generator G given by Eq. (26), with the state set $Q = \{q_1, q_2\}$, the event set $\Sigma = \{a, b\}$, the initial state $s = q_1$, the set of marker states

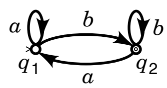

Figure 7. Transition graph of a simple determinist generator. The initial state $s = q_1$ is marked with an entering arrow, whereas the marker state q_2 is represented with a double circle.

$Q_m = \{q_2\}$, and the transition relation $\Delta = \{(q_1, \epsilon, q_1), (q_1, a, q_1), (q_1, b, q_2), (q_2, a, q_1), (q_2, \epsilon, q_2), (q_2, b, q_2)\}$, for which corresponds the transition graph in Fig. 7. Using the relations (32) and (33), the partial languages $R(i, j, k)$, $i, j, k = 1, 2$, of G listed in Table 1 can be computed successively. Thus, the language accepted by G results:

$$\begin{aligned} L * G) &= R(1,2,3) \\ &= [b \cup (\epsilon \cup a)(\epsilon \cup a)^* b] \cup [b \cup (\epsilon \cup a)(\epsilon \cup a)^* b] \\ &\quad [\epsilon \cup b) \cup a(\epsilon \cup a)^* b]^* [(\epsilon \cup b) \cup a(\epsilon \cup a)^* b] \end{aligned}$$

Max-Plus Algebra Representation of Timed Discrete Event Systems

The *max-plus* (*max*, +) algebra deals with a subclass of the timed Petri nets, namely the timed event graphs. Originally, Petri nets were introduced as nontimed logical models. Timed Petri nets have been developed for modeling and performance analysis, but were found less adequate for control purposes. The theory of timed DES emerged from the combination of the max-plus algebra framework with the system-theoretic concepts. The trends of the research on the max-plus algebra approach to DESs can be found in Ref. 23. Max-plus algebra is a convenient formalism for the systems in which synchronization is a key request for event occurrence, including both discrete events systems and continuous systems that involve synchronization. Max-plus algebra adequately describes systems for which the start of an activity requires the completion of all the activities that provide the inputs needed to perform the considered activity. In such cases, maximization is the basic operation. The complementary case is that of the systems in which an activity starts when at least one input becomes available. Minimization is the basic operation and the min-plus algebra is the adequate algebraic structure. These two limit cases correspond to the AND and OR operators from the binary logic, respectively. In mixed systems, both types of conditions can be present, and other related (usually isomorphic) dioid algebraic structures must be used. In the following we will refer only to the max-plus case.

Table 1. Partial Languages of the Generator G in Example 1

$R(i, j, k)$	$k = 1$	$k = 2$
$R(1, 1, k)$	$\varepsilon \cup a$	$(\varepsilon \cup a) \cup (\varepsilon \cup a)(\varepsilon \cup a)^*(\varepsilon \cup a)$
$R(1, 2, k)$	b	$b \cup (\varepsilon \cup a)(\varepsilon \cup a)^* b$
$R(2, 1, k)$	a	$a \cup a(\varepsilon \cup a)^*(\varepsilon \cup a)$
$R(2, 2, k)$	$\varepsilon \cup b$	$(\varepsilon \cup b) \cup a(\varepsilon \cup a)^* b$

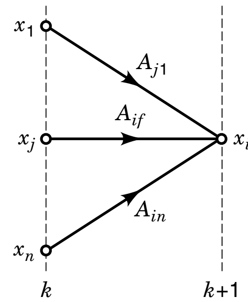

Figure 8. Section of a timed event graph showing only the edges coming into the node attached to event i. Input variables $x_j(k)$; $j = 1, \ldots, n$ give the moments when events j occur at step k, and the weights A_{ij}; $j = 1, \ldots, n$ of the edges correspond to the delays produced by the transport from j to i.

Consider the section of a timed event graph represented in Fig. 8. Each node corresponds to a certain activity, whereas the arcs coming into a node represent the conditions required to initiate the activity attached to the node. An event i (e.g., the start of a process) occurs at step $k + 1$ in the moment $x_i(k + 1)$ when all the input events (e.g., the end of the prerequisite processes) have occurred at step k in the respective moments $x_j(k)$; $j = 1, \ldots, n$, and have propagated from j to i with the transport delays A_{ij}; $j = 1, \ldots, n$. The corresponding discrete-time dynamic system model is given by the equations:

$$\begin{aligned} x_i(k+1) &= \max(A_{i1} + x_1^{(k)}, \ldots, A_{ij} + x_j^{(k)}, \ldots, A_{in} + x_n^{(k)}), \\ & i = 1, \ldots, n \end{aligned} \tag{35}$$

The analysis of this model is significantly simplified by the max-plus algebra formalism.

The max-plus algebra $(\mathbb{R}_{\max}, \oplus, \otimes)$ is a *dioid* over the set $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$, where $\mathbb{R}$ is the set of real numbers.

The *additive operation* $\oplus$ is the maximization

$$x \oplus y = \max(x, y) \tag{36}$$

and the *multiplicative operation* $\otimes$ is the usual addition

$$xy = x \oplus y = x + y \tag{37}$$

The neutral element e with respect to $\otimes$ (the "one" element of the structure) is 0, whereas the neutral element ϵ with respect to $\oplus$ (the "zero" element of the structure) is $-\infty$, which is also the absorbing element of the multiplicative operation: $a \otimes -\infty = -\infty \otimes a = -\infty$, $\forall a \in \mathbb{R}_{\max}$. This dioid is not a ring because, in general, an element of $\mathbb{R}_{\max}$ has no inverse with respect to $\oplus$. One distinctive feature of this structure is the idempotency of the addition:

$$x \oplus x = x, \quad \forall x \in \mathbb{R}_{\max}$$

The *matrix product* $AB = A \otimes B$ of two matrices of fitting sizes $(m \times p)$ and $(p \times n)$ is defined by

$$\begin{aligned} (A \otimes B)_{ij} &= \bigoplus_{k=1}^{p} A_{ik} \otimes B_{kj} = \max_{k=1,\ldots,p}(A_{ik} + B_{kj}), i = 1, \ldots, m; \\ & j = 1, \ldots, n \end{aligned} \tag{38}$$

The *matrix sum* $A \oplus B$ of two matrices of the same size ($m \times n$) is defined by

$$(A \oplus B)_{ij} = A_{ij} \otimes B_{ij} = \max(A_{ij}, B_{ij}), i = 1, \ldots, m; j = 1, \ldots, n \quad (39)$$

The *multiplication* by a scalar a of a matrix A is defined by

$$(a \otimes A)_{ij} = a \otimes A_{ij} = a + A_{ij} \quad (40)$$

With the formalism of the max-plus algebra, the equations of a time event graph become

$$x_i(k+1) = \bigoplus_{j=1}^{n} A_{ij} x_j(k) \quad i = 1, \ldots, n \quad (41)$$

or, in matrix form,

$$\mathbf{x}(k+1) = A\mathbf{x}(k) \quad (42)$$

where $\mathbf{x}(k) = [x_1^{(k)}, \ldots, x_n^{(k)}]^T$ is the state vector at time k, and $A = [A_{ij}, i, j = 1, \ldots, n]$ is the $(n \times n)$ system matrix.

The weighted graph corresponding to a square $(n \times n)$ matrix A is the triple $G(A) = (N, E, \varphi)$, where N is the set of n nodes, E is the set of edges, each representing a nonzero entry of A, and $\varphi: E \to N \times N$, with $\varphi(e_{ij}) = (j, i)$, $e_{ij} \in E$ if and only if $A_{ij} > \epsilon$. The weight of the edge e_{ij} is A_{ij}. In the following, only graphs for which there is at most one edge between any ordered pair of nodes, oriented from the first node to the second, will be considered.

Example 4. The graph in Fig. 9 corresponds to the system matrix

$$A = \begin{bmatrix} \epsilon & 9 & \epsilon \\ 1 & \epsilon & 6 \\ \epsilon & 2 & 3 \end{bmatrix}$$

Considering the state at step k given by the vector $\boldsymbol{x}(k) = [3, 2, 1]^T$, the vector at step $(k + 1)$ is

$$\mathbf{x}(k+1) = A\mathbf{x}(k) = \begin{bmatrix} \epsilon & 9 & \epsilon \\ 1 & \epsilon & 6 \\ \epsilon & 2 & 3 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} (9 \otimes 2) \\ (1 \otimes 3) \oplus (6 \otimes 1) \\ (2 \otimes 2) \oplus (3 \otimes 1) \end{bmatrix} = \begin{bmatrix} 11 \\ 7 \\ 4 \end{bmatrix}$$

A *path* in a graph is a sequence of adjacent edges and nodes: $(i_1, i_2), (i_2, i_3), \ldots, (i_{k-1}, i_k) \equiv i_1 \to i_2 \to \cdots \to i_k$. In general, it is accepted that a path can pass twice through the same node or through the same edge. A *circuit* is a closed path, i.e., a path for which the initial and the final node coincide. In the following, we will consider only elementary circuits, i.e., circuits that do not pass twice through the same node. The length of a path (circuit) is defined as the number of edges in the path (circuit). The weight of a path (circuit) is defined as the $\otimes$ multiplication (i.e., the conventional sum) of the weights of all the edges in the path (circuit): $w(i_1 \to i_2 \to \cdots \to i_k) = A_{i_k i_{k-1}} + \cdots + A_{i_2 i_1}$. The average weight of a path is its weight divided (in the classical way) by its length. For a circuit, the average weight is sometimes called the *circuit mean.*

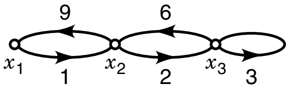

Figure 9. Timed event graph corresponding to the system matrix in Example 4.

Example 5. Examples of paths in the graph in Fig. 9 are

$1 \to 2$ (length = 1, weight = 1, average weight = 1),
$1 \to 2 \to 3$ ($l = 2$, $w = 3$, $aw = 1.5$),
$1 \to 2 \to 3 \to 3 \to 2$ ($l = 4$, $w = 12$, $aw = 3$).

There are three (elementary) circuits in this graph:

$1 \to 2 \to 1$ ($l = 2$, $w = 4$, circuit mean = 2),
$2 \to 1 \to 2$ ($l = 2$, $w = 8$, $cm = 4$),
$3 \to 3$ ($l = 1$, $w = 3$, $cm = 3$).

A graph is strongly connected if there exists a path between any two nodes of the graph. The matrix corresponding to a strongly connected graph is called irreducible. For an irreducible matrix A, then is a permutation P such that $P^T A\ P$ is an upper triangular matrix.

Example 6. The graph in Fig. 9 is strongly connected.

The power of a square matrix A^k is defined recursively by

$$A^k = A \otimes A^{k-1},\ k \in \mathbb{N}^* \quad (43)$$

where $A^0 = I$ is the identity matrix, which has $(A^0)_{ij} = e$ if $i = j$, and $(A^0)_{ij} = \epsilon$ if $i \neq j$. The entry $(A^k)_{ij}$ of the kth power of a square matrix A equals the maximum weight for all the paths of length k from node j to node i.

A square matrix is aperiodic if there exists $k_0 \in \mathbb{N}^*$ such that $(A^k)_{ij} \neq \epsilon$ for all $k \geq k_0$. Aperiodicity implies irreducibility because $(A^k)_{ij} \neq \epsilon$ means that there exists at least one path of length k from node j to node i with weight $(A^k)_{ij}$. The reverse is not true.

Example 7. The matrix A corresponding to the graph in Fig. 9 is aperiodic with $k_0 = 4$.

As in conventional algebra, if for a square matrix A there exist a vector $\mathbf{v} \neq [\epsilon, \epsilon, \ldots, \epsilon]^T$ and a scalar λ such that

$$A \otimes \boldsymbol{v} = \lambda \otimes \boldsymbol{v} \quad (44)$$

then v is called an *eigenvector* of A, and λ is the corresponding *eigenvalue.*

Example 8. It is easy to check that

$$A \begin{bmatrix} 7 \\ 3 \\ e \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \\ 5 \end{bmatrix} = 5 \begin{bmatrix} 7 \\ 3 \\ e \end{bmatrix}$$

where A is the matrix corresponding to the graph in Fig. 9. The vector $\boldsymbol{v} = [7\ 3\ e]^{\mathrm{T}}$ is an eigenvector of A for the eigenvalue $\lambda = 5$.

Some very important properties of the eigenvalues and eigenvectors of irreducible matrices are stated next without proof.

- Every square matrix has at least one eigenvalue.
- The eigenvalue is unique for an irreducible matrix.
- For an irreducible matrix, the eigenvalue equals the maximum circuit mean taken over all circuits in the strongly connected graph corresponding to the matrix.

Any circuit for which the circuit mean is maximum is called a critical circuit.

Example 9. The critical circuit of the graph in Fig. 9 is $1 \to 2 \to 1$, which has the maximum average weight over all circuits of the graph. This weight determines the eigenvalue $\lambda = 5$ of the matrix A.

The matrix A^+ is defined by

$$A^+ = \bigoplus_{k=1}^{\infty} A^k \tag{45}$$

Each entry $(A^+)_{ij}$ of the matrix gives the maximum weight for all paths of arbitrary length from node j to node i. The length increases unboundedly, so that the matrix A^+ diverges. For an irreducible matrix A, with the eigenvalue λ, a matrix A_λ is defined by

$$A_\lambda = \lambda^{-1} A \tag{46}$$

meaning that $(A_\lambda)_{ij} = A_{ij} - \lambda$.

The matrix A_λ has the remarkable property that

$$A_\lambda^+ = \bigoplus_{k=1}^{\infty} A_\lambda^k = \bigoplus_{k=1}^{n} A_\lambda^k \tag{47}$$

where n is the dimension of the square matrix A. As before, $(A_\lambda^+)_{ij}$ is the maximum weight for all paths of arbitrary length from node j to node i, in the directed graph corresponding to A_λ. The critical circuit in this graph has the weight e. A_λ has the same eigenvectors as A, but for the eigenvalue e. For any node j in a critical circuit of A, the jth column of A_λ^+ is an eigenvector of A (and of A_λ).

Example 10. For the matrix A considered earlier, the matrix A^+ diverges, but we can readily calculate A_λ and A_λ^+:

$$A_\lambda = \begin{bmatrix} \epsilon & 4 & \epsilon \\ -4 & \epsilon & 1 \\ \epsilon & -3 & -2 \end{bmatrix}, \qquad A_\lambda^+ = \begin{bmatrix} e & 4 & 5 \\ -4 & e & 1 \\ -7 & -3 & -2 \end{bmatrix}$$

The first two columns of A_λ^+ are eigenvectors of A for the eigenvalue $\lambda = 5$. It happens that the third column is also an eigenvector.

The *asymptotic behavior* of the systems described by irreducible matrices is periodic. Remarkably enough, the steady state is reached within a finite number of steps. The periodic regime is determined only by the length and the average weight of the critical circuit, which is the slowest circuit in the system. If A is irreducible and the corresponding graph has a unique critical circuit of length m and average weight λ (the eigenvalue of A), then A is asymptotically periodic with period m, i.e., there exists a $k_A \in \mathbb{N}^*$ such that

$$A^{k+m} = \lambda^m A^k, \text{ for all } k \geq k_A \tag{48}$$

Example 11. For the matrix A considered in Example 4, the length of the critical path $m = 2$, its average weight (the eigenvalue of A) is $\lambda = 5$, and $k_{\mathbf{A}} = 4$, so that $A^6 = 10 \otimes A^4$. Indeed, in the max-plus algebra $10 = 5 \otimes 5 = 5^2$.

The max-plus algebra can thus be used to evaluate the performance of timed discrete systems, in the asymptotic steady state. For this purpose, the eigenvalue λ is the key parameter of a system described by an irreducible matrix because λ determines the speed in the periodic state. Usually, $1/\lambda$ is referred to as the throughput of the system.

Petri Nets Models

Petri nets theory has been developed as a formalism able to describe in a unified way systems that included computers, programs, and a certain environment. Previously, the various components of such systems had to be described in different and unrelated formalisms: automata theory for the computer hardware, code in a sequential programming language for the program, and narrative prose for the interaction of the program with the environment. From the three mentioned elements, at most one—the program—is sequential so that the capacity to deal with the characteristics of parallel systems was a basic request. The timed Petri nets have been introduced in the late seventeen to quantitatively study the performances of parallel systems, especially referring to (1) concurrence, the possibility that events occur independently; (2) synchronization, the necessity that some events wait for the others before they can occur; and (3) conflicts, the mutual exclusion of some events. Petri nets have the advantage to have a precise semantics and to allow the efficient use of algebraic techniques. The event graphs, which are adequate for modeling collision-free synchronous systems, form a special class of Petri nets and can be described by linear equations when using max-plus algebra. An overview of Petri nets and of the concepts related to their properties can be found in the survey paper of Murata (26).

Untimed Petri Nets. An *untimed Petri net* is defined by (S, M_0), where S describes the structure of the graph attached to the net and M_0 is the initial marking of the net.

The structural part is characterized by the 5-tuple

$$S = (P, T, F, r, s) \tag{49}$$

with P the (finite) set of places and T the (finite) set of transitions. The places P (customarily represented by circles) and the transitions T (drawn as bars) form the vertices of a graph. The arcs of the graph are given by $F \subset P \times T \cup T \times P$. The maps $r: P \times T \to \mathbb{N}^*$ and $s: T \times P \to \mathbb{N}^*$ give the (positive) integer weights of the arcs going from the places toward the transitions, and from the transitions toward the places, re-

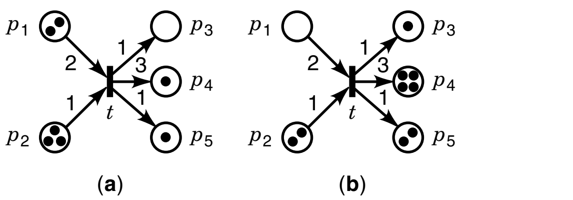

Figure 10. Firing of a transition in a Petri net. (a) Transition t is fireable because for $\forall p \in {}^*t = \{p_1, p_2\}$, the markings exceed the threshold: $M(p_1) = 2 \geq r(p_1, t) = 2$ and $M(p_2) = 2 \geq r(p_2, t) = 1$. (b) After the firing, the markings are $M'(p_1) = M(p_1) - r(p_1, t) = 0$, $M'(p_2) = 1$, $M'(p_3) = M(p_3) + s(t, p_3) = 1$, $M'(p_4) = 4$, $M'(p_5) = 2$.

spectively. It is customary to inscribe only the arcs with the weights exceeding one, whereas the arcs without any inscription have unit weight by default. Sometimes, edges with a larger weight are represented by the corresponding number of unit weight arcs in parallel. The places may contain zero or more *tokens,* usually drawn as black circles. A marking or "state" of a Petri net is given by the distribution of the tokens at a certain moment: $M: P \rightarrow \mathbb{N}$, where $M(p)$ gives the number of tokens in the place $p \in P$. The initial marking is given by M_0.

Given a transition $t \in T$, the input place set of t is defined by

$$ {}^*t = \{p \in P : (p, t) \in F\} \tag{50}$$

and the output place set, by:

$$ t^* = \{p \in P : (t, p) \in F\} \tag{51}$$

Similarly, for a place $p \in P$, the input transition sets of p is:

$$ {}^*p = \{t \in T : (t, p) \in F\} \tag{52}$$

whereas the output transition set is

$$ p^* = \{t \in T : (p, t) \in F\} \tag{53}$$

The dynamics of the Petri net is determined by the marking M. A transition t is enabled on a marking M, if the number of tokens in each place p from which there is an arc toward the transition t exceeds or at least equals the weight of the arc, i.e., if $M(p) \geq r(p, t)$ for all $p \in {}^*t$. An enabled transition may fire. When a transition t fires, the number of tokens in the places $p \in {}^*t \cup t^* \subset P$ changes. The number of tokens is decreased for each input place $p \in {}^*t$ with $r(p, t)$ pieces and increased with each output place $p \in t^*$ with $s(t, p)$ pieces. Consequently, the marking of the network places $p \in P$ changes from $M(p)$ to $M'(p)$, according to the rule

$$ M'(p) = \begin{cases} M(p) - r(p,t), & p \in^* t \\ M(p) + s(t,p), & p \in t^* \\ M(p), & \text{otherwise} \end{cases} \tag{54}$$

Example 12. Figure 10(a) represents a transition for which the firing conditions are fulfilled. Figure 10(b) gives the marking resulted after the fire.

A marking M_2 is reachable from a marking M_1 if a sequence of transition firings leading from M_1 to M_2 exists. The set of markings reachable when starting from a marking M and firing transitions is denoted by $R(M)$. The rechability problem—given M_1 and M_2, establish if $M_2 \in R(M_1)$—is exponentially decidable.

A marking M is bounded if for any place $p \in P$ the number of tokens is bounded, i.e., there is a constant integer $b \in \mathbb{N}^*$ such that $M(p) < b$, $\forall p \in P$. A Petri net is bounded for a given initial marking M_0 if it is uniformly bounded for any $M \in R(M_0)$. A Petri net is safe if the bound is 1. A Petri net is structurally bounded if it is bounded for any initial marking M_0. A Petri net is conservative if the number of tokens is constant during the evolution of the system:

$$ \sum_{p \in P} |M(p)| = \sum_{p \in P} |M_0(p)|, \forall M \in R(M_0) $$

Example 13. The Petri net in Fig. 10 is not conservative.

A transition t in a Petri net is alive for a marking $M \in R(M_0)$, if there exists $M' \in R(M)$ such that t is fireble under M'. A transition is structurally alive if it is alive for any initial marking. A Petri net is (structurally) alive if all its transitions are (structurally) alive.

The incidence matrix of a Petri net is the $|T| \times |P|$ matrix $\boldsymbol{A}$ with the elements

$$ A_{ij} = s(i, j) - r(j, i) \tag{55}$$

The evolution vector $\boldsymbol{u}_k$ at step k is a unipolar binary vector of size $|T|$

$$ \boldsymbol{u}_k = (1, 0, 1, \ldots, 0, 0)^T \tag{56}$$

which has the entries one for the transitions that fire at step k and zero for the others.

The net marking at step k can be described by a vector $\boldsymbol{M}_k$ for which the evolution law is

$$ \boldsymbol{M}_k = \boldsymbol{M}_{k-1} + \boldsymbol{A}^T \boldsymbol{u}_k; \quad k \in \mathbb{N}^* \tag{57}$$

A firing sequence $\{\boldsymbol{u}_k | k = 1, 2, \ldots, d\}$ is globally characterized by the firing vector

$$ \mathbf{x} = \sum_{k=1}^{d} \mathbf{u}_k $$

whereas the final marking is given by

$$ \mathbf{M}_f = \mathbf{M}_0 + \mathbf{A}^T \mathbf{x} \tag{58}$$

where $\mathbf{M}_0$ is the initial marking, and $\mathbf{M}_f$ is the final marking.

Example 14. Untimed Petri nets have been used for the validation of communication protocols. The Petri net in Fig. 11

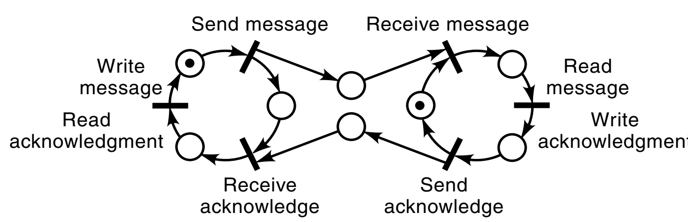

Figure 11. Untimed Petri net model of a communication protocol with acknowledge of reception.

shows such a protocol with acknowledge of reception. The system comprises cycles on the emitting and receiving parts. The position of the tokens gives the state of the system, whereas the actions are represented by the transitions. The sending side waits for confirmation from the receiving part before proceeding to the transmission of the next message. The receiving side is ready for a new message only after having sent out the acknowledgment for the preceding one. The arrival of the next message can then trigger a new cycle for sending out the confirmation.

Timed Petri Nets. Timed Petri nets offer a general formalism adequate for including a measure of time in the description of a DES. Petri nets are especially adequate to model concurrent or parallel discrete systems. A First In–First Out (FIFO) discipline is usually adopted for all the places and all the transitions. Time-related parameters are attached to each process taking place in the net. If the nth token enters a place p at the moment u, it becomes "visible" for the transitions in p^* only after the moment $u + \sigma_p(n)$, where $\sigma_p(n)$ is the rest time of the nth token in place p. An initial latency time is also ascribed to each initial token in a place p. If $M_0(p) \geq n$, the nth token existing in place p at the initial moment becomes available for the transitions in p^* starting from a moment $\xi_p(n)$. The initial latency time is a special case of the rest time and allows modeling the peculiarities of the initial phase, whenever necessary. Similarly, the nth fire of a transition t started at a moment u, ends at moment $u + \varphi_t(n)$, where $\varphi_t(n)$, is the duration of the nth firing of the transition t. The tokens are taken from the input places of the transition t and moved to the output places at the moment $u + \varphi_t(n)$.

The time parameters have to satisfy certain natural restrictions:

- All the rest times and transition durations must be nonnegative $\sigma_p(n) \geq 0$, $\varphi_t(n) \geq 0$ for all $p \in P$, $t \in T$, and $n \in N^*$.
- The initial latency times can be both positive and negative, but they are restricted by the *weak compatibility conditions* that require that for each place p: (1) there exists no transition before the initial moment $t = 0$ so that $M_0(p)$ retains its meaning of initial marking, (2) the initial tokens in a place p are taken by the output transitions in p^* before the tokens supplied to p by the input transitions in $^*p$.

A timed Petri net is thus defined by the n-tuple

$$TPN(S, M_0, \Sigma, \phi, \Xi) \tag{59}$$

where S is the structural part, M_0 is the initial marking, $\Sigma = \{\sigma_p(n);\ n \in N^* | p \in P\}$ is the set of rest times, $\phi = \{\varphi_t(n);\ n \in N^* | t \in T\}$ is the set of transition durations, and $\Xi = \{\xi_p(n);\ n \in N^* | p \in P\}$ is the set of initial latencies.

Equivalent Petri nets having only timed transitions or only timed places can be built, as shown in Fig. 12 (a, b).

The following state variables are defined to describe the time evolution of a Petri net:

- The *schedulers:* $x_t(n)$, $y_t(n)$ is the beginning and the end moments, respectively, of the nth fire of the transition $t \in T$; $v_p(n)$, $w_p(n)$ is the *entering* and the *release moments,* respectively, of the nth token in the place $p \in P$,
- The *counters:* $x^t(u)$, $y^t(u)$ is the number of times the transition $t \in T$ has started and ended, respectively, the fire at moment u; $v_p(u)$, $w_p(u)$ is the number of tokens entering and leaving, respectively, place p at moment u.

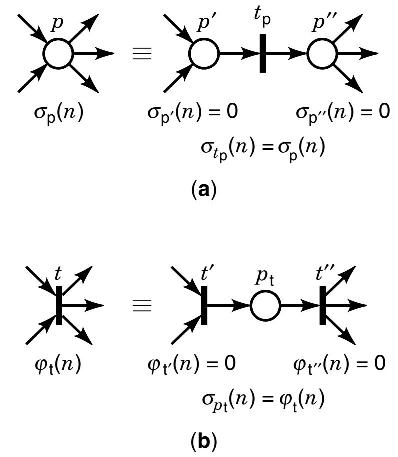

Figure 12. (a) Petri net comprising only timed transitions where the rest time of place p has been assigned as the duration of the equivalent transition t_p. (b) Dual case of a net comprising only timed places where transition t has been replaced with place p_t.

The following conventions are commonly accepted:

- $x_t(0) = y_t(0) = v_p(0) = w_p(0) = -\infty$,
- $x_t(n) = y_t(n) = v_p(n) = w_p(n) = \infty$, if the transition t never fires n times, or the place p never receives n tokens,
- $x^t(u) = y^t(u) = w^p(u) = 0$ and $v^p(u) = M_0(p)$ for $u < 0$.

For any transition $t \in T$, where $n \in N^*$

$$y_t(n) = x_t(n) + \varphi_t(n) \tag{60}$$

The FIFO rule requires

$$w_p(n) \geq v_p(n) + \sigma_p(n) \qquad \text{for } \forall p \in P, \forall n \in N^* \tag{61}$$

meaning that the order of the tokens are not changed at any of the places, and

$$y^t[y_t(n)] = x^t[x_t(n)] \qquad \text{for } \forall t \in T, \forall n \in N^* \tag{62}$$

meaning that a transition cannot start its $(n + 1)$th fire before ending the nth one.

A Petri net is called FIFO if all its places and transitions observe the FIFO discipline. Usually, the stronger conditions of constant rest times and constant transition durations are used. The FIFO constrained can result from the structure of network, without any hypothesis on the net temporizations.

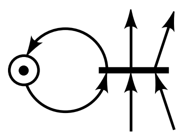

Figure 13. Cyclic transition with structurally restricted FIFO behavior.

Example 15. The Petri net in Fig. 13 contains a cyclic transition which behaves FIFO for any sequencing of the firing.

Timed Petri nets can be used for quantitative performance evaluation, e.g., when studying various queuing types. Most classical networks like Jackson single classes, fork-join queues, and token rings can be modeled with Petri nets, whereas others like multiclass networks, Kelly networks, and processor-sharing systems cannot.

Example 16. Figure 14 represents the Petri net models of some classic types of queues. The Kendall notation is used to describe a queue. The simplest queue, with any input process (.), any distribution of the timings of the server (.), one server (1) and an unlimited buffer (∞) is designated by ./././1/∞.

Petri nets allow a unified treatment of a large class of systems, avoiding the usual case-by-case performance evaluation. It has been shown that Petri nets with inhibitor edges (i.e., with a special kind of edges from places to transitions, which trigger the transitions only when the place is empty) have the computing power of a Turing machine.

The Petri nets can be characterized both by basic *qualitative properties* like stability, existence of a stationary state, and the duration of the transient state and by *performance*

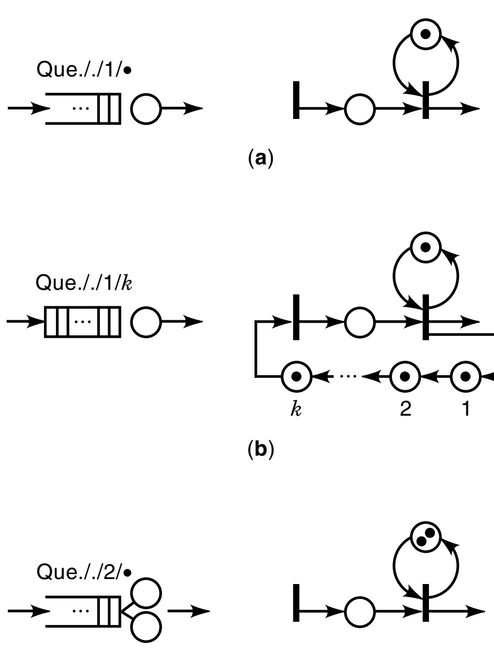

Figure 14. Queue theory and Petri net models of some classic types of queues: (a) Infinite buffer, single server; (b) Finite buffer, single server; (c) Infinite buffer, double server.

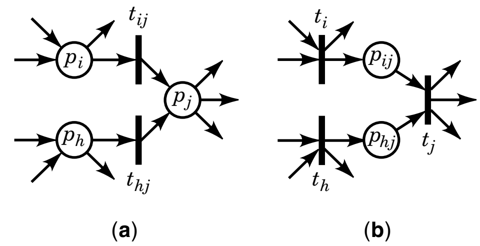

Figure 15. Special cases of Petri nets: (a) model of a state machine, (b) model of an event graph.

parameters like throughput of a transition or average number of tokens in a place. Petri nets include as special cases other frequently used models like state machines, event graphs, and free-choice nets. The following structural conditions define the mentioned special cases:

- A state machine is a Petri net for which

$$|^*t| = |t^*| = 1;\ \forall t \in T \tag{63}$$

  i.e., each transition has exactly one input place and one output place. As a consequence, between any two places p_i and p_j there is at most one transition that would be denoted by t_{ij}, with $\{p_i\} = {}^*t_{ij}$, $\{p_j\} = t_{ij}{}^*$, $\{t_{ij}\} = p_i{}^* \cap {}^*p_j$, as shown in Fig. 15(a).
- An *event graph* is a Petri net with

$$|^*p| = |p^*| = 1;\ \forall p \in P \tag{64}$$

  i.e., each place has exactly one input transition and one output transition. Correspondingly, between any two transitions t_i and t_j, there is at most one place p_{ij}, with $\{t_i\} = {}^*p_{ij}$, $\{t_j\} = p_{ij}{}^*$, $\{p_{ij}\} = t_i{}^* \cap {}^*t_j$, as shown in Fig. 15(b).
- A *free-choice net* is a Petri net for which

$$\forall p \in P, |p^*| > 1 \Rightarrow \forall t \in p^*, |^*t| = 1 \tag{65}$$

  meaning that if a place p has more than one output transition, than the place p is the only input place for each of its output transitions. It results that a free-choice graph contains substructures of the type shown in Fig. 16, so it can model both synchronization [Fig. 16(a)] and choice [Fig. 16(b)], but not both of them for the same process. Free-choice machines include the state machines and the event graphs, again as special cases. The event graphs model only synchronization; they exclude choice. It has

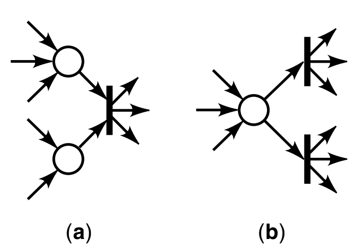

Figure 16. Special cases of Petri nets—the free-choice nets: (a) substructures modeling synchronization, (b) substructures modeling choice.

been shown than an event graph is alive if each circuit in the graph contains at least one token. In the opposite case, the net will run into a dead lock after a finite number of firing instances. In a timed event graph, a place containing k tokens can be replaced by k chained places, each one containing exactly one token, interlaced with $k - 1$ transitions (Fig. 17). The rest time σ_p of the initial place is attributed to one of the places in the chain, all the other places and transitions having no delays.

Timed event graphs can be represented as linear systems by using max-plus algebra. Because of the special structure of a timed event graph, it is convenient to make the analysis in terms of the transitions. Let us denote by $x_i(n)$ the start moment of the nth firing instance of the transition t_i, $i = 1, \ldots, k$; $k = |T|$, and by $\bullet t_i$ the set of the input transitions of t_i:

$$\bullet t_i = {}^*({}^*t_i) = \{t_j | t_j \subset {}^*p, \forall p \in {}^*t_i\} \subset T \tag{66}$$

Consider the nth firing of a transition $t_i \in \bullet t_i$. Using the equivalence in Fig. 16, the place $p_{ji} \in P$ contains at most one token. If $M(p_{ji}) = 0$, then the token enables the nth firing of t_i; else if $M(p_{ii}) = 1$, it enables the $(n + 1)$th firing of t_i. This results in the equation

$$x_j(n+1) > \max_{j \in \bullet t_i}\{x_j[n+1-M(p_{ji})] + \varphi_{t_i} + \sigma_{p_{ji}}\} \tag{67}$$

where $x = x_j[n + 1 - M(p_{ji})]$ is the start moment of the $[n + 1 - M(p_{ji})]$th firing of the transition t_j, $x + \varphi_{t_j}$ is the end moment of this process, and $x_i + \varphi_{t_j} + \sigma_{p_{ji}}$ is the moment the transition t_i is enabled by t_j.

With the delay matrices A_α, $\alpha = 0,1$, defined by

$$(A_\alpha)_{ij} = \begin{cases} \varphi_{t_i} + \sigma_{p_{ji}}, & \text{if } t_i \in \bullet t_i \text{ and } M(p_{ji}) = \alpha \\ \epsilon = -\infty, & \text{otherwise} \end{cases} \tag{68}$$

Eq. (67) can be written in the matrix form

$$x(n+1) \geq A_0 x(n+1) \oplus A_1 x(n) \tag{69}$$

With

$$A_0^* = \bigoplus_{i=0}^{\infty} A^i = \bigoplus_{i=0}^{k} A^i = I + A_0^+ \tag{70}$$

[see Eq. (45)],

$$A_0^*(\mathbf{I} - A_0) = A_0^*(\mathbf{I} - A_0) = I \tag{71}$$

results. Relation (69) becomes

$$x(n+1) \geq A_0^* A_1 x(n) \tag{72}$$

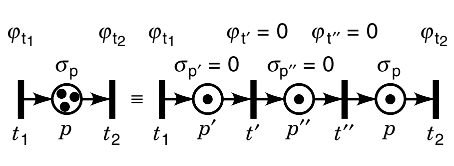

Figure 17. Equivalence of a place containing k tokens with k chained places each one containing exactly one token.

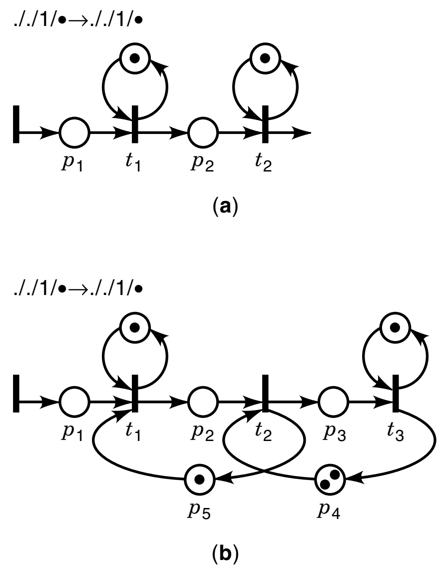

Figure 18. Chained queues: (a) no deadlocks, (b) after-service deadlock.

The minimal solution of (71) is given by the linear recurrence relation

$$x(n+1) = A_0^* A_1 x(n) \tag{73}$$

Using the max-plus algebra framework, the equations of a timed event graph become linear. As shown at Eq. (48), a relation in the form of Eq. (73) determines a periodic stationary solution. This means that event graphs have a cyclicity property: after n firings of each transition, the marking returns exactly to the initial marking. However, this is only a formal result valid in the firing event ordering scale, not in the time scale. The nth firing for different transitions occurs at different time moments $x_{t_i}(n)$ so that there exists no time period which after the marking is repeated.

Example 17. Figure 18 presents two examples of chained queues and their corresponding Petri net (event graph) models. The systems contain each of two servers preceded by queues. The example in Fig. 18(a), for which both queues have infinite buffers, has no deadlocks. The example in Fig. 18(b), exhibits an after-service deadlock. A client leaving the first queue when the buffer of the second queue is full must wait in place p_2; consequently, the access of a new client to the first service is denied.

CONTROL OF DISCRETE EVENT SYSTEMS

One major goal in studying DESs has been to devise methods for controlling the trajectory of a system so as to reach a certain set of desired states, or to avoid some undesired states—including deadlocks or traps. As pointed out in the work of Ramadge and Wonham (16–18), DESs fully qualify as objects for the control theory because they exhibit the fundamental features of potentially controllable dynamic systems. Actually, a large part of the work performed in the DES domain has been motivated by the search for proper techniques to control event sequences and to select the ones that comply with various restrictions or optimization criteria. In the following, we will explore the basics of DESs control within the framework of state machines and formal languages, as initiated by Ramadge and Wonham. The events are considered spontaneous and process-generated. The control consists of forbidding the occurrence of some of the events so as to restrict the behavior of a system to avoid undesirable trajectories. Automatic control is performed by means of another system, which tests the controlled system and acts upon it according to the available information. Thus, the set of events Σ can be partitioned into two disjoint subsets: Σ_u, containing the uncontrollable events, and Σ_c, containing the controllable ones. The control is provided by a supervisor or a discrete event controller (DEC), which has the ability to influence the evolution of the system by enabling and disabling the controllable events, i.e., by allowing or prohibiting their occurrence, so as to perform a certain control task. Various control tasks can be defined: (1) control invariance requires that a specified predicate remains invariantly satisfied whenever initially satisfied, meaning that the behavior of the system remains confined within specified bounds, (2) region avoidance requires that the system does not satisfy undesirable predicates when traversing the state space, and (3) convergence requires that the system to evolve toward a specified target predicate from given initial conditions.

The main difficulty in modeling complex processes by considering all the states and all the events is the combinatorial explosion in the number of their states. A way to keep the complexity manageable is to use event internalization, or partial observation, which leads to nondeterministic process behavior. Markov chain representation, or GSMP models, can be used to describe complex DESs in a formalism that has the capability to relax the requirement that all states and all event sequences be explicitly in the model. Other approaches to achieve an effective modeling are based on the concept of modularity and hierarchy that lead to structured models of lower complexity in comparison with the case when all individual components are taken directly into account.

Controllability and Reachability

Consider a DES modeled by the *generator* $G = (Q, \Sigma, \Delta, s, Q_m)$, where Q is the state space (an arbitrary set), Σ is the event set (or the alphabet, a finite set), Δ is the evolution law [a relation on $Q \times \Sigma \times Q$, which generalizes the transition function, see comments on Eq. (26)], $s = q_0$ is the start (initial) state, and $Q_m \subset Q$ is the set of marker states. As mentioned before, the marker states were introduced by Ramadge and Wonham to identify the "completed tasks." The set of events Σ is partitioned into Σ_c, the set of controllable events, and Σ_u, the set of uncontrollable events, with $\Sigma = \Sigma_c \cup \Sigma_u$, $\Sigma_c \cap \Sigma_u = \emptyset$.

A state $q \in Q$ is called reachable from the initial state $s = q_0$, if there exists a path $(q_0\sigma_1 q_1\sigma_2 \ldots \sigma_n q_n) \in B(G)$, such that $q_n = q$, i.e., if there exists $w = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma^*$, such that $(q_0, w, q_n) \in \Delta^*$.

A state $q \in Q$ is called controllable if there exists $w \in \Sigma^*$ and $q_m \in Q_m$, such that $(q, w, q_m) \in \Delta^*$.

Correspondingly, a generator is called reachable (controllable) if all the states $q \in Q$ are reachable (controllable).

A generator is called trim if it is both reachable and controllable.

A generator is called deterministic [see Eq. (18)] if for all $q \in Q$ and $\sigma \in \Sigma$, there exist at most one state $q' \in Q$ such that $(q, \sigma, q') \in \Delta^*$. In this case, a transition (partial) function can be defined such that $q' = \delta(q, \sigma)$, as shown at Eq. (1) and discussed at Eq. (26).

The control of a DES described by a generator G is provided through a control pattern $\gamma : \Sigma \to \{0, 1\}$, defined such that for a state $\sigma \in \Sigma_c$, $\gamma(\sigma) = 1$ if σ is enabled and $\gamma(\sigma) = 0$ if σ is disabled. For all $\sigma \in \Sigma_u$, $\gamma(\sigma) = 1$ as these events can not be disabled. The set of control patterns γ is denoted by $\Gamma \subset \{0, 1\}^{\Sigma}$.

For each control pattern, a new generator $G(\gamma) = (Q, \Sigma, \Delta^{\gamma}, s, Q_m)$ is obtained, where the controlled evolution relation Δ^{γ} is defined by

$$\forall q, q' \in Q, \forall \sigma \in \Sigma : \quad (q, \sigma, q') \in \Delta^{\gamma} \Leftrightarrow (q, \sigma, q') \in \Delta \text{ and } \gamma(\sigma) = 1 \tag{74}$$

The set of enabled events, also called the control input, for a control pattern γ is given by

$$\Sigma^e(\gamma) = \{\sigma \in \Sigma | \gamma(\sigma) = 1\} = \Sigma_c^e(\gamma) \cup \Sigma_u \tag{75}$$

where the control pattern γ plays the role of the characteristic function of the set.

As mentioned earlier, $\Sigma_u \subset \Sigma^e(\gamma)$, for any control pattern γ.

The set of feasible events for a state $q \in Q$ of the generator $G(\gamma)$ is given by $\Sigma^f(q) \cap \Sigma^e(\gamma)$.

The set of all control inputs is

$$\Sigma^e(\Gamma) = \{\Sigma^e(\gamma) | \gamma \in \Gamma\} \subseteq 2^{\Sigma} \tag{76}$$

The control of G through Γ consists in choosing a specific γ when the system is in a certain state $q \in Q$, after a certain sequence of events $w \in L$, according to the assumed controlling task.

The choice of a particular control pattern $\gamma \in \Gamma$ can be considered itself an event, so that a controlled discrete event system (CDES) with the generator

$$G(\Gamma) = (Q, \Sigma \times \Gamma, \Delta^{\Gamma}, s, Q_m) \tag{77}$$

can be defined where the evolution law given by

$$[q, (\sigma, \gamma), q] \in \Delta^{\Gamma} \Leftrightarrow (q, \sigma, q') \in \Delta^{\gamma} \tag{78}$$

Example 18. For the single model of a machine shown in Fig. 2, the control could consist of honoring or turning down requests to start a new task and passing from idle (I) to working state (W), taking into account the history of machine evolution.

The set Γ consist of two control patterns, namely γ_0, which disables the requests

$$\gamma_0(S) = 0, \gamma_0(C) = \gamma_0(B) = \gamma_0(R) = 1$$

and γ_1, which enables the requests

$$\gamma_1(S) = \gamma_1(C) = \gamma_1(B) = \gamma_1(R) = 1$$

For the system in Fig. 4 comprising two simple machines, the control of one of the machines can be made dependent of the state of the other (e.g., the second machine accepts requests only if the first one is down).

Supervision

In the standard control terminology, the generator G plays the role of the plant, the object to be controlled. The agent doing the controlling action will be called the supervisor. Formally, a supervisor over Γ is a pair

$$S = (T, \varphi) \tag{79}$$

where T is a reachable deterministic generator $T = (Q', \Sigma, \Delta, s'_0, Q'_m)$ and φ: $Q' \rightarrow \Gamma$ is the map that specifies, for each state $q' \in Q'$ reached by the generator of the supervisor, what control pattern $\gamma = \varphi(q')$ must be applied to $G(\Gamma)$.

If the behavior of $G(\Gamma)$ is used to determine the state of T, a *supervised generator* results

$$(G, S) = [Q \times Q', \Sigma, \Delta_{G,S}, (s_0, s'_0), Q_m \times Q'_m] \tag{80}$$

where

$$[(q_1, q'_1), \sigma, (q_2, q'_2)] \in \Delta_{G,S}$$
$$\Updownarrow \tag{81}$$
$$(q_1, \sigma, q_2) \in \Delta \text{ and } (q'_1, \sigma, q'_2) \in \Delta' \text{ and } \gamma(\sigma) = [\varphi(q_2)](\sigma) = 1$$

The supervisor has authority only over controllable events. The uncontrollable events $\Sigma^f(q) \cap \Sigma_u$ that may occur in a state q of the plant are called disturbances (disturbing events). Again, in standard control theory terminology T is the observer, while φ implements the feedback, so that the supervised generator operates in closed loop. Various algorithms are given in the literature for the synthesis of supervisors able to achieve different control tasks for deterministic or stochastic DESs.

The supervisor implements a map f: $L(G) \rightarrow \Gamma_e$ specifying for each observed string of events $w \in L(G)$ the control input $\Sigma^e(\gamma) = f(w)$ that must be applied to G. When designing a supervisor, the objective is to obtain a CDES that obeys the control constraints imposed by the considered control task. This means suppressing the undesirable sequences of events, while restricting as little as possible the overall freedom of the system.

The behavior of the supervised generator is described by the language $L(G, f)$ defined by $\epsilon \in L(G, f)$, $w\sigma \in L(G, f)$, if and only if $w \in L(G, f)$, $\sigma \in f(w)$ and $w\sigma \in L(G)$.

The marked language controlled by f in G is $L_m(G, f) = L_m(G) \cap L(G, f)$, i.e., the part of the original marked language that is allowed under the supervision. If Q_m represents completed tasks, the language $L_m(G, f)$ indicates the tasks that will be completed under supervision.

The supervisor S can also be modeled as another DES whose transition structure describes the control action on G. The following requirements have to be satisfied:

- If $s \in L(G, f)$ then $s \in L(S)$, and $s\sigma \in L(S)$ only if $\sigma \in f(s)$. This condition ensures that the transitions disabled by the control are not included in the transition structure of S.
- If $s \in L(G, f)$, $s\sigma \in L(G)$ and $\sigma \in f(s)$, then $s\sigma \in L(S)$. This condition ensures that a transition possible in G and allowed by the control is included in the transitive structure of S.

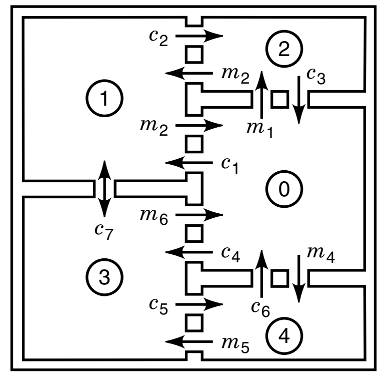

Figure 19. The cat-and-mouse maze. The cat starts from room 2; the mouse starts from room 4. The cat and the mouse each use only the passages labeled c and m, respectively. Control the system by (minimally) forbidding some of the passages (except c_7), to prevent the dangerous encounter of the parties.

An event σ can occur in $G \times S$ and produce the transition $(q, x) \rightarrow (q', x')$, only if σ is possible in both G and S, and produces the transitions $q \rightarrow q'$ and $x \rightarrow x'$. This form of supervision can be obtained from the state realization (S, φ) by trimming the transition structure of S (16).

Consider a DES for which the unsupervised (open loop) behavior is given by a language L. One of the key issues is to specify the properties of a sublanguage $K \subseteq L$ that is achievable under supervision. Because the uncontrollable events continue to occur even for the closed loop (supervised) system, the prefix closure $\overline{K}$ of such a controlled language K has to be invariant under the perturbation of the uncontrollable events. On the other hand, as K is a restriction of L, not any words in Σ^* containing uncontrollable events can occur, but only those that are also generated in the open loop conditions (i.e., that belong to L). It results that every word that belongs to L and is composed by a prefix string $w \in \overline{K}$, followed by an uncontrollable event $\sigma \in \Sigma_u$ (i.e., every word of the form $w\sigma \in L$), must also be a prefix string of K, i.e., $w\sigma \in \overline{K}$.

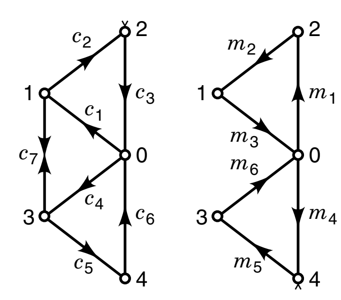

Figure 20. Generator models for the cat and for the mouse moving independently in the maze of Fig. 19.

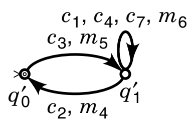

Figure 21. The generator of the supervisor for the cat-and-mouse problem.

Thus, a language $K \subseteq L \subseteq \Sigma^*$ is called controllable if

$$\overline{K}\Sigma_u \cap L = \overline{K} \tag{82}$$

Consider now a nonblocking DES with the behavior $L(G)$ and the marked behavior $L_m(G)$. For any nonempty $K \subseteq L$, there exists a supervisor f such that $L_f = K$ if and only if K is a prefix closed and controllable language. Similarly, for any nonempty $K \in L_m$, there exists a supervision f such that $L_{mf} = K$ and the closed loop behavior is not blocking if and only if K is controllable and L_m is closed (i.e., $\overline{K} \cap L_m = K$).

Thus it is possible to find a supervisor f so that $L_f = K$ when K is prefix closed and controllable. The proof of this proposition (18) provides an algorithm for constructing the state realization (S, φ) of the supervisor f from a generator of the controllable language K. For an arbitrary $K \subseteq \Sigma^*$, the family of controllable sublanguages of K is nonempty and closed under the set union and has a unique supremal element $K^{\dagger}$ under the partial order of subset inclusion. This supremal sublanguage (which can be the empty language) provides an optimal approximation of K by preserving the restrictions imposed by K, but requiring a minimally restrictive control. Denote by $P(\Sigma^*)$ the set of all languages over Σ^* (the power set of Σ^*), and define $\Omega : P(\Sigma^*) \to P(\Sigma^*)$ by

$$\Omega(J) = K \cap \sup[T : T \subseteq \Sigma^*, T = \overline{T}, T\Sigma_u \cap L = \overline{J}] \tag{83}$$

The supremal sublanguage $K^{\dagger}$ is the largest fixpoint of Ω, i.e., the largest language satisfying $\Omega(J) = J$. The iterations

$$K_{j+1} = \Omega(K_j), \quad j = 0, 1, 2, \ldots, \text{with} K_0 = K \tag{84}$$

converge to $K^{\dagger}$ after at most mn steps, where m and n are the number of states of the generators of L and K, respectively.

Example 19. Consider the famous cat-and-mouse maze (Fig. 19) introduced by Ramadge and Wonham (16), and used as a typical example of untimed DES control ever since (e.g., see the attractive Ref. 15). The cat uses only the doors labeled $c_1, \ldots, c_7$, whereas the mouse uses only those labeled $m_1, \ldots, m_6$. The generator models for the cat and the mouse are shown in Fig. 20. The state i for either of them corresponds to the room it occupies, whereas the events correspond to the transitions $i \to j$ from one room to another. We assume that door c_7 is uncontrollable, $\Sigma_u = \{c_7\}$, whereas all the other doors can be opened or closed to control the movement of the cat and the mouse. As shown earlier (see Figs. 3 and 4 at Example 2), the joint generator model when composing the generators of two subsystems has the state set $Q = Q_1 \times Q_2$, and the event set $\Sigma = \Sigma_1 \cup \Sigma_2$. The problem is to find the control scheme that leaves the greatest freedom of movement to both parties but that ensures that they (1) never occupy the same room simultaneously and (2) can always return to their initial state, i.e., the cat in room 2 and the mouse in room 4. The first condition forbids the states (i, i), while the second sets the marker state set $Q_m = \{(2, 4)\}$. To build the generator of the controlled system, i.e., of the system obeying the constraints, the following pruning steps are performed on the composed generator model for both the cat and the mouse:

1. Delete the forbidden states $\{(i, i) | i = 0, 1, \ldots, 4\}$, that correspond to the cat and the mouse being in the same room.
2. Eliminate the edges of the composed graph ending in the forbidden states, i.e.,

$$\begin{aligned}&(2,0) \overset{c_3}{\to} (0,0), (4,0) \overset{c_6}{\to} (0,0), (0,1) \overset{m_3}{\to} (0,0), (0,3)\\ &\overset{m_6}{\to} (0,0), (0,1) \overset{c_1}{\to} (1,1)(3,1) \overset{c_7}{\to} (1,1), (1,2)\\ &\overset{m_2}{\to} (1,1), (1,2) \overset{c_2}{\to} (2,2), (2,0) \overset{m_1}{\to} (2,2), (0,3)\\ &\overset{c_4}{\to} (3,3), (1,3) \overset{c_7}{\to} (3,3), (3,4) \overset{m_5}{\to} (3,3), (3,4)\\ &\overset{c_5}{\to} (4,4), (4,0) \overset{m_4}{\to} (4,4)\end{aligned}$$

3. Discard the states reachable only from the previously deleted states, i.e., the states (4, 3) and (2, 1).
4. Remove the states for which the output edges correspond to uncontrollable events ($\Sigma_u = \{c_7\}$) and lead to previously deleted states, i.e., the states (1, 3) and (3, 1).
5. From the resulting graph retain only the trim part, containing the reachable and controllable states.

The supervisor can be further simplified by an aggregation of technique. The result is a supervisor $S = (T, \varphi)$, where T is given in Fig. 21, and the map φ is given in Table 2. The state set Q' of T is made up of only two states q'_0, q'_1. In the initial state q'_0—when the cat is in room 2 and the mouse in room 4—all the transitions are enabled; in the state q'_1—when one of the parties has left its initial room—the set of transitions c_3, c_5, m_1, and m_5 are disabled. This actually isolates either the mouse in room 4 (closing c_5 and m_5) when the cat is out of room 2 or the cat in room 2 (closing c_3 and m_1) when the mouse is out of room 4. It can be noticed that transitions c_5, c_6, m_1, m_2, m_3 can no longer occur for the controlled system, being either directly forbidden, or impossible because of the restrictions.

Table 2. Mapping of Supervisor States to Control Patterns for the Cat-and-Mouse Maze Example

σ	c_1	c_2	c_3	c_4	c_5	c_6	c_7	m_1	m_2	m_3	m_4	m_5	m_6
$\varphi(q'_0) = \gamma_0$	1	1	1	1	1	1	1	1	1	1	1	1	1
$\varphi(q'_1) = \gamma_1$	1	1	0	1	0	1	1	0	1	1	1	0	1

BIBLIOGRAPHY

1. R. D. Brandt et al., Formulas for calculating supremal and normal sublanguages, *Syst. Control Lett.,* **15** (8): 111–117, 1990.
2. R. Kumar and V. K. Garg, *Modeling and Control of Logical Discrete Event Systems,* Norwell, MA: Kluwer, 1995.
3. R. Kumar, V. K. Garg, and S. I. Marcus, On supervisory control of sequential behaviors, *IEEE Trans. Autom. Control,* **37** (12): 1978–1985, 1992.
4. R. Kumar, V. Garg, and S. I. Marcus, Predicates and predicate transformers for supervisory control of discrete event dynamic systems, *IEEE Trans. Autom. Control,* **38** (2): 232–247, 1993.
5. R. Kumar, V. K. Garg, and S. I. Marcus, Language stability and stabilizability of discrete event systems, *SIAM J. Control Optim.,* **31** (5): 1294–1320, 1993.
6. Y. Li, Robust and adaptive supervisory control of discrete-event systems, *Proc. Amer. Control Conf.,* Chicago, IL, 1992.
7. Y.-H. Li, Optimal control of fair discrete-event systems, *Appl. Math. Comput. Sci.,* **6**: 803–814, 1996.
8. Y. Li and W. M. Wonham, Controllability and observability in the state-feedback control of discrete-event systems, *Proc. 27th Conf. Decision Control,* Austin, TX, 1988, pp. 203–208.
9. Y. Li and W. M. Wonham, Decentralized control and coordination of discrete-event systems with partial observation, *IEEE Trans. Autom. Control,* **35**: 1330–1337, 1990.
10. Y. Li and W. M. Wonham, Control of vector discrete event systems I—The base model, *IEEE Trans. Autom. Control,* **38** (8): 1214–1227, 1993.
11. Y. Li and W. M. Wonham, Control of vector discrete event systems II—Controller synthesis, *IEEE Trans. Autom. Control,* **39** (3): 512–531, 1994.
12. Y. Li and W. M. Wonham, Concurrent vector discrete event systems, *IEEE Trans. Autom. Control,* **40** (4): 628–638, 1995.
13. F. Lin and W. M. Wonham, On observability of discrete event systems, *Inf. Sci.,* **44**: 173–198, 1988.
14. F. Lin and W. M. Wonham, Decentralized supervisory control of discrete-event systems, *Inf. Sci.,* **44**: 199–224, 1988.
15. C. Praagman, Discrete event systems, Description by formal languages, Res. Memo., 505, University of Gröningen, The Netherlands, 1992.
16. P. J. Ramadge and W. M. Wonham, Supervisory control of a class of discrete-event processes, *SIAM J. Control Optim.,* **25**: 206–230, 1987.
17. P. J. Ramadge and W. M. Wonham, Modular feedback logic for discrete event systems, *SIAM J. Contr. Optim.,* **25**: 1202–1218, 1987.
18. P. J. G. Ramadge and W. M. Wonham, The control of discrete event systems, *Proc. IEEE,* **77**: 81–98, 1989.
19. P. J. Ramadge, Some tractable problems in the supervisory control of discrete event systems described by Büchi automata, *IEEE Trans. Autom. Control.,* **AC-34**: 10–19, 1989.
20. F. Baccelli and M. Canales, Parallel simulation of stochastic Petri nets using recurrence equations, *ACN Trans. Modeling Comput. Simulation,* **3** (1): 20–41, 1993.
21. F. Baccelli et al., *Synchronization and Linearity: An Algebra for Discrete Event Systems,* New York: Wiley, 1992.
22. F. Baccelli, N. Furmento, and B. Gaujal, Parallel and distributed simulation of free choice Petri nets, *PADS,* Lake Placid, NY, 1995, pp. 3–10.
23. F. Baccelli and B. Gaujal, Liveness in free-choice Petri nets—An algebraic approach, *INRIA Sophia—Antipolis,* Rapport de recherche no. 2839, 1996, pp. 39.
24. K. Hiraishi and M. Nakano, On symbolic model checking in Petri nets, *IEICE Trans. Fundamentals of Electron. Commun. Comput. Sci.,* **E78-A**: 1479–1486, 1995.
25. L. E. Holloway and B. H. Krogh, Synthesis of feedback control logic for a class of controlled Petri nets, *IEEE Trans. Autom. Control,* **35**: 514–523, 1990.
26. T. Murata, Petri nets: Properties, analysis and applications, *Proc. IEEE,* **77**: 541–580, 1989.
27. A. Ohta and T. Hisamura, On some analysis properties of Petri net systems under the earliest firing rule, *IEICE Trans. Fundamentals Electron. Commun. Comput. Sci.,* **E79-A**: 1791–1796, 1996.
28. S. Takai, S. Kusumoto, and S. Kodama, Modular control of Petri nets under partial observation, *Trans. Inst. Syst. Control Inf. Eng.,* **9**: 598–605, 1996.
29. X. Xie, Dynamics and convergence rate of ordinal comparison of stochastic discrete-event systems, *IEEE Trans. Autom. Control,* **42** (4): 580–590, 1997.
30. J. G. Braker, *Algorithms and applications in timed discrete event systems,* Ph.D. Thesis, Technische Universiteit Delft, The Netherlands, 1993.
31. B. A. Brandin and W. M. Wonham, Supervisory control of timed discrete-event systems, *IEEE Trans. Autom. Control,* **39** (2): 329–342, 1994.
32. P. J. Haas and G. S. Shedler, Recurrence and regenerative in non-Markovian networks of queues, *Stochastic Mode,* **3**: 29–52, 1987.
33. H. Shimohata, S. Kataoka, and T. Yamada, A resource allocation problem on timed marked graphs: A decomposition approach, *Int. J. Syst. Sci.,* **27**: 405–411, 1996.
34. C. G. Cassandras and S. G. Strickland, Observable augmented systems for sensitivity analysis of Markov processes: The predictability of discrete-event-systems, *IEEE Trans. Autom. Control,* **34** (10): 1026–1037, 1989.
35. C. Glasserman and P. Vakili, Comparing Markov chains simulated in parallel, *Probability Eng. Infor. Sci.,* **8** (3): 309–326, 1994.
36. C. Cassandras and S. G. Strickland, Sample path properties of timed discrete event systems, *Proc. IEEE,* **77** (1): 59–71, 1989.
37. P. W. Glynn, A GSMP formalism for discrete event systems, *Proc. IEEE,* **77** (1): 14–23, 1989.
38. C. Cassandras and Y. C. Ho, An event domain formalism for sample path perturbation analysis of discrete event dynamic systems, *IEEE Trans. Autom. Control,* **AC-32**: 858–866, 1987.
39. Y. C. Ho, Performance evaluation and perturbation analysis of discrete event dynamic systems, *IEEE Trans. Autom. Control,* **AC-32** (7): 563–572, 1987.
40. Y. C. Ho, Special issue on dynamics of discrete event systems, *Proc. IEEE,* **77** (1): 1–232, 1989.
41. R. Y. Rubinstein and A. Shapiro, *Discrete Event Systems—Sensitivity Analysis and Stochastic Optimization by Score Function Method,* New York: Wiley, 1993.
42. S. G. Strickland and C. G. Cassandras, Sensitivity analysis of discrete-event systems with non-Markovian event processes, *Proc. 29th IEEE Conf. Decision Control,* 111–116, 1989.
43. J. Banks, Output analysis capabilities of simulation software, *Simulation,* **66**: 23–30, 1996.
44. W. T. Chang, S. Ha, and E. A. Lee, Heterogenous simulation—Mixing discrete-event models with dataflow (invited paper for RASSP special issue), *J. VLSI Signal Processing,* **13**: 1–25, 1997.
45. S. Ghosh, E. Debenedictis, and Yu Meng-Lin, YADDES: A novel algorithm for deadlock-free distributed discrete-event simulation, *Int. J. Comput. Simulation,* **5**: 43–83, 1995.

46. D. L. Kettenis, An algorithm for parallel combined continuous and discrete-event simulation, *Simulation Practice and Theory,* **5**: 167–184, 1997.
47. J. Lin, A discrete event simulation support system in C/sup ++/, *J. Syst. Sci. Syst. Eng.,* **5**: 456–464, 1996.
48. L. Mallet and P. Mussi, Object oriented parallel discrete event simulation: The prosit approach, *Modeling and Simulation,* Lyon, France, June 1993; also in INRIA Res. Rept. 2232.
49. P. Mussi and G. Siegel, Sequential simulation in PROSIT: Programming model and implementation, INRIA Tech. Rept. RR-2713; *Eur. Simulation Symp.,* Erlangen, Germany, 1995, pp. 297–301.
50. T. Nakanishi, A discrete system simulation language SIM-SCRIPT II.5, *Joho Shori,* **37**: 226–223, 1996.
51. N. T. Patsis, C. H. Chen, and M. E. Larson, SIMD parallel discrete-event dynamic system simulation, *IEEE Trans. Control Syst. Technol.,* **5** (1): 30–40, 1997.
52. S. Taylor, N. Kalantry, and S. C. Winter, The parallelization of discrete event simulation: A methodology, *IEE Colloquium: Increased Production Through Discrete Event Simulation,* London, 1993.
53. E. G. Ulrich, V. D. Agraval, and J. H. Arabian, *Concurrent and Comparative Discrete Event Simulation,* Norwell, MA: Kluwer (Academic), 1994.
54. P. Vakili, L. Mollamustafaoglu, and Y. C. Ho, Massively parallel simulation of a class of discrete-event systems, *Proc. IEEE Symp. Frontier Massively Parallel Comput.,* 1992.
55. P. Varaiya, Process models for discrete event systems, in M. A. Kaashoek, J. H. van Schuppen, and A. C. M. Ran (eds.), *Proc. Int. Symp. MTNS, Amsterdam 1989, vol. 1, Realization and Modeling in System Theory,* Boston, MA: Birkhäuser, 1990, pp. 23–41.
56. V. Varpaaniemi et al., PROC Reference Manual, Series B, Tech. Rept. 13, Helsinki University of Technology, 1995.
57. K. Watkins, *Discrete Event Simulation in C,* New York: McGraw-Hill, London, 1993.
58. K. C. Wong, Discrete-event control architecture: An algebraic approach, Ph.D. Thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ontario, Canada, 1994.
59. K. C. Wong and W. M. Wonham, Hierarchical control of discrete-event systems, *Discrete Event Dynamic Syst.,* **6** (3): 241–273, 1996.
60. P. Fanti, G. Maione, and B. Turchiano, New policies for deadlock avoidance in flexible manufacturing cells, *Autom. Strum.,* **44**: 91–95, 1996.
61. W. M. Wonham and P. J. Ramadge, On the supremal controllable sublanguage of a given language, *SIAM J. Control Optim.,* **25** (3): 637–659, 1987.
62. W. M. Wonham and P. J. Ramadge, Modular supervisory control of discrete systems, *Math. Control, Signals Syst.,* **1**: 13–30, 1988.

Reading List

V. Anantharam and T. Konstantopoulos, Stationary solutions of stochastic recursions describing discrete event systems, *Stoch. Process. Appl.,* **68**: 181–194, 1997.

H. Arsham, Goal-seeking problem in discrete event systems simulation, *Microelectron. Reliab.,* **37**: 391–395, 1997.

P. Astuti and B. J. McCarragher, The stability of a class of discrete event systems using Markov Chains, *Int. J. Control,* **64** (3): 391–408, 1996.

J.-L. Boimond and J.-L. Ferrier, Internal model control and max+ algebra: Controller design, *IEEE Trans. Autom. Control,* **41**: 457–461, 1996.

Y. Brave and M. Heymann, On stabilization of discrete-event processes, *Int. J. Control,* **51** (5): 1101–1117, 1990.

P. E. Caines and S. Wang, COCOLOG: A conditional observer and controller logic for finite machines, *SIAM J. Control Optim.,* **33**: 1687–1715, 1995.

X. R. Cao, A comparison of the dynamics of continuous and discrete event systems, *Proc. IEEE,* **77**: 7–13, 1989.

C. G. Cassandras, S. Lafortune, and G. J. Olsder, Introduction to the modeling, control and optimization of discrete event systems, *Trends in Control. A European Perspective,* Rome, Italy, 1995, 217–291.

C. Chase and P. J. Ramadge, On real time scheduling policies for flexible manufacturing systems, *IEEE Trans. Autom. Control,* **37** (4): 491–496, 1992.

S. L. Chung, S. Lafortune, and F. Lin, Limited look ahead policies in supervisory control of discrete event systems, *IEEE Trans. Autom. Control,* **37** (12): 1921–1935, 1992.

R. Cieslak et al., Supervisory control of discrete event processes with partial observations, *IEEE Trans. Autom. Control,* **33** (3): 249–260, 1988.

D. D. Cofer, Control and analysis of real-time discrete event systems, Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, 1995.

G. Cohen and J. P. Quadrat (eds.), Discrete event systems, *11th Int. Conf Anal. and Optimization of Syst.,* INRIA France: Springer Verlag, 1994.

G. Cohen et al., A linear-system theoretic view of discrete-event processes and its use for performance evaluation in manufacturing, *IEEE Trans. Automat. Control,* **AC-30** (3): 210–220, 1985.

A. A. Desrochers and R Y. Al-Jaar, *Applications of Petri Nets in Manufacturing Systems,* New York: IEEE Press, 1995.

M. Fabian and B. Lennartson, Object-oriented supervisory control with a class of nondeterministic specifications, *33rd CDC,* Buena Vista, FL, 1994.

H. E. Garcia and A. Ray, State-space supervisory control of reconfigurable discrete event systems, *Int. J. Control,* **63** (4): 767–797, 1996.

B. Gaujal, Parallélisme et simulaion des systèmes à événements discrets, Ph.D. Thesis, L'université de Nice—Sophia Antipolis, 1994.

B. Gaujal et al., High speed simulation of discrete event systems by mixing process oriented and equational approaches, *Parallel Computing,* **23**: 219–233, 1997.

M. Heymann, Concurrency and discrete event control, *IEEE Control Syst. Mag.,* **10** (4): 103–112, 1990.

K. Inan and P. Varaiya, Finitely recursive process models for discrete event systems, *IEEE Trans. Autom. Control,* **33** (7): 626–639, 1988.

V. S. Kouikoglou and Y. A. Phillis, An exact discrete-event model and control policies for production lines and buffers, *IEEE Trans. Autom. Control,* **36** (5): 515–527, 1991.

A. M. Krasnosel'skii and M. Sorene, Discrete asynchronous iterations, *Autom. Remote Control,* **56**, 1557–1564, 1996; translation of *Autom. Telemekh.,* **56** (11): 64–73, 1995.

S. R. Kulkarni and P. J. Ramadge, Model and controller selection policies based on output prediction errors, *IEEE Trans. Autom. Control,* **41**: 1594–1604, 1996.

R. Kumar and L. E. Holloway, Supervisory control of deterministic Petri nets with regular specification languages, *IEEE Trans. Autom. Control,* **41**: 245–249, 1996.

G. Lingzhong et al., D-automaton model of discrete event dynamic system, *Chin. J. Automation,* **7**: 231–238, 1995.

J. M. Lopez et al., A software engineering method for the design of discrete manufacturing cell control, *Int. J. Comput. Integr. Manuf.,* **10**: 126–141, 1997.

O. Maimon and M. Last, Information-efficient control of discrete event stochastic systems, *IEEE Trans. Syst. Man Cybern.,* **27**: 23–33, 1997.

B. McCarragher, Petri net modeling for robotic assembly and trajectory planning, *IEEE Trans. Ind. Electron.,* **41**: 631–640, 1994.

B. McCarragher and H. Asada, The discrete event modeling and trajectory planning of robotic assembly, *ASME J. Dynamic Syst. Measurement Control,* **117** (3): 394–400, 1995.

K. Mossig, Synthesis of a controller for discrete event systems by eigenvector assignment with respect to max-plus-algebra, *Automatisierungstechnik,* **44**: 80–86, 1996.

G. J. Olsder, Eigenvalues of dynamic max-min systems, *Discrete Event Dynamic Syst.,* **1**: 177–207, 1991.

K. M. Passino, A. N. Michel, and P. J. Antsaklis, Lyapunov stability of a class of discrete event systems, *IEEE Trans. Autom. Control,* **39**: 269–279, 1994.

A. Patra, S. Mukhopadhyay, and S. Bose, Logical models of discrete event systems: A comparative exposition, *Sadhana,* **21**: 683–718, 1996.

K. Rudie and W. M. Wonham, Think globally, act locally: Decentralized supervisory control, *IEEE Trans. Autom. Control,* **37** (11): 1692–1708, 1992.

M. Sampath et al., Failure diagnosis using discrete-event models, *IEEE Trans. Control Syst. Technol.,* **4**: 105–124, 1996.

A. V. Savkin et al., Hybrid dynamic systems: Robust control synthesis problems, *Syst. Control Lett.,* **29**: 81–90, 1996.

S. G. Tzafestas and C. N. Athanassiou, Flexible Petri nets for intelligent robot cell modeling, *Found. Computing Decision Sci.,* **20**: 239–252, 1995.

A. Tzes, K. Seongho, and W. R. McShane, Applications of Petri networks to transportation network modeling, *IEEE Trans. Veh. Technol.,* **45**: 391–400, 1996.

R. A. Williams, B. Benhabib, and K. C. Smith, A DES-theory-based hybrid supervisory control system for manufacturing systems, *J. Manuf. Syst.,* **15** (2): 71–83, 1996.

Y. Willner and H. Heymann, Supervisory control of concurrent discrete-event systems, *Int. J. Control,* **54** (5): 1143–1169, 1991.

Y. Xiaojun, M. D. Lemmon, and P. J. Antsaklis, On the supremal controllable sublanguage in the discrete-event model of nondeterministic hybrid control systems, *IEEE Trans. Autom. Control,* **40**: 2098–2103, 1995.

K. Yamalidou et al., Feedback control of Petri nets based on place invariants, *Automatica,* **32**: 15–28, 1996.

S. Young and V. K. Garg, Model uncertainty in discrete event systems, Internal Report, Department of Electrical and Computer Engineering, University of Texas, Austin, TX, 1994.

H. Zhong, Hierarchical control of discrete-event systems, Ph.D. Thesis, Department of Electrical Engineering, University of Toronto, Toronto, Ontario, Canada, 1992.

PAUL DAN CRISTEA
"Politehnica" University of Bucharest

DISCRETE EVENT SYSTEMS (DES). See DISCRETE EVENT DYNAMICAL SYSTEMS.

DISCRETE HARTLEY TRANSFORMS. See HARTLEY TRANSFORMS.

DISCRETE-TIME ANALOG CIRCUITS. See SWITCHED CAPACITOR NETWORKS.