ANNUAL REVIEWS

Annual Review of Control, Robotics, and Autonomous Systems

Discrete Event Systems: Modeling, Observation, and Control

Stéphane Lafortune

Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109-2122, USA; email: stephane@umich.edu

Annu. Rev. Control Robot. Auton. Syst. 2019. 2:141–59

First published as a Review in Advance on November 9, 2018

The *Annual Review of Control, Robotics, and Autonomous Systems* is online at control.annualreviews.org

https://doi.org/10.1146/annurev-control-053018-023659

Keywords

discrete event systems, supervisory control, diagnosis, opacity, automata

Abstract

This article begins with an introduction to the modeling of discrete event systems, a class of dynamical systems with discrete states and event-driven dynamics. It then focuses on logical discrete event models, primarily automata, and reviews observation and control problems and their solution methodologies. Specifically, it discusses diagnosability and opacity in the context of partially observed discrete event systems. It then discusses supervisory control for both fully and partially observed systems. The emphasis is on presenting fundamental results first, followed by a discussion of current research directions.

1. INTRODUCTION

Discrete event systems (DESs) are dynamic systems with two defining characteristics: Their state spaces are discrete and potentially infinite, and their dynamics are event driven as opposed to time driven. This means that there are no differential or difference equations in this class of systems. Events that occur asynchronously (in general) cause a jump in the state space, from one state to another. The set of events, denoted by E, is also a discrete set; the discussion that follows assumes that E has finite cardinality.

There are dynamic systems that are inherently modeled as DESs; consider, for instance, modeling the rules of a communication protocol, the movement of parts in an automated manufacturing system, or the automation logic in a process control system. However, DESs can also be obtained by abstracting, in both space and time, the behavior of cyber-physical systems. For instance, one can abstract the possible trajectories of a robot in physical space by constructing a discrete grid over that space and defining events for transitions from one cell of the grid to an adjacent one.

The behavior of a DES is therefore captured by a discrete transition structure, where the nodes represent the states and the arcs connecting the nodes represent all the possible transitions between states that are caused by the occurrence of events. Such discrete transition structures can take different forms, depending on what attributes are to be attached to them for logical or quantitative analyses. This article focuses primarily on automata models of DESs but also discusses labeled transition systems and Petri nets. After presenting these modeling formalisms, it reviews observation and control problems for DESs and presents general solution methodologies. Specifically, it reviews the properties of diagnosability and opacity in the context of partially observed DESs. It then discusses supervisory control for both fully and partially observed systems, when logical specifications are imposed on the controlled behavior. The emphasis is on reviewing fundamental results first, followed by a discussion of current research directions.

2. MODELING OF DISCRETE EVENT SYSTEMS

This section contains introductory material that is necessary to make the review accessible to nonexperts; more advanced topics are discussed in the subsequent sections on observation and control. For more details on the topics covered in this section, see References 1–6.

2.1. Modeling Formalisms

Discrete event modeling formalisms must capture the event-triggered transitions between the discrete states of a DES. The most common modeling formalism is that of automata, which has a nice connection with formal language theory.

2.1.1. Automata. A deterministic automaton, denoted by G, is a six-tuple

$$G = (X, E, f, \Gamma, x_0, X_m),$$

where X is the set of states, which could be infinite; E is the finite set of events associated with the transitions in G; $f : X \times E \to X$ is the transition function; $f(x, e) = y$ means that there is a transition labeled by event e from state x to state y (in general, f is a partial function on its domain); $\Gamma : X \to 2^E$ is the feasible event function; $\Gamma(x)$ is the set of all events e for which $f(x, e)$ is defined; x_0 is the initial state; and $X_m \subseteq X$ is the set of marked states. (Given a set A, the notation 2^A means the power set of A, i.e., the set of all subsets of A.) When discussing several automata, we will index the various elements with the name of the automaton if needed to avoid confusion: X_G, E_G, and so forth.

A nondeterministic automaton is a six-tuple

$$G = (X, E \cup \{\varepsilon\}, f, \Gamma, X_0, X_m),$$

where these objects have the same interpretation as in the definition of a deterministic automaton, with two changes (here, ε denotes the empty string): (*a*) f is a (partial) function $f : X \times (E \cup \{\varepsilon\}) \to 2^X$, i.e., $f(x, e) \subseteq X$ whenever it is defined, and (*b*) the initial state may itself be a set of states, i.e., $X_0 \subseteq X$. (This abuses the notation slightly and uses the same symbol, f, for the transition function of both a nondeterministic automaton and a deterministic one; unless specified otherwise, the discussion below applies to both cases.)

Automata (deterministic or nondeterministic) are used to represent and manipulate formal languages over event set E. Denote by E^* the set of all finite strings of elements of E; this operation is called the Kleene closure. The transition function f of a deterministic automaton from domain $X \times E$ is extended to domain $X \times E^*$ in the recursive manner:

$$\begin{aligned} f(x, \varepsilon) &:= x, \\ f(x, se) &:= f[f(x, s), e] \text{ for } s \in E^* \text{ and } e \in E. \end{aligned}$$

A similar extension is done for the transition function of a nondeterministic automaton, but the details are omitted.

A DES is then modeled as an automaton G that represents two languages: $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$. Their definitions for a deterministic ($X_0 = \{x_0\}$, in this case) or nondeterministic G are as follows: The language generated by G is defined as

$$\mathcal{L}(G) := \{s \in E^* : (\exists x_0 \in X_0) f(x_0, s) \text{ is defined}\}$$

and the language marked by G is defined as

$$\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) : (\exists x_0 \in X_0) f(x_0, s) \cap X_m \neq \emptyset\}.$$

String s is in $\mathcal{L}(G)$ if and only if it corresponds to an admissible path in the transition structure of G, starting at an initial state. $\mathcal{L}(G)$ is prefix-closed by definition, meaning that every prefix of every string in $\mathcal{L}(G)$ is also in $\mathcal{L}(G)$. If f is a total function over its domain, then necessarily $\mathcal{L}(G) = E^*$. The set of states that can be reached from X_0 by a string in $\mathcal{L}(G)$ is the set of accessible states of G. In practice, we assume that all states in X are accessible (i.e., states that are not accessible are removed at modeling time).

Language $\mathcal{L}_m(G)$ is the subset of $\mathcal{L}(G)$ consisting only of the strings s for which $f(x_0, s) \cap X_m \neq \emptyset$—i.e., these strings correspond to paths in the transition structure that end at a marked state (or may end there, if G is nondeterministic). Language $\mathcal{L}_m(G)$ is not prefix-closed in general, unless $X_m = X$.

Marking of states is used to study behavioral properties such as deadlock and livelock. A deadlock state of deterministic automaton G is a state where $\Gamma(x) = \emptyset$ but $x \notin X_m$; this means that the system has no possible transition at the current state, and that state is not marked. A livelock occurs where there is a strongly connected component containing at least two states, none of them marked, and from which no marked state is reachable by following the transitions in G. The states of G that can eventually reach a marked state are called coaccessible. The notion of nonblockingness is used to formalize deadlock and livelock analysis. Automaton G is said to be blocking if $\overline{\mathcal{L}_m(G)} \subset \mathcal{L}(G)$, where the set inclusion is proper, and nonblocking when $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$. Thus, if an automaton is blocking, deadlock and/or livelock can happen. In a nonblocking automaton, all states are coaccessible and there are no deadlocks or livelocks.

When using automaton models, we often exploit the duality between regular languages and their finite-state automata representations. One can think of finite-state automata as a simple data structure to represent regular languages. This duality is exemplified by stating properties of DESs in terms of formal languages and then verifying them using automaton models.

2.1.2. Labeled transition systems. A modeling formalism that is widely used in formal methods is that of labeled transition systems. A labeled transition system, denoted by T, is a discrete transition structure that can be viewed as a generalization of an automaton with outputs (or Moore automaton). We define a labeled transition system as a six-tuple $T = (X, E, f_t, X_0, AP, L)$, where X is the state set, $X_0 \subseteq X$ is the initial state set, and E is the event set. T is in general nondeterministic, and $f_t : X \times (E \cup \{\varepsilon\}) \to 2^X$ is the corresponding partial transition function. The transitions of T are labeled by events in the set E or by ε, according to the function f_t, as in an automaton; this labeling is the reason for the qualifier "labeled" in the term labeled transition system. In many cases, the transitions of T are not labeled—i.e., the event set E is not used. In this case, we drop the adjective "labeled" and refer to the model as a transition system. A feature that differentiates transition systems from automata is the set AP, which is a set of atomic propositions (or propositional symbols) to be associated with the states of T. Such atomic propositions are used to express desired system properties as logical formulas in temporal logic, such as linear temporal logic or computation tree logic. The function L is used to map the states of T to the set of atomic propositions that hold true in each state—that is, $L : X \to 2^{AP}$ is the labeling function of T.

When studying the dynamical behavior of a labeled transition system, the focus is typically not on the string of events generated, but rather on the sequence of states and events generated, called a run, along with the set of atomic propositions at each state of the run.

2.1.3. Petri nets. Another modeling formalism for DESs that is widely used is that of Petri nets. Petri nets are also discrete structures, but unlike automata or labeled transition systems, they do not explicitly represent each state of the system. Instead, they use a bipartite discrete structure with place nodes and transition nodes to model pre- and postconditions on the execution of events, using a token mechanism to capture these conditions. The transition nodes of the Petri net can be labeled by events, leading to a labeled Petri net that will then generate a formal language. However, the primary advantage of the Petri net modeling formalism is that the state of a Petri net (which is the vector of token counts in each of the places) is numerical, as each entry takes nonnegative integer values. One can then write a state update equation that depends on the incidence matrix of the Petri net structure. Using this incidence matrix and the state vector, one can leverage linear algebraic techniques to study properties of Petri nets. A recent review by Giua & Silva (7) provides an overview of the historical development of the field of Petri nets from the perspective of systems theory and automatic control.

2.2. Composition Operation

Models of DESs are rarely obtained in a monolithic manner. In practice, each system component is modeled individually, and the overall DES is the composition of the individual models. This section reviews the most common composition operation for automata.

2.2.1. Parallel composition of automata. Consider two (deterministic) automata $G_1 = (X_1, E_1, f_1, \Gamma_1, x_{01}, X_{m1})$ and $G_2 = (X_2, E_2, f_2, \Gamma_2, x_{02}, X_{m2})$. G_1 and G_2 are assumed to be accessible (i.e., all states that are not reachable from their respective initial states have been deleted,

as have the transitions attached to these states); this is the accessible operation, denoted by Ac. No assumptions are made at this point about the two event sets E_1 and E_2.

The parallel composition of G_1 and G_2 is the automaton

$$G_1||G_2 := Ac[X_1 \times X_2, E_1 \cup E_2, f, \Gamma_{1||2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}],$$

where

$$f[(x_1, x_2), e] := \begin{cases} [f_1(x_1, e), f_2(x_2, e)] & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ [f_1(x_1, e), x_2] & \text{if } e \in \Gamma_1(x_1) \setminus E_2 \\ [x_1, f_2(x_2, e)] & \text{if } e \in \Gamma_2(x_2) \setminus E_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

and thus $\Gamma_{1||2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus E_2] \cup [\Gamma_2(x_2) \setminus E_1]$.

In the parallel composition, a common event (i.e., an event in $E_1 \cap E_2$) can be executed only if the two automata both execute it simultaneously. Thus, the two automata are synchronized on the common events. The other, private events—namely, those in $(E_2 \setminus E_1) \cup (E_1 \setminus E_2)$—are not subject to such a constraint and can be executed whenever possible.

If $E_1 = E_2$, then all transitions are forced to be synchronized, yielding

$$\begin{aligned} \mathcal{L}(G_1||G_2) &= \mathcal{L}(G_1) \cap \mathcal{L}(G_2), \\ \mathcal{L}_m(G_1||G_2) &= \mathcal{L}_m(G_1) \cap \mathcal{L}_m(G_2). \end{aligned}$$

If $E_1 \cap E_2 = \emptyset$, then there are no synchronized transitions, and $G_1||G_2$ is the concurrent behavior of G_1 and G_2. This is often termed the shuffle of G_1 and G_2.

In the general case,

$$\mathcal{L}(G_1||G_2) = P_1^{-1}[\mathcal{L}(G_1)] \cap P_2^{-1}[\mathcal{L}(G_2)], \mathcal{L}_m(G_1||G_2) = P_1^{-1}[\mathcal{L}_m(G_1)] \cap P_2^{-1}[\mathcal{L}_m(G_2)],$$

where we have used projection operations and their inverses. Specifically, the projection operation

$$P_i : (E_1 \cup E_2)^* \to E_i^* \text{ for } i = 1, 2$$

is defined as follows:

$$\begin{aligned} P_i(\varepsilon) &:= \varepsilon, \\ P_i(e) &:= \begin{cases} e & \text{if } e \in E_i, \\ \varepsilon & \text{if } e \notin E_i, \end{cases} \\ P_i(se) &:= P_i(s)P_i(e) \text{ for } s \in (E_1 \cup E_2)^*,\ e \in (E_1 \cup E_2). \end{aligned}$$

The corresponding inverse projection $P_i^{-1} : E_i^* \to 2^{(E_1 \cup E_2)^*}$ is obtained as follows: For a string $t \in E_i^*$, we have that

$$P_i^{-1}(t) := \{s \in (E_1 \cup E_2)^* : P_i(s) = t\};$$

that is, $P_i^{-1}(t)$ returns all the strings in the domain of P_i that are mapped to projected string t. These definitions of projection and inverse projection are extended to sets of strings in the obvious manner.

2.2.2. Composition of transition systems and Petri nets. Labeled transition systems and Petri nets can also be composed by parallel composition or related operations. In the case of Petri nets, composition of models can often be done by overlapping places that represent the coupling between different components; when this is possible, we refer to the component models as

place-bordered Petri nets. In this case, the size of the model (i.e., the Petri net structure) grows linearly in the number of components.

2.3. Levels of Abstraction

The discussion in this article focuses on logical models of DESs, where the behavior is described by strings of events. The ordering of these events is essential, but exact timing information is not included, which is referred to as the logical level of abstraction. When timing information is added to a logical DES model, we obtain a timed model, such as a timed automaton (8). The temporal level of abstraction is needed when dealing with deadline constraints, for instance. By definition, it is richer than the logical level. Finally, when statistical information about the occurrence of events in a state is added to a logical or a timed DES model, we obtain a stochastic (timed) DES model. There are a large variety of such models, depending on the statistical assumptions made about the transition probabilities and about the time durations spent in states between transitions. Stochastic or probabilistic automata are extensions of logical ones where transition probabilities are included in the model but time durations in states are not explicitly accounted for. Generalized semi-Markov processes are a general class of stochastic processes generated by stochastic timed automata; continuous-time Markov chains are a widely studied special class of those (see, e.g., 1). The stochastic level of abstraction is needed when quantitative performance criteria need to be evaluated, in a probabilistic sense. The stochastic level is richer than the logical or temporal levels. These levels of abstraction explain the large and often disparate literature on DESs.

At each level of abstraction, several classes of problems arise, such as behavior analysis (reachability of states, state estimation, etc.), control to achieve a given specification, optimization of a given performance criterion (qualitative or quantitative), and simulation. Discrete event simulation has been studied for a long time, primarily in the operations research community.

The remainder of this review focuses on the logical level of abstraction and on specifications that do not involve numerical quantities. We discuss observation problems that involve state estimation and model-based inferencing, as well as supervisory control problems for logical specifications that involve safety and nonblockingness.

3. OBSERVATION OF DISCRETE EVENT SYSTEMS

This section discusses two problems related to observing the behavior of a DES: event diagnosis and opacity. It considers automaton models and assumes that the automaton under consideration is either nondeterministic or partially observed. In a partially observed automaton, the set of events E is partitioned as $E = E_o \cup E_{uo}$, where E_o is the set of observable events and E_{uo} is the set of unobservable events; these two sets are disjoint. The projection operation $P_o : E^* \to E_o^*$ is defined similarly to the above P_i to erase unobservable events. In this context, the notion of an observer automaton, or simply an observer, is central. The observer automaton of a nondeterministic or partially observed automaton is a determinized version obtained by the standard subset construction method (see, e.g., 1). In order to formalize this process, we define two operators.

The unobservable reach of the subset of states $S \subseteq X$ is given by

$$UR(S) := \{x \in X | (\exists u \in S)(\exists s \in E_{uo}^*)\ x = f(u, s)\}. \quad 1.$$

The observable reach (or next states) of the subset of states $S \subseteq X$ given the execution of observable event $e \in E_o$ is defined as

$$NX(S, e) := \{x \in X | (\exists u \in S)\ x = f(u, e)\}. \quad 2.$$

The observer of G, denoted by $Obs(G)$, is the deterministic automaton $Obs(G) = (X_{obs}, E_o, f_{obs}, x_{obs,0})$, where $X_{obs} \subseteq 2^X$ is the state space, E_o is the set of observable events, the initial state is defined as $x_{obs,0} = UR(X_0)$, and the transition function $f_{obs} : X_{obs} \times E_o \to X_{obs}$ is defined as $f_{obs}(S, e) = UR[NX(S, e)]$. In practice, only the accessible part of the observer from its initial state is constructed; thus, X_{obs} is the set of reachable states. By construction, $\mathcal{L}[Obs(G)]$ equals $P_o[\mathcal{L}(G)]$. We call the state $f_{obs}[x_{obs,0}, P_o(s)]$ reached in the observer by string $s \in \mathcal{L}(G)$ the current state estimate associated with string s, where we have extended the transition function of the observer to strings.

3.1. Diagnosability

Event diagnosis is an important issue, not only for faults but also for attacks that may occur on a given system. An unobservable event e_d is diagnosable in language $\mathcal{L}(G)$ if every occurrence of e_d can be detected with certainty in a bounded number of events after its occurrence. To avoid dealing with terminating traces, we assume that $\mathcal{L}(G)$ is live—i.e., $(\forall x \in X)\Gamma(x) \neq \emptyset$. Formally, $e_d \in E_{uo}$ is diagnosable in live language $\mathcal{L}(G)$ with respect to projection P_o if the following holds:

$$(\exists n \in \mathbb{N})(\forall s : e_d \in s)(\forall t \in \mathcal{L}(G)/s)[||t|| \geq n \Rightarrow D], \quad 3.$$

where the diagnosability condition D is

$$\omega \in P_o^{-1}[P_o(st)] \cap \mathcal{L}(G) \Rightarrow e_d \in \omega. \quad 4.$$

In the above, $e_d \in s$ means that string s contains event e_d, $\mathcal{L}(G)/s$ is the set of all strings t such that $st \in \mathcal{L}(G)$, and $||t||$ is the length (number of events) of t. The set $P_o^{-1}[P_o(st)] \cap \mathcal{L}(G)$ is the best estimate of what the system could have done based on observing st.

This notion of diagnosability was introduced by Sampath et al. (9). It is a strong requirement, as it invokes the universal quantifier twice: For every trace of events that ends with event e_d and for every continuation of that trace, the event in question must eventually be diagnosed—i.e., we are sure that it did occur in the past. Since logical discrete event models are employed, "eventually" is quantified by counting the number of events after e_d; the existential quantifier captures the existence of such a bound, denoted by n, over the entire system language.

Three problem domains pertain to event diagnosis: how to perform diagnosis online, how to determine (offline) whether diagnosability holds for a given system, and how to enforce diagnosability if it does not hold. Regarding the third domain, the two possible enforcement techniques are to restrict the behavior of the system by supervisory control, in order to prevent event strings where diagnosability is violated (this is a special instance of the problem discussed in Section 5), and to dynamically adjust the observations emitted by the system by turning sensors on or off, in order to guarantee diagnosability. The second is known as the problem of sensor activation.

An important concept used in solving the first two problems above is that of the diagnoser automaton, or simply the diagnoser, denoted by $Diag(G)$. The diagnoser is a refined version of the observer, where labels are attached to system states as a way of memorizing the past occurrence of the event e_d to be diagnosed. Formally, $Diag(G)$ equals $Obs(G||A_{label})$, where A_{label} is a two-state automaton with state set $\{N, Y\}$. Label N represents that event e_d has not occurred. Upon an occurrence of e_d, A_{label} transitions to its state Y, which represents that e_d has occurred, and then it stays in that state.

The problems mentioned above have been investigated extensively in the literature over the last 20 years for both automaton and Petri net models. Zaytoon & Lafortune (10) provided a thorough survey of the state of the art as of 2013, Lafortune et al. (11) made historical remarks on the development of diagnosability theory for DESs, and Sears & Rudie (12) reviewed sensor

activation in general (not only for diagnosability). Here, we discuss some recent research trends and challenges.

One research trend is to study the diagnosability properties of infinite-state systems, modeled by unbounded Petri nets; a Petri net is unbounded if the token count in one or more places can grow arbitrarily large. Several works have tackled this problem, using different approaches that either adapt techniques developed for diagnosability analysis of automata or exploit the representation of the Petri net state as a vector (see, e.g., 13–15). For infinite-state systems, the definition of diagnosability needs to be adjusted. In Equation 3, the first two clauses are swapped, so that the bound n may depend on the string s. This is because a uniform bound may not exist over all strings that contain e_d, unlike in the case of finite-state automata.

Quantifying the diagnosability properties of a system in terms of a probabilistic model is an important problem that is receiving increasing attention. Thorsley & Teneketzis (16) published an important paper that laid the groundwork in this area and developed two types of diagnosability conditions. Recently, many groups have further developed this line of research (see, e.g., 17, 18).

Finally, some recent works have leveraged the theory of diagnosability to address the detection of attacks on sensors and actuators in feedback control loops in cyber-physical systems (see Section 5.2). In this context, attacks on sensor readings that, for instance, erase a genuine event sent from the sensor to the supervisory controller are modeled as unobservable events. The defender's goal is to make such events diagnosable, while the attacker's goal is to keep its attack stealthy by avoiding such diagnosis.

3.2. Opacity

Opacity is an information flow property that captures the ability to keep some information about the system secret from an eavesdropper, termed an intruder, that knows the structure of the system and monitors its observable behavior. The information is kept secret in the sense that it is never known for sure by the intruder. Let us state the definition of current-state opacity. The context is that we have a partially observed deterministic automaton G and a set of secret states $X_{\text{secret}} \subset X$. Current-state opacity holds if the following condition is true:

$$\forall t \in \mathcal{L}(G), \exists s \in \mathcal{L}(G) : f(x_0, t) \in X_{\text{secret}} \Rightarrow [f(x_0, s) \notin X_{\text{secret}} \text{ and } P_o(s) = P_o(t)]. \quad 5.$$

In words, if string t reveals the secret (by ending in a secret state of G), then there must exist a distinct string s that does not reveal the secret and is observationally equivalent. Hence, upon observing string $P_o(t)$, the intruder's estimate of the state of the system will contain both a secret state $[f(x_0, t)]$ and a nonsecret state $[f(x_0, s)]$. Indeed, the verification of current-state opacity is easily done using $Obs(G)$: It holds if no state of $Obs(G)$ is contained in X_{secret}.

By changing the form of the secret, other notions of opacity can be defined, such as initial-state opacity (X_{secret} is a subset of the set of initial states) and language-based opacity (there is a secret language) (see 19, 20). Moreover, the inferences made by the intruder need not be only for the current-state estimate; they might involve smoothing, i.e., inferences about past states using observations up to the present. The notions of K-step and infinite-step opacity fall into this category.

Opacity is an active research area in DESs. As it is a very general notion, opacity can be adapted to model a variety of privacy and security issues that arise in modern technological systems. Similarly to work on diagnosability, research efforts on opacity are centered around (*a*) the efficient verification of various notions of opacity, both for finite-state automata and for Petri nets that model DESs with infinite state spaces, and (*b*) the enforcement of opacity for nonopaque systems, using a variety of techniques. Jacob et al. (21) provided an excellent overview of the research on opacity up to 2016; the historical remarks by Lafortune et al. (11) also cover the emergence of this avenue of research in DESs.

4. CONTROL OF DISCRETE EVENT SYSTEMS: FULL OBSERVATION

The synthesis of a supervisor (or controller) for a DES whose transitions are fully observed is a well-understood problem for a large class of specifications. This section briefly discusses the salient features of two synthesis approaches, which in fact have much in common. The discussion is at a conceptual level, and the reader is referred to the recent literature for formal presentations of the topics discussed.

4.1. Supervisory Control Theory

In supervisory control theory, initiated in seminal work by Ramadge and Wonham (see 22–25), the DES is modeled as a deterministic automaton, G, and the specification for the control problem is expressed as another deterministic automaton, H, which is assumed to be trim (accessible and coaccessible). The control mechanism involves a supervisor, denoted by S, that dynamically enables or disables the controllable events of G. The set of controllable events $E_c \subseteq E$ is identified at modeling time and represents the available actuation capabilities of the system. The set $E_{uc} = E \setminus E_c$ is the set of uncontrollable events. The supervisor S is a function from strings in $\mathcal{L}(G)$ to control actions (enabled events by convention) in the set 2^E. The controlled system is denoted by S/G and can be interpreted as another DES, whose languages generated [$\mathcal{L}(S/G)$] and marked [$\mathcal{L}_m(S/G)$] are subsets of $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$, respectively, due to the disablement of transitions of G.

In practice, the function S is encoded, or realized, as another (deterministic) automaton R, where it is required that R be trim and that all its states be marked. Moreover, $E_R = E$ by convention. The control actions of S are then effectively captured by the rules of parallel composition $R||G$, subject to the controllability condition: S should never disable a feasible uncontrollable event in G, i.e., if the composed system $R||G$ is at state (x_R, x_G), then $\Gamma_G(x_G) \cap E_{uc} \subseteq \Gamma_R(x_R)$. Under these conditions, we have that $\mathcal{L}(S/G) = \mathcal{L}(G) \cap \mathcal{L}(R)$ and $\mathcal{L}_m(S/G) = \mathcal{L}_m(G) \cap \mathcal{L}(R) = \mathcal{L}_m(G) \cap \mathcal{L}(S/G)$. Thus, one can think of the controlled system S/G as the automaton $R||G$.

The language generated by H captures the desired safety specification for the controlled system. Specifically, $\mathcal{L}(H) \subseteq \mathcal{L}(G)$ is the set of legal behaviors imposed on the uncontrolled behavior $\mathcal{L}(G)$. For the sake of simplicity, assume that $E_H = E$ and that all the states of H are marked. The nonblocking specification is that the controlled system S/G should be nonblocking—i.e., $R||G$ should be nonblocking. Observe that in our setup, marking of states of G is what matters for nonblockingness, since all states of H are marked.

When both G and H have finite state spaces, effective algorithms exist to automatically synthesize R from G and H. First, we must form the product system $H||G$ in order to have the right discrete structure over which state pruning can be done to obtain the answer. This is because we are imposing a language specification $\mathcal{L}(H)$ onto G. Note that $\mathcal{L}(H||G) = \mathcal{L}(H)$ since $\mathcal{L}(H) \subseteq \mathcal{L}(G)$. In the automaton $H||G$, state pairs are of the form (x_H, x_G), allowing us to identify the current state of G for any string in $\mathcal{L}(H)$. Pruning of states (and associated transitions) is done on $H||G$ in order to satisfy the controllability condition mentioned above and the nonblockingness condition on the pruned structure. This pruning of states is an iterative process, as deletion of states that violate controllability might cause blocking, and deletion of states to remove blocking might induce violations of controllability. At convergence, the realization R of the desired supervisor S is obtained, as a pruned version of $H||G$.

By construction, the controlled system S/G has four key properties: (*a*) controllability (S never disables an uncontrollable event), (*b*) safety [S/G generates only legal strings, i.e., $\mathcal{L}(S/G) \subseteq \mathcal{L}(H)$], (*c*) nonblockingness [S/G is nonblocking, i.e., $\overline{\mathcal{L}_m(S/G)} = \mathcal{L}(S/G)$], and (*d*) maximal permissiveness [$\mathcal{L}(S/G)$ cannot be made any larger without violating one of the three preceding properties].

Maximal permissiveness is an essential concept in supervisory control theory. A controlled system that does nothing is safe but useless in practice. Hence, we are looking for solutions that let the system G execute as much as possible. Since we are working with languages, the optimality criterion used is set inclusion. A fundamental result of supervisory control theory is that there is a unique largest solution satisfying all four conditions above. It is called the supremal controllable sublanguage [of $\mathcal{L}(H) \cap \mathcal{L}_m(G)$ in our setup] with respect to $\mathcal{L}(G)$ and E_{uc}. This is the solution returned by the above-described pruning procedure. The solution returned may be the empty set, which means that no solution to the posed supervisory control problem exists; in this case, one cannot even turn the system on, as it will uncontrollably transition to blocking states or to unsafe states from the initial state even if all controllable events are disabled.

The theory of supervisory control in the framework of Ramadge and Wonham originated in the 1980s, and in the 1990s and 2000s, researchers focused more on the development of the theory than on applications. One important reason for that was the lack of software tools that could handle the large state spaces in real applications. However, the current decade has seen an emergence of new applications in a range of domains, from automated manufacturing to transportation, process control, and software systems, to mention a few (see, e.g., 26–32).

Supervisory control theory is not the only (supervisory) control theory for DESs. For instance, other works have considered supervisory control of Petri nets (33, 34).

4.2. Reactive Synthesis

There is considerable interest in control science and engineering in the use of methodologies and computational techniques from formal methods (an area of computer science) to address the synthesis of controllers with provable correctness guarantees. This is especially true for cyber-physical systems with both logical and continuous variables (also known as hybrid systems), where the correctness guarantees are captured not only in terms of traditional quantitative criteria (e.g., stability) but also in terms of logical and temporal constraints that are expressed in some type of temporal logic (3). The area of formal methods in computer science encompasses not only verification problems (for hardware or software systems) but also synthesis problems (for controllers that implement given specifications). When synthesizing a controller for a system that interacts with its environment, the synthesis problem becomes one of reactive synthesis (for a rigorous introduction to reactive synthesis, see part I of Reference 35). Reactive synthesis techniques from formal methods in computer science are a key component of the work in formal methods in control science and engineering (see, e.g., 4, 36, 37). This work is closely related to work done in supervisory control of DESs, with some notable differences.

In reactive synthesis, the model of the uncontrolled system (i.e., the plant) either is absent or, if present, is captured in the form of temporal logic constraints on the environment or in the form of an explicit (labeled) transition system. This transition system is often obtained by abstracting the continuous dynamics and continuous states of the underlying cyber-physical system (38). Arguably, the most significant difference between reactive synthesis and supervisory control is the way the specifications on the controlled system are expressed. Most of supervisory control is concerned with regular language specifications, whereas most of reactive synthesis is concerned with temporal logic specifications. Linear temporal logic, computation tree logic, or some fragment of those are usually employed to capture safety and liveness specifications.

The most common computational solution procedure in reactive synthesis is conceptually related to the algorithm for the supremal controllable sublanguage discussed above. The temporal logic formula is translated into an equivalent automaton on infinite strings (e.g., a Büchi or Rabin automaton), the product of this automaton with the transition system is taken, and finally the

winning region is extracted from the product structure (by fixpoint algorithms). The term winning region comes from the fact that the product structure defines a game between the controller (to be synthesized) and the environment (captured by the plant model and other assumptions) (see 4).

Regular language specifications are usually specified in terms of star languages (i.e., subsets of E^*), while temporal logic specifications correspond to omega languages (i.e., languages of infinite strings or subsets of E^ω). Despite this difference, we can map the nonblockingness condition of Section 4.1 to a temporal logic formula in computation tree logic as follows. If we associate the propositional variable *acc* with the marked states of G, then nonblockingness corresponds to declaring $AGEF(acc)$, where A stands for the universal quantifier, G is the "globally" operator in temporal logic, E stands for the existential quantifier, and F is the "eventually" operator in temporal logic. Using this correspondence, one can map the supervisory control problem of the preceding section to an instance of a specific type of reactive synthesis problem, captured by the formula $AGEF(acc)$; this was formally done by Ehlers et al. (39). A further connection between the theory of supervisory control for infinite strings (not discussed in this article) and reactive synthesis was presented by Schmuck et al. (40).

Much is to be gained by further bridging the gap between supervisory control and reactive synthesis. On the one hand, the theory of supervisory control has identified several DES theoretic properties and horizontal and vertical decomposition techniques that could be exploited in reactive synthesis solution procedures. On the other hand, the research in reactive synthesis has led to the development of efficient computational procedures for controller synthesis that could be leveraged in supervisory control.

4.3. Challenges

It is fair to say that the primary challenge faced in the application of the theory of supervisory control, and in fact in the application of reactive synthesis methodologies for discrete systems in general, is one of computational scalability. Complex technological systems involve several interacting components, and composition by parallel composition causes, in the worst case, an exponential growth of the state space in the number of system components. The state space explosion problem manifests itself not only in the offline synthesis of supervisors but also in their online implementation. Below, we discuss three facets of this problem and highlight some recent attempts at dealing with scalability. This is not meant to be an exhaustive list of the ways to tackle the state space explosion problem; quite often, the best approach is problem dependent and involves exploiting structural properties of the system at hand.

4.3.1. Modular control. In general, the model of a DES is obtained not in a monolithic manner, but rather by modeling the system components individually and by capturing their coupling via common events. Thus, G is of the form $G = ||_{i=1}^{N} G_i$. Similarly, the safety specification is itself the conjunction of a set of safety specifications, each modeled by an automaton; thus, $H = ||_{i=1}^{M} H_i$. There may then be a global condition of nonblockingness on the controlled system.

Researchers have investigated numerous approaches for exploiting the structure of G and H in the verification of controllability and nonblockingness and in the synthesis of supervisors. One key challenge in modular synthesis is to preserve maximal permissiveness. Another is to guarantee nonblockingness, which is inherently a global property that cannot be modularized. It is possible for two individually nonblocking supervisors to block when they are used in conjunction.

Many works have considered compositional approaches where the specifications are considered one at a time and for part of the system components (to which they each pertain). Abstraction methods are often used at each step to reduce the size of intermediate results at no loss of generality.

This has given rise to the definition of several classes of abstractions that specialize or adapt the well-known bisimulation and weak bisimulation relations (3).

There is a very large literature on modular and compositional approaches. For interested readers, References 41–49 represent a sample of the work on this broad topic.

4.3.2. Supervisor representation. When a supervisor has been synthesized, feedback control is done by updating the state of the supervisor upon each (observable) event occurrence by the system. The active event set of the current supervisor state encodes the current control action (i.e., set of enabled events). This means that the supervisor is essentially stored as a lookup table that is queried at runtime. However, this table may be too large to be practically stored in a given implementation. For this reason, researchers have looked at different ways of encoding the control policy of a supervisor. One approach, which works for systems that possess certain structural properties, is to use a classifier to separate the set of safe states from the set of unsafe states, in a (possibly refined) state space representation of the system. In that case, at a given safe state, the supervisor enables transitions only to other safe states. Under certain conditions, the classifier can be a set of linear inequalities over a numerical representation of the state space, which works well if the system model can be converted to a Petri net. This approach was exploited by Nazeem et al. (50) and then generalized by Nazeem & Reveliotis (51).

4.3.3. Software tools. Several software tools are available for performing various calculations regarding analysis and control of DESs. Most of these tools are academic and were developed for educational purposes; examples include DESLAB (Federal University of Rio de Janeiro), DESUMA (University of Michigan and Mount Allison University), DESpot (McMaster University), IDES (Integrated Discrete Event Systems; Queen's University), libFAUDES (Friedrich-Alexander University Discrete Event Systems library; University of Erlangen-Nuremberg), Supremica (Chalmers University and University of Waikato), and TCT (University of Toronto). Some of these tools, such as Supremica, employ symbolic methods—namely, binary decision diagrams (52)—to store the transition functions of automata. The use of binary decision diagrams can lead to significant scalability gains (see 53, 54). Researchers have recently started to investigate how to leverage SAT solvers—powerful tools developed in computer science to solve satisfiability problems—to address DES analysis and synthesis problems (55, 56).

5. CONTROL OF DISCRETE EVENT SYSTEMS: PARTIAL OBSERVATION

Section 3 considered partially observed DESs and reviewed the two properties of diagnosability and opacity. This section presents the supervisory control problem for partially observed DESs and concludes with a discussion of cyber-security and privacy.

5.1. Synthesis of Partial-Observation Supervisors

The generalization of the supervisory control problem discussed in Section 4.1 to the case of partial observation was studied and its main features characterized soon after the original full-observation case (57–59). As in Section 3, the event set of the system E is partitioned as $E = E_o \cup E_{uo}$, with the associated projection P_o. This time, the supervisor sees $P_o(s)$ only when the system generates string s, due to the sensing limitations captured by E_{uo}. The controllability condition discussed in Section 4.1 and imposed on a supervisor remains in force, but the supervisor must additionally issue the same control decisions for events in E_c for two strings s and t that look the same to

it—i.e., $P_o(s) = P_o(t)$. This requirement is known as the observability condition, a name that does not fully capture its close connection to control in the context of supervisory control theory.

It was soon realized that the supremal controllable and observable sublanguage of a given language $\mathcal{L}(H)$ with respect to $\mathcal{L}(G)$, E_c, and E_o need not exist in general, in contrast to the full-observation case. However, under the condition $E_c \subseteq E_o$, the supremal controllable and observable sublanguage does exist, and algorithms for its computation are available (but are not discussed here; for further details, see the treatment of the supremal controllable normal sublanguage in Reference 1). This means that, in the general case, there does not exist a unique solution that is safe, nonblocking, and as permissive as any other safe and nonblocking solution. In that case, it is often of interest to synthesize maximal solutions, under the partial order of set inclusion, especially if there are problem-specific criteria that can guide the selection of one such maximal element. While different methods were proposed for the synthesis of maximal safe supervisors when nonblockingness is relaxed (see, e.g., 60), it was not until recently that the synthesis of maximal safe and nonblocking supervisors was solved in full generality (see 61).

It is worth discussing the solution methodology of Yin & Lafortune (61), as it departs from earlier attempts at solving this problem. Instead of a language-based approach to the problem, as in the original work on observability and the recent work on a new, stronger condition called relative observability (62), the approach adopted by Yin & Lafortune (61) consists of constructing a discrete game structure that captures all possible control actions of a partial-observation supervisor (on the one hand) and all possible system events under such control actions (on the other hand). This game structure is essentially a bipartite graph and is often called a game graph (or arena); work done in reactive synthesis under imperfect information follows a related approach (63). In the methodology developed by Yin & Lafortune (61), there are S-nodes, where the supervisor selects a control action, and E-nodes, where the system, acting as the environment, executes an enabled event. For the purpose of this construction, it is assumed that the original automaton G (system) and H (specification) are used to refine G and obtain G_r such that the refined state space, denoted by X_r, can be partitioned into legal and illegal states: $X_r = X_{legal} \cup X_{ill}$, where $X_{legal} \cap X_{ill} = \emptyset$. Such preprocessing is commonly done to transform a language-based specification into a state-based specification (see 59–61).

The definition of each node in the game structure is based on the notion of information state in system theory—specifically, what necessary information about the past needs to be memorized in a state. In the case of an S-state, the supervisor needs to know the best estimate of the current state of the system on the basis of the past trajectory of control actions and events executed by the system. Hence, an S-state is a subset IS of the set of system states X_r: $IS \subseteq X_r$. Transitions from S-states to E-states are labeled by control actions that are admissible in the current S-state: All uncontrollable events must be enabled, and a subset of E_c is enabled. Let γ denote such a control action. We claim that the right information state for the resulting E-state upon transition labeled γ from S-state IS is $[UR_\gamma(IS), \gamma]$, where $E_{IS} := UR_\gamma(IS)$ is the unobservable reach of set IS restricted to the unobservable events in the set γ (since only those are enabled). Note that we must remember in the E-state what control action is in effect at that time, which is why γ is included as the second component of the E-state. With such information, the transitions from this E-state will be all the enabled feasible observable events in all the states in E_{IS}; for each such event, we obtain a new information state (i.e., a new S-state), as is done in the construction of the observer automaton. Thus, the transitions from E-states to S-states are labeled by observable events. Overall, the game graph so constructed will be deterministic.

In the above construction, we include for each "player" (supervisor and system) all their possible "moves": all possible control actions at an S-state and all possible events at an E-state. However, we stop the construction at any E-state where $E_{IS} \cap X_{ill} \neq \emptyset$ and flag that E-state as illegal. We then

prune the resulting game graph to remove all illegal E-states, with two requirements: An S-state must always have at least one successor, unless all the states in it are terminating in the system, and there must be a transition for every feasible enabled event out of an E-state, as the supervisor cannot prevent the system from executing an enabled event. The first requirement captures the fact that the supervisor must always have a valid move in this game; S-states that violate this condition must also be pruned. The resulting game graph after such iterative pruning is often called the winning region of the game (recall the discussion in Section 4.2), as the supervisor has at least one valid move at each time, for all possible moves of the system. In fact, all valid moves of the supervisor at each S-state are included in the winning region obtained; for this reason, Yin & Lafortune (61) called it the All Inclusive Controller (AIC).

The game construction procedure can be interpreted, from the viewpoint of the supervisor, as a worst-case analysis, since it considers all system moves (under the current control action). While it addresses the safety specification, through the set X_{ill}, it does not explicitly address nonblockingness, since marking of states in G has not been taken into account so far. The methodology of Yin & Lafortune (61) proceeds by further pruning the AIC to obtain what is termed the Nonblocking AIC (NB-AIC), which is such that each system state in an S-state has a path to a marked system state in a future S-state. A safe and nonblocking supervisor can then be extracted from the NB-AIC by a process that may require unfolding some cycles in the NB-AIC. Moreover, this extraction process guarantees the maximality of the solution (in the language sense, with set inclusion as partial order). The details are omitted here, since they are beyond the scope of the discussion. The important point is that there is a finitely convergent algorithm that can extract a supervisor from the NB-AIC that is provably safe, nonblocking, and maximally permissive, thereby solving the supervisory control problem under partial observation when $E_c \not\subseteq E_o$.

In the absence of a unique supremal solution, in general, one might ask which maximal solution should be extracted from the NB-AIC, since maximal solutions are by definition not comparable. This problem is application dependent, as application-specific considerations might dictate the choice of a maximal solution. A generic approach to this problem is to consider a lower bound for the controlled system behavior and extract a maximal solution that provably contains this lower bound (if one exists). An algorithmic procedure for this approach was presented by Yin & Lafortune (64).

Section 3 discussed the two properties of diagnosability and opacity for partially observed DESs. Enforcing these properties by supervisory control is a viable strategy if restricting the behavior of the system is allowed. Specifically, supervisory control can be used to delete strings that violate diagnosability or opacity. Hence, these control problems fall in the class of problems discussed in this section, and they can be approached by the methodology presented by Yin & Lafortune (65), subject to technical details specific to each property. Specifically, the safety condition previously captured by the set X_{ill} is replaced by a condition on the first component of E-states that captures either diagnosability or opacity. The case of opacity is straightforward: Illegal states are those that reveal the secret, such that E_{IS} contains only secret states. The case of diagnosability requires fixing a priori the bound n in Section 3.1 and considering n-diagnosability, which is expressible in terms of information states (for details, see 65). Nonblockingness is generally not a requirement here, so the AIC (with safety changed to diagnosability or opacity) would suffice.

5.2. Cyber-Security and Privacy

Cyber-security and privacy are increasingly important problems in cyber-physical systems (66). Attacks on control systems that can cause physical damage to a system have been demonstrated in a range of applications, from industrial control systems to intelligent transportation systems

(67, 68). At the same time, privacy considerations arise with the integration of cyber-physical systems into our daily lives. Hence, there is a need to develop analytical frameworks to study cyber-security and privacy in cyber-physical control systems (69, 70).

Researchers have started to investigate attacks on sensors and actuators at the supervisory layer of cyber-physical control systems (see, e.g., 71–75). It is assumed that since sensors and actuators operate in a networked environment, these devices may become compromised if an attacker can infiltrate them and affect the sensor readings sent to a supervisor or affect the control actions sent to actuators. Various approaches have been considered, such as looking at the robustness of a supervisor in the presence of attacks or running online diagnostic tools to detect an attack. The problem can also be investigated from the attacker's viewpoint: how to manipulate sensor readings or actuator commands in order to cause physical damage before being detected.

Discrete event modeling formalisms can capture attacks on sensors and actuators by incorporating suitably defined edit functions that alter the event stream in the channels from the sensors to the supervisor and from the supervisor to the actuators. The theories of diagnosability (Section 3.1) and supervisory control under partial observation (Section 5.1) can then be leveraged to study the effect of such attacks on the controlled system. Edit functions can, for instance, erase genuine sensor readings or actuator commands and/or insert fictitious ones. Hence, the system is effectively controlled by a combination of the original supervisor and the actions of the attacker. These problems are inherently control problems under partial observation, since unobservable events are used to capture attack moves, which can be viewed as unobservable fault events. However, unlike passive diagnostics, the attacker is often strategic, and it may need to be explicitly modeled as another supervisor acting on the system.

Regarding privacy, the notion of opacity (Section 3.2) provides a formal approach to tackle certain kinds of privacy requirements, such as obfuscating the exact position or trajectory of a user being tracked by location-based services. Here, the secret states would be certain locations, and the user would want to hide visits to them. Obfuscation would be achieved by inserting fictitious moves of the user to reveal an incorrect location to the eavesdropper, so that the secret locations remain opaque (76).

We expect to see significant research activities on problems of cyber-security and privacy using DES theoretic techniques in the coming years.

SUMMARY POINTS

1. Discrete event systems (DESs) arise in the modeling of dynamic systems that are inherently discrete (e.g., discrete manufacturing processes, software execution, and computer protocols) as well as when constructing discrete abstractions of cyber-physical systems (e.g., transportation, process control, and autonomous systems).
2. DESs can be analyzed at different levels of abstraction (logical, temporal, and stochastic); each level has its own set of modeling formalisms and associated analytical techniques.
3. The properties of diagnosability and opacity for partially observed or nondeterministic DESs provide general frameworks for studying observation problems using model-based inferencing.
4. Supervisory control of DESs in control engineering is related to reactive synthesis in computer science, as both are formal, model-based approaches for the synthesis of controllers that must provably satisfy various safety and liveness properties.

5. The theory of supervisory control for partially observed DESs provides a general framework for synthesis of supervisors that enforce safety, nonblockingness, diagnosability, and opacity.
6. Studying privacy and security at the supervisory control layer of complex control systems for cyber-physical systems is an essential complement to other approaches to these problems that explicitly consider continuous-variable and time-driven representations of these systems.

FUTURE ISSUES

1. Additional research is needed on the scalability of algorithmic procedures in supervisory control, exploiting system structure.
2. Software tools for supervisory control need to be developed that are both user-friendly and scalable.
3. The supervisory layer of cyber-physical control systems requires security analysis.
4. The use of discrete event models and techniques for privacy analysis and enforcement should be further developed.

DISCLOSURE STATEMENT

The author is not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

The research of the author is supported primarily by grants from the US National Science Foundation.

LITERATURE CITED

1. Cassandras CG, Lafortune S. 2008. *Introduction to Discrete Event Systems*. New York: Springer. 2nd ed.
2. Seatzu C, Silva M, van Schuppen J, eds. 2013. *Control of Discrete Event Systems: Automata and Petri Net Perspectives*. London: Springer
3. Baier C, Katoen JP. 2008. *Principles of Model Checking*. Cambridge, MA: MIT Press
4. Belta C, Yordanov B, Gol EA. 2017. *Formal Methods for Discrete-Time Dynamical Systems*. Cham, Switz.: Springer
5. Tabuada P. 2009. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Dordrecht, Neth.: Springer
6. Murata T. 1989. Petri nets: properties, analysis and applications. *Proc. IEEE* 77:541–80
7. Giua A, Silva M. 2018. Petri nets and automatic control: a historical perspective. *Annu. Rev. Control* 45:223–39
8. Alur R, Dill DL. 1994. A theory of timed automata. *Theor. Comput. Sci.* 126:183–235
9. Sampath M, Sengupta R, Lafortune S, Sinnamohideen K, Teneketzis D. 1995. Diagnosability of discrete event systems. *IEEE Trans. Autom. Control* 40:1555–75

10. Zaytoon J, Lafortune S. 2013. Overview of fault diagnosis methods for discrete event systems. *Annu. Rev. Control* 37:308–20
11. Lafortune S, Lin F, Hadjicostis C. 2018. On the history of diagnosability and opacity in discrete event systems. *Annu. Rev. Control* 45:257–66
12. Sears D, Rudie K. 2016. Minimal sensor activation and minimal communication in discrete-event systems. *Discrete Event Dyn. Syst. Theory Appl.* 26:295–349
13. Cabasino MP, Giua A, Seatzu C. 2010. Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica* 46:1531–39
14. Cabasino MP, Giua A, Lafortune S, Seatzu C. 2012. A new approach for diagnosability analysis of Petri nets using verifier nets. *IEEE Trans. Autom. Control* 57:3104–17
15. Cabasino MP, Giua A, Hadjicostis CN, Seatzu C. 2015. Fault model identification and synthesis in Petri nets. *Discrete Event Dyn. Syst. Theory Appl.* 25:419–40
16. Thorsley D, Teneketzis D. 2005. Diagnosability of stochastic discrete-event systems. *IEEE Trans. Autom. Control* 50:476–92
17. Bertrand N, Fabre E, Haar S, Haddad S, Hélouët L. 2014. Active diagnosis for probabilistic systems. In *Foundations of Software Science and Computation Structures*, ed. A Muscholl, pp. 29–42. Berlin: Springer
18. Chen J, Keroglou C, Hadjicostis CN, Kumar R. 2018. Revised test for stochastic diagnosability of discrete-event systems. *IEEE Trans. Autom. Sci. Eng.* 15:404–8
19. Saboori A, Hadjicostis CN. 2007. Notions of security and opacity in discrete event systems. In *46th IEEE Conference on Decision and Control*, pp. 5056–61. New York: IEEE
20. Lin F. 2011. Opacity of discrete event systems and its applications. *Automatica* 47:496–503
21. Jacob R, Lesage JJ, Faure JM. 2016. Overview of discrete event systems opacity: models, validation, and quantification. *Annu. Rev. Control* 41:135–46
22. Ramadge PJ, Wonham WM. 1987. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* 25:206–30
23. Wonham WM, Ramadge PJ. 1987. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim.* 25:637–59
24. Ramadge PJ, Wonham WM. 1989. The control of discrete event systems. *Proc. IEEE* 77:81–98
25. Wonham WM, Cai K, Rudie K. 2018. Supervisory control of discrete-event systems: a brief history. *Annu. Rev. Control* 45:250–56
26. Moor T, Schmidt K, Perk S. 2010. Applied supervisory control for a flexible manufacturing system. In *10th IFAC Workshop on Discrete Event Systems*, pp. 253–58. IFAC Proc. Vol. 43(12). Amsterdam: Elsevier
27. Forschelen STJ, van de Mortel-Fronczak JM, Su R, Rooda JE. 2012. Application of supervisory control theory to theme park vehicles. *Discrete Event Dyn. Syst. Theory Appl.* 22:511–40
28. Liao H, Wang Y, Stanley J, Lafortune S, Reveliotis S, et al. 2013. Eliminating concurrency bugs in multithreaded software: a new approach based on discrete-event control. *IEEE Trans. Control Syst. Technol.* 21:2067–82
29. Theunissen RJM, Petreczky M, Schiffelers RRH, van Beek DA, Rooda JE. 2014. Application of supervisory control synthesis to a patient support table of a magnetic resonance imaging scanner. *IEEE Trans. Autom. Sci. Eng.* 11:20–32
30. Rawlings BC, Christenson B, Wassick JM, Ydstie BE. 2014. Supervisor synthesis to satisfy safety and reachability requirements in chemical process control. In *12th IFAC International Workshop on Discrete Event Systems*, pp. 195–200. IFAC Proc. Vol. 47(2). Amsterdam: Elsevier
31. Atampore F, Dingel J, Rudie K. 2016. Automated service composition via supervisory control theory. In *2016 13th International Workshop on Discrete Event Systems (WODES)*, pp. 28–35. New York: IEEE
32. Reijnen FFH, Goorden MA, van de Mortel-Fronczak JM, Rooda JE. 2017. Supervisory control synthesis for a waterway lock. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 1562–63. New York: IEEE
33. Moody J, Antsaklis P. 1998. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Boston: Kluwer Acad.
34. Iordache MV, Antsaklis PJ. 2006. *Supervisory Control of Concurrent Systems: A Petri Net Structural Approach*. Boston: Birkhäuser

35. Ehlers R. 2013. *Symmetric and efficient synthesis*. PhD Thesis, Saarland Univ., Saarbrücken, Ger.
36. Liu J, Ozay N, Topcu U, Murray R. 2013. Synthesis of reactive switching protocols from temporal logic specifications. *IEEE Trans. Autom. Control* 58:1771–85
37. Kress-Gazit H, Lahijanian M, Raman V. 2018. Synthesis for robots: guarantees and feedback for robot behavior. *Annu. Rev. Control Robot. Auton. Syst.* 1:211–36
38. Alur R, Henzinger TA, Lafferriere G, Pappas GJ. 2000. Discrete abstractions of hybrid systems. *Proc. IEEE* 88:971–84
39. Ehlers R, Lafortune S, Tripakis S, Vardi MY. 2017. Supervisory control and reactive synthesis: a comparative introduction. *Discrete Event Dyn. Syst. Theory Appl.* 27:209–60
40. Schmuck AK, Moor T, Majumdar R. 2018. On the relation between reactive synthesis and supervisory control of input/output behaviours. In *14th IFAC Workshop on Discrete Event Systems*, ed. G De Tommasi, pp. 31–38. IFAC-PapersOnLine 51(7). Amsterdam: Elsevier
41. Leduc RJ, Brandin BA, Lawford M, Wonham WM. 2005. Hierarchical interface-based supervisory control—part I: serial case. *IEEE Trans. Autom. Control* 50:1322–35
42. Flordal H, Malik R, Fabian M, Åkesson K. 2007. Compositional synthesis of maximally permissive supervisors using supervision equivalence. *Discrete Event Dyn. Syst. Theory Appl.* 17:475–504
43. Feng L, Wonham WM. 2008. Supervisory control architecture for discrete-event systems. *IEEE Trans. Autom. Control* 53:1449–61
44. Schmidt K, Moor T, Perk S. 2008. Nonblocking hierarchical control of decentralized discrete event systems. *IEEE Trans. Autom. Control* 53:2252–65
45. Hill RC, Tilbury DM, Lafortune S. 2010. Modular supervisory control with equivalence-based abstraction and covering-based conflict resolution. *Discrete Event Dyn. Syst. Theory Appl.* 20:139–85
46. Su R, van Schuppen JH, Rooda JE. 2010. Aggregative synthesis of distributed supervisors based on automaton abstraction. *IEEE Trans. Autom. Control* 55:1627–40
47. Gummadi R, Singh N, Sreenivas RS. 2011. On tractable instances of modular supervisory control. *IEEE Trans. Autom. Control* 56:1621–35
48. Komenda J, Masopust T, van Schuppen JH. 2012. Supervisory control synthesis of discrete-event systems using a coordination scheme. *Automatica* 48:247–54
49. Mohajerani S, Malik R, Fabian M. 2014. A framework for compositional synthesis of modular nonblocking supervisors. *IEEE Trans. Autom. Control* 59:150–62
50. Nazeem A, Reveliotis S, Wang Y, Lafortune S. 2011. Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: the linear case. *IEEE Trans. Autom. Control* 56:1818–33
51. Nazeem A, Reveliotis S. 2012. Designing compact and maximally permissive deadlock avoidance policies for complex resource allocation systems through classification theory: the nonlinear case. *IEEE Trans. Autom. Control* 57:1670–84
52. Bryant RE. 1992. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.* 24:293–318
53. Fei Z, Miremadi S, Åkesson K, Lennartson B. 2014. Efficient symbolic supervisor synthesis for extended finite automata. *IEEE Trans. Control Syst. Technol.* 22:2368–75
54. Rawlings BC, Wassick JM, Ydstie BE. 2017. Application of formal verification and falsification to large-scale chemical plant automation systems. *Comput. Chem. Eng.* 114:211–20
55. Claessen K, Een N, Sheeran M, Sörensson N, Voronov A, Åkesson K. 2009. SAT-solving in practice, with a tutorial example from supervisory control. *Discrete Event Dyn. Syst. Theory Appl.* 19:495
56. Shoaei MR, Kovács L, Lennartson B. 2014. Supervisory control of discrete-event systems via IC3. In *Hardware and Software: Verification and Testing*, ed. E Yahav, pp. 252–66. Berlin: Springer
57. Lin F, Wonham W. 1988. On observability of discrete-event systems. *Inf. Sci.* 44:173–98
58. Cieslak R, Desclaux C, Fawaz AS, Varaiya P. 1988. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Autom. Control* 33:249–60
59. Cho H, Marcus SI. 1989. On supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation. *Math. Control Signals Syst.* 2:47–69
60. Hadj-Alouane NB, Lafortune S, Lin F. 1996. Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation. *Discrete Event Dyn. Syst. Theory Appl.* 6:379–427

61. Yin X, Lafortune S. 2016. Synthesis of maximally permissive supervisors for partially observed discrete event systems. *IEEE Trans. Autom. Control* 61:1239–54
62. Cai K, Zhang R, Wonham WM. 2015. Relative observability of discrete event systems and its supremal sublanguages. *IEEE Trans. Autom. Control* 60:659–70
63. Chatterjee K, Doyen L, Henzinger TA, Raskin JF. 2006. Algorithms for omega-regular games with imperfect information. In *Computer Science Logic*, ed. Z Ésik, pp. 287–302. Berlin: Springer
64. Yin X, Lafortune S. 2018. Synthesis of maximally permissive nonblocking supervisors for the lower bound containment problem. *IEEE Trans. Autom. Control* 63:4435–41
65. Yin X, Lafortune S. 2016. A uniform approach for synthesizing property-enforcing supervisors for partially-observed discrete-event systems. *IEEE Trans. Autom. Control* 61:2140–54
66. Sandberg H, Amin S, Johansson KH. 2015. Cyberphysical security in networked control systems: an introduction to the issue. *IEEE Control Syst.* 35:20–23
67. Farwell JP, Rohozinski R. 2011. Stuxnet and the future of cyber war. *Survival* 53:23–40
68. Chen QA, Yin Y, Feng Y, Mao ZM, Liu HX. 2018. *Exposing congestion attack on emerging connected vehicle based traffic signal control*. Paper presented at the 25th Network and Distributed System Security Symposium, San Diego, CA, Feb. 18–21. **https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_01B-2_Chen_paper.pdf**
69. Teixeira A, Pérez D, Sandberg H, Johansson KH. 2012. Attack models and scenarios for networked control systems. In *Proceedings of the 1st International Conference on High Confidence Networked Systems*, pp. 55–64. New York: ACM
70. Han S, Pappas GJ. 2018. Privacy in control and dynamical systems. *Annu. Rev. Control Robot. Auton. Syst.* 1:309–32
71. Thorsley D, Teneketzis D. 2006. Intrusion detection in controlled discrete event systems. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 6047–54. New York: IEEE
72. Wakaiki M, Tabuada P, Hespanha JP. 2017. Supervisory control of discrete-event systems under attacks. arXiv:1701.00881 [cs.SY]
73. Su R. 2017. A cyber attack model with bounded sensor reading alterations. In *2017 American Control Conference (ACC)*, pp. 3200–5. New York: IEEE
74. Meira Góes R, Kang E, Kwong R, Lafortune S. 2017. Stealthy deception attacks for cyber-physical systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 4224–30. New York: IEEE
75. Lima PM, Carvalho LK, Moreira MV. 2018. Detectable and undetectable network attack security of cyber-physical systems. In *14th IFAC Workshop on Discrete Event Systems*, ed. G De Tommasi, pp. 179–83. IFAC-PapersOnLine 51(7). Amsterdam: Elsevier
76. Meira Góes R, Rawlings BC, Recker N, Willett G, Lafortune S. 2018. Demonstration of indoor location privacy enforcement using obfuscation. In *14th IFAC Workshop on Discrete Event Systems*, ed. G De Tommasi, pp. 145–51. IFAC-PapersOnLine 51(7). Amsterdam: Elsevier