CWI Quarterly Volume 8 (1) 1995, pp. 31 – 45

# Control of Discrete Event Systems – Research at the Interface of Control Theory and Computer Science

Ard Overkamp
Jan H. van Schuppen

*CWI*
*P.O. Box 94079, 1090 GB Amsterdam, The Netherlands*

This expository paper is directed to a general audience of engineers, mathematicians, and computer scientists. A discrete event system is a mathematical model (in the form of an automaton, Petri nets, or process algebra) of, for example, a computer controlled engineering system such as a communication network. Control theory for discrete event systems aims at synthesis procedures for a supervisor that forces a discrete event system such that it satisfies prespecified control objectives. As an example it is discussed how the control problem of blocking prevention for nondeterministic systems may be solved by the use of failure semantics.

## 1. Introduction

The purpose of this paper is to introduce the reader to the research topic of control of discrete event systems. This expository paper is written for a general audience of engineers, mathematicians, and computer scientists. No specific background is needed neither of system and control theory nor of computer science. Only subsection 4.2 contains results derived at CWI.

The motivation for control of discrete event systems comes from control of engineering systems, manufacturing processes, and computer systems. Examples are online scheduling of transactions in databases, control of a rapid thermal processor, and design of a communication protocol. The control objectives in such problems are, for example, liveness, safety, and prevention of blocking.

In modeling of practical control problems use is made of models from computer science: automata, Petri nets, and process algebras. A discrete event system is often taken to be an automaton in which the outside world can influence the occurrence of events.

The control problem for discrete event systems is then often formulated as: Construct a supervisor which observes the events of the system and determines after every event which elements of the set of possible next events must be prevented from occuring. Control objectives are as mentioned above, primarily to guarantee a certain level of liveness and safety.

Control theory for discrete event systems makes use of several subareas of computer science such as automata theory, process algebras, logic, temporal logic, complexity, etc.

A description of the paper by section follows. Section 2 contains motivation and Section 3 models of discrete event systems. Control synthesis problems are discussed in Section 4. Guidelines for further reading may be found in Section 5.

## 2. Motivation

Research in control of discrete event systems is motivated by practical control problems in, for example, communication networks, databases, manufacturing systems, and traffic systems (metro lines, railways, and freeway traffic). See for references Section 5.

Example 1 Consider a telephone network. Subscribers can generate events such as 'taking the receiver off the hook', 'replacing the receiver', 'press a button'. The telephone network itself also generates events, such as 'ring the bell', 'start the dialtone', 'establish a connection'. Some sequences of events represent unwanted behavior. For instance, a bell should not ring if the receiver is off the hook. Other sequences represent wanted behavior. For instance, a connection should be established if the right protocol is followed by both subscribers. The caller should have taken the receiver off the hook, waited for the dialtone, dialed the correct number, etcetera. Some of these event sequences are enforced by the hardware of the telephone network. A receiver can only be replaced after it is taken off the hook. Some sequences have to be enforced by a supervisor. In the old days a human operator was necessary to guarantee the correct behavior. Nowadays a computer does the job. The challenging task is to automatically synthesize the computer program when provided information only about the uncontrolled behavior of the telephone network and the control objectives.

Practical control problems in the areas mentioned lead to control problems at the level of engineering and computer science. Examples of control objectives for problems at this level are:

1. *Safety.* Prevent the controlled system from reaching states at which a disaster may occur.

2. *Liveness.* Guarantee that the controlled system is able to perform a specified minimum level of performance.

An example of a safety property is *blocking*: Prevent that the controlled system reaches a state from which it cannot proceed to any other state. One speaks of *deadlock* in case of a system with two independently operating supervisors in the situation where both supervisors are waiting for each other [10]. Control problems at the engineering level are transformed into control problems at the control theory level, see section 4 below.

Terminology of systems and control is summarized below. An *event* is the occurrence of an action. A *discrete event system* or *plant* is a mathematical model of an object exhibiting a sequence of events. A *supervisor* is the mathematical object that restricts the operation of a discrete event system. The discrete event system in connection with the supervisor will be called the *controlled discrete event system* or the *closed-loop system*. A *control objective* is a specification on the behavior of the closed-loop system.

Control problems for discrete event systems at the level of control theory lead in general to the following theoretical questions:

- *Existence.* Does there exist a supervisor such that the closed-loop system satisfies the control objectives?
- *Decidability.* Can a supervisor be constructed with an algorithm that terminates in a finite number of steps?
- *Algorithm.* How to construct an algorithm that produces a supervisor meeting the control objectives?
- *Complexity.* How, polynomially or exponentially, does the number of computations of an algorithm for the construction of a supervisor depend on the parameters of the problem?

It is the aim of control theory for discrete event systems to answer questions as these.

The approach to control of a discrete event system taken in the research area of systems and control differs from that taken in computer science. In control theory the approach is control synthesis. In computer science the approach is specification and verification. Thus, in computer science a specification is made that the controlled system is to satisfy, an implementation for the closed-loop system is made, and finally it is verified whether or not the closed-loop system meets the specification. The latter step is called *verification.* Verification can be done by automata theoretic tools, see [26]. Simulation is a popular method to test whether an algorithm or program satisfies the specification but for most practical problems complete testing by simulation is unfeasible. Which approach to control is to be preferred, that of control theory or that of computer science? The answer to this question must be based on experience with a large number of practical control problems for discrete event systems. It will take several more years to collect such experience.

The area of modeling and control of discrete event systems is intertwined with computer science. Modeling of discrete event systems is based on computer science models. Also logic and temporal logic is used in both areas. The subject of verification is also of interest to systems and control. The use of computers in engineering and data processing is expected to lead to new control problems for which systems and control, and computer science will be needed.

The approach of supervisory control is entirely different from control theory as practised in stochastic control and from control of queues as practised in operations research. In the latter areas the processing time is of major interest. Correctness is the major concern in control of discrete event systems. In timed discrete event systems the concept of time also appears but often in constraints and not as part of the cost function.

## 3. Modeling of discrete event systems

To model practical control problems in terms of computer science concepts the following model classes are currently used: (1) automata; (2) Petri nets; (3) process algebras. Automata will be described in detail below.

Which model class is to be preferred for the practical control problems mentioned in section 2? The choice of a model class must be a trade-off between descriptive power and complexity. In regard to descriptive power it has been proven that the model class of Petri nets strictly contains the automata, while the model class of process algebras strictly contains Petri nets. A control problem in Petri nets or in process algebras may be undecidable, that is, there does not exist an algorithm for that problem which terminates in a finite number of steps. The authors prefer automata theory over Petri nets because the concept of state is more explicit. This makes control synthesis easier. The authors like the model class of process algebras because of its modeling power. In many engineering disciplines the model class of Petri nets is rather popular.

There follows terminology and notation on automata.

Definition 1 An *automaton* is a collection

$$A = (\Sigma, X, d, x_0, X_m),$$

where $\Sigma$ is a finite set of *event labels* called the *alphabet*, $X$ is set of *states*, $x_0 \in X$ is the *initial state*, $X_m \subset X$ is the set of *marked states*, and $d : \Sigma \times X \to X$ is said to be the transition function.

A *generator* is an automaton in which $d : \Sigma \times X \to X$ is only a partial function. Thus, for $x \in X$ there is a subset $\Sigma(x) \subset \Sigma$ such that $d(.,x) : \Sigma(x) \to X$ is a function.

A *finite state system* is a generator in which $X$ is a finite set. The term *discrete event system*, or for short *system* or *plant*, is defined in this paper to be an automaton.

In an automaton events are modelled as to occur spontaneously. The mechanism that selects an event is not modelled. In a discrete event system there is no model for a clock. Events occur sequentially.

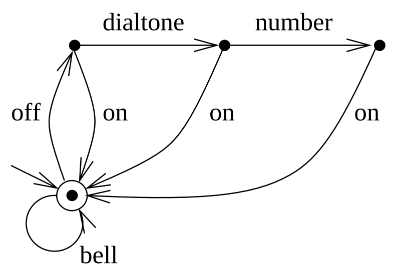

FIGURE 1. Automaton representing a telephone unit.

EXAMPLE 2 A complete model of a system such as a telephone network consists usually of a number of modules, each representing a part of the total system. In Fig. 1 the automaton representing the behavior of the telephone unit is shown. The off-event represents the lifting of the receiver. The on-event represents putting the receiver back on the hook. Encircled nodes indicate marked states. The small arrow that does not start at a node points to the initial state. The automaton describes that a number can only be chosen after a dialtone is given. This is an abstraction of the fact that buttons can be pressed before a dialtone is given, but that these actions do not result in an event inside the system. It is not represented in the automaton that a dialtone should not be given when the receiver is taken off the hook to answer a call. This behavior has to be enforced by a supervisor.

An automaton generates a language which concept is defined below.

DEFINITION 2 Let $\Sigma$ be a set which will be called the *event alphabet*. A *string* is a sequence of events

$$s = (\sigma_1, \sigma_2, \ldots, \sigma_n),$$

where for $n \in Z_+$, $i = 1, \ldots, n$, $\sigma_i \in \Sigma$. The *empty string*, denoted by $\epsilon$, is defined to be the string without elements. The *set of all strings* over $\Sigma$ including the empty string, is denoted by $\Sigma^*$. A *language* is defined to be a subset $L \subset \Sigma^*$.

The *prefix closure* of a language $L$, denoted by $\bar{L}$, is defined to be the set

$$\bar{L} = \{s \in \Sigma^* | \exists t \in \Sigma^* \text{ such that } st \in L\}.$$

The language $L \subset \Sigma^*$ is said to be *prefix closed* if $L = \bar{L}$.

DEFINITION 3 Let $G = (\Sigma, X, d, x_0, X_m)$ be a generator. Extend the transition function $d : \Sigma \times X \to X$ to $d : \Sigma^* \times X \to X$ by

$$\begin{aligned} d(\epsilon, x) &= x, \\ d(s\sigma, x) &= d(\sigma, d(s, x)), \text{ if well defined and for } \sigma \in \Sigma^*, s \in \Sigma. \end{aligned}$$

The *behavior* of $G$ is defined to be the language

$$L(G) = \{s \in \Sigma^* | d(s, x_0) \text{ is defined}\}, \tag{1}$$

and the *marked behavior* is defined to be the language

$$L_m(G) = \{s \in L(G) | d(s, x_0) \in X_m\}. \tag{2}$$

A string in $L_m(G)$ is said to be a *marked string*.

A string in the behavior is a finite string that $G$ can generate. A string in the marked behavior is a finite string that ends in a marked state. A marked string represents a completed task. If $G$ is a generator then by definition of $L(G)$ this set is prefix closed.

Does there exist a generator $G$ for a given language $L$ such that the language generated by $G$ equals $L$, or $L(G) = L$? This representation problem is rather fundamental in system theory and automata theory. Not every language has such a representation. A major theorem of automata theory, see [21, Section 2.5], states that any regular language can be represented by a finite-state automaton. The concept of a regular language will not be defined in this paper because of space limitations. Thus, a finite-state generator produces a regular language while a regular language can be represented as being generated by a generator. The formalisms of regular languages and of finite state generators are thus equivalent. In the remainder of the paper the two formalisms are used interchangeably, with most results being formulated in terms of languages.

Control can be enforced by synchronization of the plant with a controller (supervisor). A supervisor can only block events of the plant. It cannot enforce the execution of a event.

DEFINITION 4 The *synchronous composition* of plant $G = (\Sigma, X, d_g, x_0, X_m)$ and supervisor $S = (\Sigma, Q, d_s, q_0, Q_m)$ is the automaton $G||S = (\Sigma, X \times Q, d_{gs}, (x_0, q_0), X_m \times Q_m)$, where

$$d_{gs}(\sigma, (x_g, q_s)) = \begin{cases} (d_g(\sigma, x_g), d_s(\sigma, q_s)), & \text{if well defined,} \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

Events in the synchronous composition are possible only if they are possible in the plant as well as in the supervisor. Then

$$\begin{aligned} L(G||S) &= L(G) \cap L(S), \\ L_m(G||S) &= L_m(G) \cap L_m(S). \end{aligned}$$

## 4. CONTROL SYNTHESIS

The purpose of this section is to describe how practical control problems are formulated and solved at the level of control theory.

## *4.1. Supervisory control synthesis*

In this subsection the basic concepts of supervisory control, as introduced by Ramadge and Wonham [39], will be explained. The general problem of control theory for discrete event systems is to find a controller (supervisor) that influences the behavior of the plant in such a way that it meets the control objectives.

In some applications the supervisor does not have the ability to block all events. For instance if an alarm event is executed when some water level exceeds a treshhold, then this event can be observed by the supervisor but it cannot be blocked. If this event has to be prevented from occuring then somewhere else in the system some other events have to be blocked (For instance the closing of a waste gate) such that the alarm event cannot occur anymore. Usually the presence of uncontrollable events is modelled by splitting up the event set into two subsets $\Sigma_c$ and $\Sigma_u$, where $\Sigma_c$ represents the controllable events, and $\Sigma_u$ the uncontrollable events. It is required that a supervisor never blocks an uncontrollable event. Such a supervisor is called *complete.*

The main objective of control synthesis for discrete event systems is to find a complete supervisor which allows only legal event sequences. These sequences together form the *legal language.* This language is specified by an automaton, denoted $E$, which generates exactly all legal strings. The basic control objective is to find a complete supervisor such that $L(G||S) = L(E)$. It was shown by Ramadge and Wonham that this supervisor exists only if the plant cannot go from a legal string to an illegal string by executing only uncontrollable events. This property is formulated in the controllability condition.

Definition 5 Let $G$ be a plant and $\Sigma_u$ the set of uncontrollable events. The language $K$ is said to be *controllable* if

$$\bar{K}\Sigma_u \cap L(G) \subseteq \bar{K},$$

where $\bar{K}\Sigma_u = \{s\sigma \in \Sigma^* | s \in \bar{K}, \sigma \in \Sigma_u\}$.

Theorem 1 *Let $G$ be a plant and $E$ a specification of the legal behavior, with $L(E) \subseteq L(G)$. There exists a complete supervisor, $S$, such that $L(G||S) = L(E)$ if and only if the language $L(E)$ is controllable.*

If the language $L(E)$ is not controllable then there exists no supervisor such that $G||S$ generates exactly all legal strings. The control objectives may be relaxed such that any system that generates no illegal strings is satisfactory. Thus, a supervisor is looked for such that $L(G||S) \subseteq L(E)$.

Theorem 2 *Let $G$ be a plant and $E$ a specification of the legal behavior, with $L(E) \subseteq L(G)$. There exists a complete supervisor, $S$, such that $L(G||S) \subseteq L(E)$ if and only if there exist a prefix-closed and controllable language contained in $L(E)$.*

Ramadge and Wonham also showed that the set of languages that are prefix-closed, controllable and contained in $L(E)$ is closed under arbitrary unions. This implies that there is a unique supremal element in this set. That is, there exists a language such that all languages that are controllable and contained in $L(E)$ are a subset of this language. From lattice theory a fixed point algorithm is known that computes this supremal language. This algorithm has polynomial complexity with respect to the number of states in the state space of the automata $G$ and $E$. The automaton that generates this supremal language can be used as supervisor.

### *4.2. Blocking*

The relaxed control objective in the previous section does not guarantee that the closed-loop system will never block. After the system has generated a certain string, it may happen that no subsequent event is possible. Either events cannot be generated by the plant, or the supervisor blocks the events. The marked behavior, as defined in Section 2, may be used to guarantee that systems are nonblocking. Because we will use another definition of nonblocking later on, we will indicate this form with marking-nonblocking. A system is said to be marking-nonblocking if every string that the system generates can be extended to a marked string.

DEFINITION 6 System $E$ is said to be *marking-nonblocking* if

$$L(E) = \overline{L_m(E)}.$$

The supervisory control problem for systems with marking is to find a complete supervisor such that $L_m(G||S) \subseteq L_m(E)$ and $G||S$ is marking-nonblocking. Note that these two conditions together imply that no illegal string will be generated. That is, $L(G||S) = \overline{L_m(G||S)} \subseteq \overline{L_m(E)} \subseteq L(E)$.

THEOREM 3 *Let $G$ be a plant and $E$ the specification of the legal behavior, with $L_m(E) \subseteq L_m(G)$. There exists a complete supervisor, $S$, such that $L_m(G||S) \subseteq L_m(E)$ and $L(G||S) = \overline{L_m(G||S)}$, if and only if there exist a controllable language contained in $L_m(E)$.*

Note that the definition of controllability is given for general, not necessarily prefix-closed, languages. As in the previous section, the set of languages that are controllable and contained in $L_m(E)$ is closed under arbitrary unions. This means that there exists a supremal language in this set. Let $K$ be this supremal language. Then, the automaton, S, with $L(S) = \bar{K}$ and $L_m(S) = K$ can be used as a supervisor.

If some parts of a system are not completely modelled, or only a part of the system can be observed, then the system may exhibit nondeterministic behavior. That is, the observed sequence of events does not uniquely determine the state of the system.

DEFINITION 7 A *nondeterministic automaton* is defined to be a automaton in which the transition function $d$ is of the form $d : \Sigma \times X \to 2^X$. The set $d(\sigma, x)$ precisely contains all states that can be reached from state $x$ by event $\sigma$.

Consider now the supervisory control problem of blocking prevention for nondeterministic systems. Marking is not sufficient to guarantee that nondeterministic systems are nonblocking. Consider the following example.

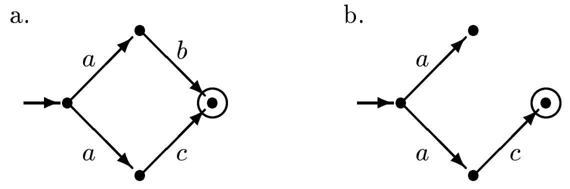

FIGURE 2. Blocking of a nondeterministic automaton.

EXAMPLE 3 Let $G$ be an automaton as in Fig. 2.a. Suppose string $ab$ is illegal. If event $b$ is blocked, then an automaton as in Fig. 2.b is obtained. It is clear that this system can block after event $a$. But this is not detectable by considering the marked language. From $L(G) = \{a, ac\} = \overline{\{ac\}} = \overline{L_m(G)}$ it follows that $G$ is marking-nonblocking.

Hoare [19] introduced a different method to deal with blocking in nondeterministic systems. Not only the language of the system is considered but also the events that cannot be executed are taken into account. If a nondeterministic system is offered a set of admissible events, via the synchronous composition, and the system can be in a state in which it cannot execute any of the offered events, then the system is said to refuse all events in this set. Such a set of events is called a refusal.

DEFINITION 8 The *set of refusals* or the *refusal set* of the system $G$ after string $s$ is defined to be the set

$$\begin{aligned} \mathrm{ref}(G,s) &= \{R \subseteq \Sigma : \exists x \in d(s, x_0) \text{ s.t. } R \cap \lambda(x) = \emptyset\}, \\ &\quad \text{where } \lambda(x) = \{\sigma \in \Sigma | d(\sigma, x) \text{ is defined}\}. \end{aligned}$$

Note that a refusal is a set of events. In the definition above $R$ is a refusal. So a refusal set or set of refusals is a set of sets of events.

The method introduced by Hoare is known as failure semantics. Using this method, blocking of a nondeterministic system can be defined as the situation in which all events can be refused.

DEFINITION 9 The (nondeterministic) system $G$ is said to be *nonblocking* if for all $s$ in $L(G)$, $\Sigma \notin \mathrm{ref}(G, s)$.

The controlled system is guaranteed to be nonblocking if the legal behavior is nonblocking and the controlled system does not refuse more than the system describing the legal behavior. This statement motivates the reduction relation.

Definition 10 One says that system $G$ *reduces* system $E$, denoted $G \precsim E$, if

$$L(G) \subseteq L(E), \text{ and}$$
$$\forall s \in L(G), \ \text{ref}(G, s) \subseteq \text{ref}(E, s).$$

The supervisory control problem for nondeterministic systems is to find a complete supervisor, $S$, such that $G||S \precsim E$. It is shown in [37] that an important condition for the existence of such a supervisor is the reducibility condition.

Definition 11 Language K is said to be *reducible* (w.r.t. $G$ and $E$) if

$$\forall s \in K, \quad \forall R_g \in \text{ref}(G, s), \ \rho(K, s) \cup R_g \in \text{ref}(E, s),$$
$$\text{where } \rho(K, s) = \{\sigma \in \Sigma | s\sigma \notin K\}.$$

Theorem 4 *Let $G$ be a nondeterministic system and $E$ a specification of the legal behavior. There exists a complete supervisor, $S$, such that $G||S \precsim E$, if and only if there exists a controllable and reducible language contained in $L(E)$.*

Thus, if $E$ is nonblocking and if there exists a controllable and reducible language contained in $L(E)$, then there exists a supervisor $S$ such that $L(G||S) \subseteq L(E)$ and $G||S$ is nonblocking.

It has been shown that the set of reducible and controllable languages is closed under arbitrary unions. So a unique supremal language is contained in the set and is computable by a fixed point algorithm with polynomial complexity. As with deterministic systems, the (deterministic) automaton that generates this supremal language can be used as supervisor.

## 5. Guidelines for further reading

Practical control problems for which control of discrete event systems has been analysed include: database operations [27]; rapid thermal multiprocessor [6]; and protocol design for communication networks [13, 17, 40].

Automata theory at an introductory level may be found in [21] and at an advanced level in [15]. A book on Petri nets is [14] and a book on related models [4]. The theory of process algebras may be found in [5, 18, 19, 31]. For temporal logic see [30].

Supervisory control of discrete event systems was started and mainly developed by W.M. Wonham and his doctoral students, see [38, 39, 41, 43, 45, 47]. An overview paper is [44]. For publications of S. Lafortune and co-workers see [11, 12, 27, 28]. The supervisory control problem with failure semantics is treated in [37]. A large quantity of additional publications remains unmentioned because of space limitations.

Control of infinite string automata was developed by J.G. Thistle [43]. Such strings are used to express liveness conditions. Techniques to do verification for the associated languages were developed by R.P. Kurshan [26]. A book by Kurshan will appear shortly. Modeling for control of discrete event systems by process algebras was considered by K. Inan and P. Varaiya in [22, 23].

Time plays a role in many practical control problems. Examples of such problems are the operation of a railway gate [36] or the operation of a telephone network. Timed discrete event systems are closely related to the computer science area of real-time systems. A stimulating discussion on theoretical concepts for real-time systems is presented in [25, 42]. Modeling of timed discrete event systems brings with it several new issues compared with untimed discrete event systems, such as the role of durations and forcing of events. Models of timed discrete event systems that have been proposed include discrete-time systems [7], timed automata proposed by R. Alur and D. Dill [1, 3], temporal logic [36], and timed process algebras developed by J. Sifakis and co-workers [32, 33, 34]. Control of timed discrete event systems is treated in [7, 20, 29, 36, 46] of which the work by G. Hoffmann and H. Wong-Toi is of particular interest. An application to specification and design of a telephone exchange is presented in [24].

A hybrid system is a mathematical model of a phenomenon in which the model includes logical variables and continuous variables described by differential equations. Many computer controlled engineering systems are hybrid systems, for example a temperature controller for a house or the controller of an air plane. Models for hybrid systems were proposed in [2, 9, 16, 34, 35]. For an approach to control of hybrid systems, see [8].

## 6. Concluding remarks

What has been achieved in control of discrete event systems? For practical problems with logical variables discrete event systems have been formulated as mathematical models. These systems are the basic building blocks for control. Control synthesis results yield algorithms that produce supervisors that will satisfy a specified level of performance.

What research directions should be explored in control of discrete event systems? First experience must be gained with realistic and practical problems as they appear in industrial laboratories. Modeling of discrete event systems would benefit from a deeper analysis of the trade-off between modeling power and complexity. Hierarchical decomposition may be a direction to explore. A discrete event system has little mathematical structure hence it is difficult to enlist the aid of parts of traditional mathematics. The developments in theoretical computer science should be watched closely. Control theory of discrete event systems should also concentrate attention on decentralized control motivated by the use of networks of computers. Faster algorithms for control synthesis would be useful in practice.

Control of timed discrete event systems needs more motivation by realistic and practical problems. Experience must be gained with the model classes of timed discrete event systems and timed process algebras. Control of hybrid systems leads to a diverse set of problems. Research in this area has only recently started.

## Acknowledgements

The authors acknowledge their influence by the pioneering research in control of discrete event systems of W.M. Wonham (University of Toronto) and his doctoral students. They also acknowledge discussions on the subject with R.K. Boel, K. Inan, S. Lafortune, F. Vaandrager, and P. Varaiya.

## References

1. R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In X, editor, *Proc. of the 5th IEEE Symposium on Logic in Computer Science*, pages 414–425, X, 1990. X.
2. R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In G. Cohen and J.-P. Quadrat, editors, *11th International Conference on Analysis and Optimization of Systems - Discrete Event Systems*, number 199 in Lecture Notes in Control and Information Sciences, pages 331–351, London, 1994. Springer-Verlag.
3. R. Alur and D. Dill. The theory of timed automata. In J.W. de Bakker, C. Huizing, and W.P. de Roever, editors, *Real-time: Theory and practice, Proceedings of the REX Workshop, Mook, The Netherlands, June 3-7, 1991*, number 600 in Lecture Notes in Computer Science, pages 45–74, Berlin, 1992. Springer.
4. F.L. Baccelli, G. Cohen, G.J. Olsder, and J.-P. Quadrat. *Synchronization and linearity - An algebra for discrete event systems.* John Wiley & Sons, Chichester, 1992.
5. J.C.M. Baeten and W.P. Weijland. *Process algebra.* Cambridge University Press, Cambridge, 1990.
6. S. Balemi, G.J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G.F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Trans. Automatic Control*, 38:1040–1059, 1993.
7. B.A. Brandin and W.M. Wonham. Supervisory control of times discrete event systems. *IEEE Trans. Automatic Control*, 39:329–342, 1994.
8. M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Background, model, and theory. Report LIDS-P-2239, Laboratory for Information and Decision Systems, M.I.T., Cambridge, MA, 1994.
9. R.W. Brockett. Hybrid models for motion control systems. In H.L. Trentelman and J.C. Willems, editors, *Essays on control: Perspectives in the theory and its applications*, pages 29–53. Birkhäuser, New York, 1993.

10. A. Burns and A. Wellings. *Real-time systems and their programming languages.* Addison-Wesley, X, 1990.
11. E. Chen and S. Lafortune. Dealing with blocking in supervisory control of discrete-event systems. *IEEE Trans. Automatic Control*, 36:724–735, 1991.
12. Sheng-Luen Chung, S. Lafortune, and Feng Lin. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Trans. Automatic Control*, 37:1921–1935, 1992.
13. R. Cieslak, C. Desclaux, A.S. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Automatic Control*, 33:249–260, 1988.
14. R. David and H. Alla. *Petri nets and grafcet.* Prentice Hall, New York, 1992.
15. S. Eilenberg. *Automata, languages, and machines (Volumes A and B).* Academic Press, New York, 1974, 1976.
16. R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid systems.* Number 736 in Lecture Notes in Computer Science. Springer, New York, 1993.
17. E. Haghverdi and K. Inan. Verification by consecutive projections. In X, editor, *FORTE 92 Proceedings*, pages x–y, X, 1992. X.
18. M. Hennessy. *Algebraic theory of processes.* M.I.T. Press, Cambridge, MA, 1988.
19. C.A.R. Hoare. *Communicating sequential processes.* Prentice/Hall International, Englewood Cliffs, NJ, 1985.
20. G.J. Hoffmann and H. Wong-Toi. The input-output control of real-time discrete event systems. Report ISL/GFF/92-1, Information Systems Laboratory, Stanford University, Stanford, 1992.
21. J.E. Hopcroft and J.D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Company, Reading, MA, 1979.
22. K. Inan and P. Varaiya. Finitely recursive process models for discrete event systems. *IEEE Trans. Automatic Control*, 33:626–639, 1988.
23. K.M. Inan and P.P. Varaiya. Algebras of discrete event models. *Proc. IEEE*, 77:24–38, 1989.
24. A. Kay and J.N. Reed. A rely and guarantee method for times CSP: A specification and design of a telephone exchange. *IEEE Trans. Software Eng.*, 19:625–639, 1993.
25. R. Kurki-Suonio. Real-time: Further misconceptions (or half-truths). *Computer*, 27:71–76, 1994 (June, no. 6).
26. R.P. Kurshan. Automata-theoretic verification of coordinating processes. In G. Cohen and J.-P. Quadrat, editors, *11th International Conference on Analysis and Optimization of Systems - Discrete Event Systems*, number 199 in Lecture Notes in Control and Information Sciences, pages 16–28, London, 1994. Springer-Verlag.
27. S. Lafortune. Modeling and analysis of transaction execution in database systems. *IEEE Trans. Automatic Control*, 33:429–447, 1988.

28. S. Lafortune and Enke Chen. The infimal closed controllable superlanguage and its application in supervisory control. *IEEE Trans. Automatic Control*, 35:398–405, 1990.
29. Y. Li and W.M. Wonham. Supervisory control of real-time discrete event systems. *Information Sciences*, 46:159–183, 1988.
30. Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems - Specification.* Springer-Verlag, Berlin, 1992.
31. R. Milner. *Communication and concurrency.* Prentice-Hall, Englewood Cliffs, NJ, 1989.
32. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: Theory and applications. *Information and Computation*, 114:131–178, 1994.
33. X. Nicollin, J. Sifakis, and S. Yovine. Compiling real-time specifications into extended automata. *IEEE Trans. Software Engineering*, 18:794–804, 1992.
34. X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. In J.W. de Bakker, C. Huizing, and W.P. de Roever, editors, *Real-time: Theory and practice, Proceedings of the REX Workshop, Mook, The Netherlands, June 3-7, 1991*, number 600 in Lecture Notes in Computer Science, pages 45–74, Berlin, 1992. Springer.
35. X. Nicollin, J. Sifakis, and S. Yovine. From ATP to timed graphs and hybrid systems. *Acta Informatica*, 30:181–202, 1993.
36. J.S. Ostroff. *Temporal logic for real-time systems.* Research Studies Press Ltd., Taunton, England, 1989.
37. A. Overkamp. Supervisory control for nondeterministic systems. In G. Cohen and J.-P. Quadrat, editors, *11th International Conference on Analysis and Optimization of Systems - Discrete Event Systems*, number 199 in Lecture Notes in Control and Information Sciences, pages 59–65, London, 1994. Springer-Verlag.
38. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25:206–230, 1987.
39. P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proc. IEEE*, 77:81–98, 1989.
40. K. Rudie and W.M. Wonham. Protocol verification using discrete-event systems. In *Proceedings of the 31st IEEE Conference on Decision and Control*, pages 3770–3777, New York, 1992. IEEE Press.
41. K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Automatic Control*, 37:1692–1708, 1992.
42. J.A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21, No. 10 (Oct.):10–19, 1988.
43. J.G. Thistle. *Control of infinite behaviour of discrete-event systems.* PhD thesis, Department of Electrical Engineering, University of Toronto, Toronto, 1991.
44. J.G. Thistle. Logical aspects of control of discrete event systems: A survey of tools and techniques. In G. Cohen and J.-P. Quadrat, editors, *11th International Conference on Analysis and Optimization of Systems - Discrete*

*Event Systems*, number 199 in Lecture Notes in Control and Information Sciences, pages 3–15, London, 1994. Springer-Verlag.

45. K.C. Wong. *Discrete-event control architecture: An algebraic approach.* Systems and control group report, University of Toronto, Toronto, 1994.
46. Howard Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. Report STAN-CS-92-1411, Department of Computer Science, Stanford University, Stanford, 1992.
47. W.M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control & Opt.*, 25:637–659, 1987.