SUPERVISORY CONTROL OF DISCRETE EVENT SYSTEMS

Stéphane Lafortune
The University of Michigan, USA

Keywords: Controllability, controller synthesis, nonblocking, observability, supervisor, supremal controllable sublanguage.

Contents

Summary

An overview of key results of the theory of supervisory control for untimed discrete event systems is presented. The notions of uncontrollable and unobservable events are introduced and the control paradigm of supervisory control theory is described. The properties of controllability and observability are shown to arise as necessary and sufficient conditions for the existence of supervisors that achieve a given desired behavior. The automated synthesis of supervisors from automata models of the uncontrolled system and of the desired behavior is considered. The algebraic properties

of controllability and observability are discussed and design techniques for dealing with uncontrollability and unobservability are described.

1. Introduction

This chapter is concerned with the problem of controlling the behavior of a discrete event system in order to achieve a given set of qualitative objectives. The system is modeled as the generator of a formal language to which control aspects are added in the sense that certain events (i.e., transitions) can be disabled by an external controller. In general, the controller may not be able to observe all the events that occur in the system. Since the mid-1980's, several researchers have built on and extended the seminal work of Ramadge and Wonham and their co-workers, who initiated this general control paradigm for discrete event systems. The resulting system and control theory for discrete event systems is known as "supervisory control theory." This theory addresses the synthesis of controllers (also called supervisors) for discrete event systems in order to satisfy a set of logical specifications on the admissible orderings of the events that can be executed by the system. The main results of this theory include: (i) characterization of fundamental system-theoretic properties of discrete event systems in the framework of formal languages and (ii) development of controller synthesis algorithms for discrete event systems in the framework of automata. This theory has been extended to timed and stochastic models of discrete event systems; however, logical specifications and untimed models are the focus of this chapter.

There exist other control theories for discrete event systems, especially for systems modeled as Petri nets or modeled using formal logics. However, supervisory control theory is arguably the most complete of these theories regarding its ability to deal with partial event controllability and partial event observability. Moreover, since the system-theoretic concepts of this theory are posed in the framework of formal languages, they are applicable to any discrete-event modeling formalism. These considerations have motivated the choice of supervisory control theory as the theme of this chapter.

1.3. Uncontrolled System

Assume that the given discrete event system is modeled by automaton G,

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

where: X is the state space of G (it need not be finite); E is the finite event set of G; f is the (partial) state transition function: $f : X \times E \to X$; Γ is the feasible event function: $\Gamma : X \to E$; x_0 is the unique initial state; and X_m is the subset of X that contains the marked states. (Familiarity with *Modeling of Discrete Event Systems* and the notation therein is assumed in the remainder of this chapter.)

Automaton G models the "uncontrolled behavior" of the discrete event system: (i) the language generated by G is denoted by $L(G)$; (ii) the language marked by G is denoted by $L_m(G)$. The premise is that this behavior is not satisfactory and must be modified by control; modifying the behavior is to be understood as *restricting* the

behavior to a subset of $L(G)$. In order to alter the behavior of G a *supervisor* is introduced; supervisors are denoted by S. Note that the "plant" G is separated from the "controller" (or supervisor) S, as is customary in control theory. (We shall use the terms controller and supervisor for S interchangeably.)

The language $L(G)$ contains strings of events that are not acceptable because they violate some condition imposed on the system. It could be that certain states of G are undesirable and should be avoided. These could be states where G blocks, via deadlock or livelock; or they could be states that are physically inadmissible, for example, a collision of a robot with an automated guided vehicle or an attempt to place a part in a full buffer in an automated manufacturing system. Moreover, it could be that some strings in $L(G)$ contain substrings that are not allowed. These substrings may violate a desired ordering of certain events, for example, requests for the use of a common resource should be granted in a "first-come first-served" manner. Control objectives that are expressed in terms of illegal states in G and/or illegal substrings in $L(G)$ are referred to as *safety specifications*. Thus, *sublanguages* of $L(G)$ that represent the "legal" or "admissible" behavior for the controlled system will be considered. In addition, absence of deadlocks and livelocks in the controlled system may be required; these objectives are known as *liveness specifications*.

1.4. Feedback Control

Consider a general control paradigm for how S interacts with G. In this paradigm, S sees (or observes) some, possibly all, of the events that G executes. Then, S tells G which events in the current feasible event set of G are allowed next. More precisely, S has the capability of disabling some, but not necessarily all, feasible events of G. The decision about which events to disable will be allowed to change whenever S observes the execution of a new event by G. In this manner, S exerts *dynamic feedback control* on G.

In order to capture the limited actuation capabilities that the supervisor S has over the system G, the event set E is partitioned into two disjoint subsets

$$E = E_c \cup E_{uc}$$

where: E_c is the set of *controllable* events: these are the events that can be prevented from happening, or disabled, by supervisor S; and E_{uc} is the set of *uncontrollable* events: these events cannot be prevented from happening by supervisor S. By definition, S cannot disable an uncontrollable event.

Similarly, in order to capture the limited sensing capabilities that S has with respect to the behavior of G, the event set E is partitioned into two disjoint subsets

$$E = E_o \cup E_{uo}$$

where: E_o is the set of *observable* events: these are the events that can be seen by the

supervisor; E_{uo} is the set of *unobservable* events: these are the events that cannot be seen by the supervisor.

Observe that no specific assumptions are made about the relation between the controllability and observability properties of an event; an unobservable event could be controllable, an uncontrollable event could be observable, and so forth. (Think of a controller of traffic signals: letting a vehicle cross an intersection is a controllable event that need not be observable unless appropriate vehicle detectors are installed at the intersection.) The situation when E_o is a proper subset of E is called *control under partial observation.* In this case, it is customary to denote supervisors by S_P rather than simply S.

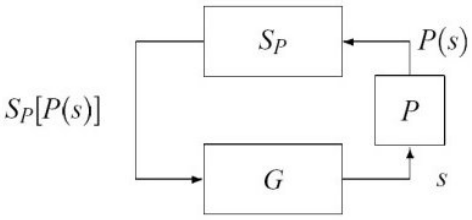

Figure 1. The feedback loop of supervisory control in the case of partial observation

The feedback loop for the general situation of control under partial observation is depicted in Fig. 1. This block diagram includes a block labeled P for "partial observation." The effect of each block on the next one in the diagram must be formalized. The operation P is defined as a projection operation from domain E^* to codomain E_o^*, $P : E^* \to E_o^*$. P erases the unobservable events in a string $s \in E^*$ to obtain a string $P(s) \in E_o^*$:

$$P(\varepsilon) := \varepsilon \qquad (1)$$

$$P(e) := \begin{cases} e & \text{if } e \in E_o \\ \varepsilon & \text{if } e \in E_{uo} \end{cases} \qquad (2), (3)$$

$$P(se) := P(s)P(e) \text{ for } s \in E^*, e \in E. \qquad (4)$$

(The symbol ε denotes the empty string.) It is straightforward to extend P to sets of strings, i.e., languages. Projection P is the only type of projection that is considered in this chapter.

Due to the presence of P, the supervisor cannot distinguish between two strings s_1 and s_2 that have the same projection, i.e., for which $P(s_1) = P(s_2)$; for such $s_1, s_2 \in L(G)$, the supervisor will necessarily issue the same control action, $S_P[P(s_1)]$. In order to capture this fact, a partial-observation supervisor is defined as a function

$S_P : P[L(G)] \to 2^E$, where 2^E denotes the power set of E, namely the set of all subsets of E. This means that the control action issued by S_P can change only after the occurrence of an observable event, namely, when $P(s)$ changes.

When an (enabled) observable event occurs, the control action is *instantaneously* updated. It is assumed that this update occurs before any unobservable event occurs. For each $s \in L(G)$ generated so far by G under the control of S_P,

$$S_P[P(s)] \cap \Gamma(f(x_0, s))$$

is the set of *enabled events* that G is allowed to execute at its current state $f(x_0, s)$ under the control of S_P. In other words, G cannot execute an event that is in its current feasible event set, $\Gamma(f(x_0, s))$, if that event is not also contained in $S_P[P(s)]$. $S_P[P(s)]$ is called the *control action* at s; S_P is the *control policy*.

The closed-loop behavior when S_P is controlling G as was just described is denoted by S_P/G. The behavior of the controlled system S_P/G is formally defined in the following manner:

- The *language generated* by S_P/G is defined recursively as the set of all strings that are allowed to occur under the control of S_P:

1. $\varepsilon \in L(S_P/G)$
2. $[(s \in L(S_P/G)) \text{ and } (s\sigma \in L(G)) \text{ and } (\sigma \in S_P[P(s)])] \Leftrightarrow [s\sigma \in L(S_P/G)]$.

- The *language marked* by S_P/G is defined as the set of marked strings that survive under the control of S_P:

$$L_m(S_P/G) := L(S_P/G) \cap L_m(G). \qquad (5)$$

It is important to note that the languages $L(S_P/G)$ and $L_m(S_P/G)$ are defined over E, and not E_o, i.e., they correspond to the closed-loop behavior of G before the effect of projection P, namely, as seen at the "output" of G in Fig. 1.

The above definitions of S_P, S_P/G, $L(S_P/G)$ and $L_m(S_P/G)$ specialize in the obvious manner to S, S/G, $L(S/G)$ and $L_m(S/G)$ in the case of full observation, when $E_o = E$.

Recall that an automaton H is said to be *nonblocking* if $L(H) = \overline{L_m(H)}$, namely, when every string generated by H can be extended to a string marked by H. (Recall that the "overbar" notation denotes the prefix-closure operation.) A similar property for controlled behaviors can be defined: supervisor S is said to be nonblocking if

$L(S/G) = \overline{L_m(S/G)}$. (Similarly for S_P/G.) This property is key to satisfying the liveness specifications mentioned earlier.

1.3. Industrial Applications of Supervisory Control Theory

The published literature contains many examples of the application of the concepts and techniques of supervisory control theory to many different areas, including: automated manufacturing, chemical process control, database management, document processing systems, heating, ventilation, and air-conditioning systems, industrial automation, intelligent transportation systems, and software systems for telephony and telecommunications.

These applications invariably involve models with a large number of states, making them impractical for simple illustrations of theoretical and algorithmic concepts. In order to get an intuitive understanding of some of the issues that arise in supervisor design, a simple example of an uncontrolled system with its associated specification is examined in the following section.

1.4. Supervisor Design: Illustrative Example

Let the uncontrolled system be modeled by the automaton G in Fig. 2. In the figure, the automaton is depicted as a directed graph where the initial state is identified by the arrow pointing into it and the marked states are identified by double circles.

The specification on G is given in terms of the marked language of G, $L_m(G)$: Control G in order to allow only the marked strings in $L_m(G)$, and their prefixes, where a_1 precedes a_2 if and only if b_1 precedes b_2.

The first step is to represent the desired behavior using an automaton model. Let the desired automaton be denoted by H_a (where the subscript a stands for "admissible"). A little thought shows that one needs to remember how state 4 of G is reached, i.e., this state in H_a should be split; number as 4 and 9 the two resulting states of H_a, where 4 is the state reached by string a_2a_1 and 9 is the state reached by string a_1a_2.

Then event b_1 should not occur in state 4 and event b_2 should not occur in state 9. The desired H_a is depicted in Fig. 2. As compared with G, H_a marks only strings where a_1 precedes a_2 if and only if b_1 precedes b_2.

Also, as required by the specification, the language generated by H_a contains only prefixes of strings marked by H_a; namely, $L(H_a) = \overline{L_m(H_a)}$, or H_a is nonblocking.

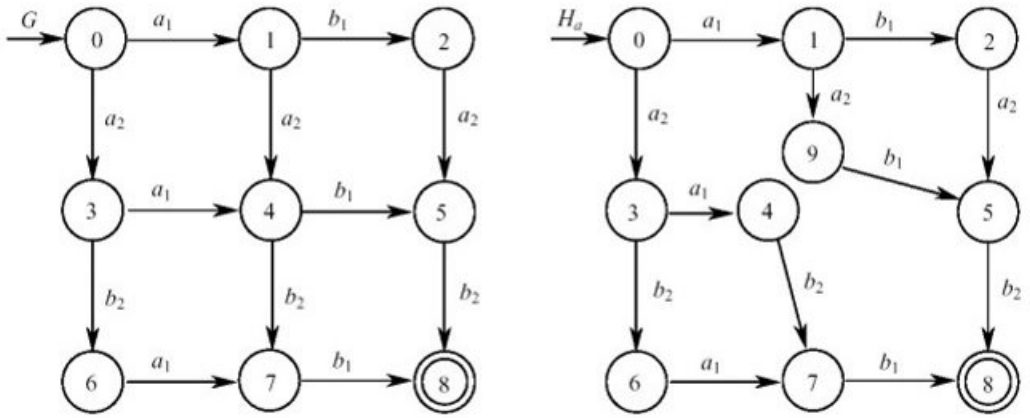

Figure 2. Automata G and H_a for illustrative example

What does a supervisor S need to do to guarantee that $L(S/G) \subseteq L(H_a)$ and that S/G is nonblocking? If all events are controllable and observable, then it is straightforward to determine what the desired supervisor, say S_1, should do. S_1 should use H_a to track the behavior of the system, namely, every event executed by G should cause S_1 to execute the same event in H_a and thus update the state of H_a. When H_a enters state 9, S should disable event b_2, and when H_a enters state 4, S_1 should disable event b_1; otherwise, S_1 enables all feasible events in G (as seen in the feasible events of the corresponding state in H_a). In this case, S_1/G achieves exactly H_a in the sense that $L(S_1/G) = L(H_a)$ and $L_m(S_1/G) = L_m(H_a)$.

To illustrate the effect of limited controllability, assume that $E_{uc} = \{a_2, b_2\}$. The supervisor for this case, denoted by S_2, can again use H_a to track the behavior of G. If S_2 allows state 9 of H_a to be reached, then it is too late to prevent inadmissible strings from being generated, since S_2 must disable event b_2 when H_a is in state 9, but b_2 is uncontrollable and thus cannot be disabled. The predecessor of state 9 in H_a is state 1, and state 1 has a transition to state 9 with uncontrollable event a_2. This means that S_2 should not allow state 1 of H_a to be reached either. Consequently, S_2 must disable event a_1 when H_a is in state 0. After observing event a_2, S_2 can enable a_1. Finally, S_2 should disable b_1 when H_a enters state 4. Consequently, the presence of uncontrollable events forces the behavior of the system to be restricted to a proper subset of $L(H_a)$, even though all of $L(H_a)$ is admissible: $L_m(S_2/G) = \{a_2a_1b_2b_1, a_2b_2a_1b_1\}$ and $L(S_2/G) = \overline{L_m(S_2/G)}$.

To illustrate the effect of limited observability, assume that $E_{uo} = \{a_2\}$. Let E_{uc} be unchanged: $E_{uc} = \{a_2, b_2\}$. Denote by S_3 the supervisor to design for this case. As was seen above, the uncontrollability of a_2 and b_2 force the disablement of event a_1 when

the system is "started" (i.e., when H_a is in state 0). Since a_2 is not observable, S_3 must also enable b_2 at the beginning, otherwise it would be disabling a feasible uncontrollable event, something that is not allowed in the control paradigm. (Recall that S_3 can only update its control action after the occurrence of an observable event.) Consequently, the next event that S_3 expects to see is b_2. Once it sees event b_2, S_3 can enable both a_1 and b_1. Overall, the only marked string allowed under control in this case is $a_2 b_2 a_1 b_1$. Thus the unobservability of a_2 forces the system behavior to be restricted to an even smaller subset of $L(H_a)$ than was obtained above for S_2/G.

The reader is encouraged to repeat this intuitive process of supervisor design for other combinations of E_{uc} and E_{uo}. (For instance, try $E_{uo} = \{a_2\}$ as above but set $E_{uc} = \varnothing$.) It quickly becomes obvious that even for this simple example, with only a few marked strings and at most 10 states, uncontrollability and unobservability can very rapidly complicate finding a supervisor that guarantees legality and nonblocking of the controlled system. Intuition may still allow one to design a correct supervisor for this example, but obviously, formal methods are required if one is to solve any problem with more than a few dozen states in G and in the automaton representation of the admissible language, in the presence of uncontrollable and/or unobservable events. The following sections in this chapter address more formally the effect of the presence of uncontrollable and unobservable events, respectively.

2. Control of Fully-Observed Discrete Event Systems

In this section, it is assumed that $E_o = E$, namely, there are no unobservable events. The presentation is focused on the implication of the presence of uncontrollable events on the classes of languages that can be achieved under supervision, according to the feedback loop of Fig. 1.

-
-
-

Bibliography

Brandin B.A., Wonham W.M. (1994). Supervisory control of timed discrete-event systems. *IEEE Trans. Automatic Control* **39(**2), 329-342. [This paper presents an extension of supervisory control theory to timed models of discrete-event systems].

Cassandras C.G., Lafortune S. (1999). *Introduction to Discrete Event Systems.* Kluwer Academic Publishers. [Chapter 3 of this textbook on discrete event systems presents a detailed coverage of the theory of supervisory control presented in this chapter; see the references therein for applications of

supervisory control theory].
Holloway L.E. Krogh B.H., Giua A. (1997). A survey of Petri net methods for controlled discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications* **7**(2), 151-190. [Comprehensive survey paper on the control of discrete event systems modeled by Petri nets].
Kumar R., Garg V.K. (1995). *Modeling and Control of Logical Discrete Event Systems.* Kluwer Academic Publishers. [A research monograph on the theory of supervisory control].
Maler O., Pnueli A., Sifakis J. (1995). On the synthesis of discrete controllers for timed systems. In *Proceedings of STACS' 95*, pp. 229-242, Springer Verlag, Vol. 900 of Lecture Notes in Computer Science. [This paper discusses controller synthesis for timed models of discrete event systems by finding a winning strategy for certain games defined by timed automata].
Marchand H., Bournai P., Le Borgne M., Le Guernic P. (2000). Synthesis of discrete-event controllers based on the Signal environment. *Discrete Event Dynamic Systems: Theory and Applications* **10**(4), 325-346. [This paper describes an environment for controller synthesis of discrete event systems based on polynomial dynamical system models and on the synchronous language SIGNAL].
Moody J., Antsaklis P.J. (1998). *Supervisory Control of Discrete Event Systems Using Petri nets.* Kluwer Academic Publishers. [This book introduces an approach to the synthesis of controllers for discrete event systems modeled by Petri nets using notions of place invariants].
Ostroff J.S. (1989). *Temporal Logic for Real-Time Systems.* Research Studies Press and John Wiley & Sons. [This book describes an approach for real-time control of discrete-event systems modeled by timed automata and where the specification is described using temporal logic].
Ramadge P.J., Wonham W.M. (1989). The control of discrete event systems. *Proceedings of the IEEE* **77**(1), 81-98. [Excellent survey paper introducing the theory of supervisory control initiated by the authors].

Biographical Sketch

Stéphane Lafortune received the B. Eng degree from École Polytechnique de Montréal in 1980, the M. Eng. degree from McGill University in 1982, and the Ph.D. degree from the University of California at Berkeley in 1986, all in electrical engineering. Since September 1986, he has been with the University of Michigan, Ann Arbor, where he is a Professor of Electrical Engineering and Computer Science. Dr. Lafortune is a Fellow of the IEEE (1999). He received the Presidential Young Investigator Award from the National Science Foundation in 1990 and the George S. Axelby Outstanding Paper Award from the Control Systems Society of the IEEE in 1994 (for a paper co-authored with S. L. Chung and F. Lin) and in 2001 (for a paper co-authored with G. Barrett). Dr. Lafortune was a member of the Editorial Board of the Journal of Discrete Event Dynamic Systems: Theory and Applications in the period 1993-2000. He was Associate Editor at Large (1996-1999) and Associate Editor (1993-1996) of the IEEE Transactions on Automatic Control. His research interests are in discrete event systems, intelligent transportation systems, and communication networks. He co-authored, with C. Cassandras, the textbook *Introduction to Discrete Event Systems* (Kluwer Academic Publishers, 1999). Recent publications are available at the Web site www.eecs.umich.edu/umdes.