Chapter 48
Supervisory Control of Discrete-Event Systems

Jan Komenda and Tomáš Masopust

Abstract The aim of this essay is to provide a brief introduction to supervisory control theory of discrete-event systems.

48.1 Motivation

In our daily lives we are confronted with many different machines that can be viewed as instances of discrete-event systems. Probably the most popular examples are personal computers, laptops, tablets, mobile phones, ATM machines, beverage machines, copy machines, medical scanners, and others. Many large engineering systems, such as manufacturing or transportation systems, contain discrete-event components, for instance conveyor belts, robots, storage capacities, on-board computers in vehicles, etc. Large complex engineering systems are often built out of the smaller ones as their synchronous or asynchronous compositions. Notice that the composition of discrete-event systems results in a discrete-event system again. Discrete-event systems composed of two or more subsystems are called distributed. As the complexity of distributed systems grows, human operators are not able to design a controller by hand, and a formal approach to design a controller or supervisor is needed. The task of a supervisor is to impose a given requirement on the behavior of the system that is usually referred to as control specification. Traditionally, control specifications are formulated only informally and software engineers translate them into a control software manually. This is a time-consuming and error-prone process; moreover, the produced software needs to be verified using model-checking techniques.

Supervisory control is a formal method providing a theory to design supervisors for discrete-event systems. Using supervisory control theory, the control design

Jan Komenda · Tomáš Masopust
Institute of Mathematics, Academy of Sciences of the Czech Republic, Žižkova 22, 616 62 Brno, Czech Republic, e-mail: komenda@ipm.cz, masopust@math.cas.cz

process becomes fully automated. Typical examples of control specifications are to avoid dangerous states in the systems such as overflows or underflows of buffers in manufacturing systems, to avoid collision of vehicles in transportation systems or to control access to databases with many users, where a new user may enter only after all previous users have completed their tasks (writing to the database). Supervisory control theory is also applicable to continuous-time and hybrid systems (those composed of a discrete and continuous components) after these systems are abstracted into discrete-event systems.

The aim of this essay is to acquaint the reader with the basic notions, concepts and results of discrete-event systems and supervisory control theory. It is an introduction to the subsequent essay on coordination control, Chapter **??**. For further details, the reader is referred to [2, 9, 12]. The pioneering work on this topic are the papers [6, 7].

48.2 Concepts

This chapter differs from the other chapters of this book in the considered model. The model used in this chapter is a finite automaton, also called a finite-state machine. In supervisory control theory, automata are usually called generators. An introduction to automata theory and formal languages can be found in [4, 10]. Here we recall only the notions necessary for the presented theory.

Consider a discrete-event system. Each action of such a system is described by an event. Let Σ denote a finite nonempty set of *events*. The behavior of a system is then a finite sequence of actions, hence we can see it as a finite sequence of events. Such a finite sequence is called a *word* over the event set Σ. The set of all words over the event set Σ is denoted by Σ^*. The initial behavior of a system, where no action has yet been performed, is described by the *empty word*, denoted by ε. For instance, for an event set $\Sigma = \{e,h,l,o\}$, the word *hello* is an example of a word over Σ.

As already mentioned, the basic model of discrete-event systems used here is a generator. A *generator* is a quintuple $G = (Q,\Sigma,f,q_0,Q_m)$, where Q is a finite nonempty set of *states*, Σ is a finite set of events (*event set*), $f : Q \times \Sigma \to Q$ is a *partial transition function*, $q_0 \in Q$ is the *initial state*, and $Q_m \subseteq Q$ is a set of *marked states*. For instance, let $G = (\{1,2,3\},\{\$1,\$2,tea,cof\},f,1,\{1\})$ be a generator, modeling a simple beverage machine. A graphical representation of generators uses labeled graphs as demonstrated in Fig. 48.1. States of the generator are drawn as circles, a transition $f(q,a) = p$ is depicted as a labeled arrow from state q to state p labeled by event a, the initial state is denoted by an incoming arrow that does not come from any other state, and the marked states are drawn as double circles.

The transition function f can be extended from events to words as the function $\hat{f} : Q \times \Sigma^* \to Q$ so that $\hat{f}(q,\varepsilon) = q$ and $\hat{f}(q,aw) = \hat{f}(f(q,a),w)$, for $a \in \Sigma$ and $w \in \Sigma^*$. The behavior of a generator G is described in terms of languages. The language *generated* by G is the set $L(G) = \{s \in \Sigma^* \mid \hat{f}(q_0,s) \in Q\}$, and the language *marked*

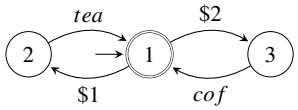

Fig. 48.1 A graphical representation of the generator G of a simple beverage machine; tea costs \$1, while we need to pay \$2 to get coffee.

by G is the set $L_m(G) = \{s \in \Sigma^* \mid \hat{f}(q_0, s) \in Q_m\}$. Obviously, $L_m(G) \subseteq L(G)$. Thus, for our example, we have $L(G) = \{\varepsilon, \$1, \$1\, tea, \$1\, tea\, \$2, \$1\, tea\, \$2\, coffe, \ldots\}$ and $L_m(G) = \{\varepsilon, \$1\, tea, \$1\, tea\, \$2\, coffe, \ldots\}$.

The set of behaviors of a generator is called a (regular) language. On the other hand, a *(regular) language* L over an event set Σ is a set $L \subseteq \Sigma^*$ such that there exists a generator G with $L_m(G) = L$. The prefix closure $\overline{L}$ of a language L over an event set Σ is the set of all prefixes of all its words, that is, $\overline{L} = \{w \in \Sigma^* \mid$ there exists $u \in \Sigma^*$ such that $wu \in L\}$; language L is prefix-closed if $L = \overline{L}$.

To control a system, we need to specify which events can be controlled, that is, disabled by a supervisor. This is done by the notion of a controlled generator. A *controlled generator* over an event set Σ is a triple (G, Σ_c, Γ), where G is a generator over $\Sigma = \Sigma_c \cup \Sigma_u$, where Σ_c is the set of *controllable events*, $\Sigma_u = \Sigma \setminus \Sigma_c$ is the set of *uncontrollable events*, and $\Gamma = \{\gamma \subseteq \Sigma \mid \Sigma_u \subseteq \gamma\}$ is the *set of control patterns*. Only controllable events can be disabled by the supervisor, while uncontrollable events cannot be prevented from happening. Typical instances of uncontrollable events are fault events, ticks of clocks, high priority events, unpreventable events due to hardware or actuation limitations, events that should not be disabled, and so on. Supervisors choose events among those from control patterns.

Example 48.1. Let $G = (\{1,2,3\}, \{a,b,c,d\}, f, 1, \{1\})$ be a generator depicted in Fig. 48.2 with the set of controllable events $\Sigma_c = \{a,c\}$. Then $\Sigma_u = \{b,d\}$ are uncontrollable events (whose transitions are depicted by dashed arrows) and $\Gamma = \{\{b,d\}, \{a,b,d\}, \{b,c,d\}, \{a,b,c,d\}\}$ is the set of control patterns. One can think of this example as a simple manufacturing system, where a single resource (e.g. a machine) is shared by two manufacturing lines: one line is represented by operations (event sequences) $(ab)^*$ and the other one by operations $(cd)^*$. All possible schedules are considered, that is, the resource can be attributed to an arbitrary sequence of both lines.

A *supervisor* for the controlled generator (G, Σ_c, Γ) is a map $S : L(G) \to \Gamma$. The meaning of a supervisor is that for any state of the system, i.e., a word w generated

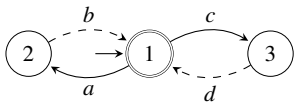

Fig. 48.2 Generator G

by the generator, $S(w)$ defines the set of events that are enabled in the system after w is generated. By definition of Γ, only controllable events can be disabled.

Applying a supervisor on a (controlled) generator results in the closed-loop system. The *closed-loop system* associated with the controlled generator (G,Σ_c,Γ) and the supervisor S is the resulting (supervised/controlled) system defined as the minimal language $L(S/G) \subseteq \Sigma^*$ such that

1. $\varepsilon \in L(S/G)$ and
2. if $s \in L(S/G)$, $a \in S(s)$, and $sa \in L(G)$, then $sa \in L(S/G)$.

The intuition is that the supervisor disables some transitions of the generator G, but it can never disable any transition under an uncontrollable event. The marked language of the closed-loop system is defined as $L_m(S/G) = L(S/G) \cap L_m(G)$, meaning that in the closed-loop system the states are marked in the same way as in G.

If the closed-loop system is nonblocking, that is, $\overline{L_m(S/G)} = L(S/G)$, then the supervisor S is called *nonblocking*.

Example 48.2. Consider the controlled generator from Example 48.1. Our goal is to define a supervisor that disables events c and a in an alternating way when the plant is back in the initial state. More precisely, c is disabled in the initial state at the beginning of the work of the system (that is, the generated word is ε), a is disabled in the state after ab is generated, c is disabled after $abcd$ is generated and so forth. We define the supervisor S as follows. For $k \geq 0$,

- $S((abcd)^k) = \{a,b,d\}$,
- $S((abcd)^k ab) = \{b,c,d\}$,
- for all other words w, $S(w) = \{a,b,c,d\}$.

The closed-loop system is then $L(S/G) = \overline{L_m(S/G)} = \overline{\{(abcd)^k \mid k \geq 0\}}$, so the supervisor is nonblocking.

The following two concepts play a central role in supervisory control [12].

Definition 48.1 (Controllable language). Let G be a generator over an event set Σ. A language $K \subseteq L(G)$ is *controllable* with respect to $L(G)$ and Σ_u if

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

The concept of controllability of a specification language in supervisory control theory differs from controllability in classical control theory of linear or nonlinear systems. It is however closely related to the concept of invariant spaces from geometrical control theory, because it requires that one cannot exit from the specification by an uncontrollable transition while staying within the plant language.

Definition 48.2 ($L_m(G)$-closed language). Let G be a generator. A nonempty language $K \subseteq L_m(G)$ is $L_m(G)$-*closed* if

$$K = \overline{K} \cap L_m(G)\,.$$

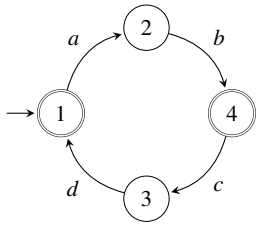

Fig. 48.3 Generator of the specification K

Example 48.3. Consider the generator G defined in Example 48.1, and define a specification language K as the language of the generator depicted in Fig. 48.3. Note that the specification restricts the behavior of the manufacturing system by imposing a particular schedule so that the resource is attributed in an alternating way to both manufacturing lines. One can verify that K is controllable with respect to $L(G)$ and Σ_u, and $L_m(G)$-closed.

48.3 Supervisory Control Problem

In this section, we formally define the supervisory control problem. Let K be a specification language, and let G be a plant (generator). The control objective of supervisory control is to find a nonblocking supervisor S (if possible) such that the closed-loop system satisfies the specification, that is,

$$L(S/G) = \overline{K} \qquad \text{and} \qquad L_m(S/G) = K\,.$$

It cannot be satisfied if $K \not\subseteq L_m(G)$, therefore we can assume that $K \subseteq L_m(G)$.

48.4 Supervisory Control Theory

The supervisory control problem is to find conditions that are equivalent to the existence of a supervisor that achieves a specification. Two conditions defined above, *controllability* and $L_m(G)$*-closedness*, are necessary and sufficient for the existence of a nonblocking supervisor that achieves the specification, see [2, 12] for the proofs of the following theorems.

Theorem 48.1. *Consider the problem specified above. There exists a nonblocking supervisor S solving the problem if and only if the specification language K is both controllable with respect to $L(G)$ and Σ_u, and $L_m(G)$-closed.*

Example 48.4. Consider the plant and the specification of Example 48.3. By Theorem 48.1, there exists a supervisor S solving the supervisory control problem. This

supervisor is described in Example 48.2. Note that the supervisor can be represented as an automaton. Moreover, if the specification K is controllable with respect to the plant language $L(G)$ and Σ_u, the generator for the specification is precisely the automaton representation of the supervisor. Thus, the automaton representation of the supervisor S is depicted in Fig. 48.3.

It remains to explain what to do if the specification language is not controllable (in some sense, the $L_m(G)$-closedness is not an issue, because if K is not $L_m(G)$-closed, then $\overline{K} \cap L_m(G)$ is considered as a new specification, cf. [2]). For uncontrollable specification languages, controllable sublanguages of the specification are considered instead. Note that control specifications are most often safety specifications expressed by a language inclusion and, therefore, it is reasonable to compute a controllable sublanguage of a specification, if the specification fails to be controllable. The notation $C(K, L(G), \Sigma_u)$ stands for the set of controllable sublanguages of the specification K with respect to $L(G)$ and Σ_u. It is not hard to check that controllability is preserved by language unions. Consequently, there always exists the supremal controllable sublanguage of the specification language among the controllable sublanguages, denoted by $\sup C(K, L(G), \Sigma_u)$, see [2, 12].

Theorem 48.2. *The supremal controllable sublanguage of a specification language always exists and is equal to the union of all controllable sublanguages of the specification.*

The supremal controllable sublanguage is an important concept, because it represents the maximally permissive (or, equivalently, minimally restrictive) solution to the supervisory control problem.

48.5 Nonblockingness in Distributed Systems

In this section we study the blocking issue that can appear in distributed discrete-event systems. A *distributed discrete-event system* with synchronous communication is a concurrent system formed by the synchronous product of several local subsystems. The engineering relevance of distributed systems modeled by discrete-event systems can be justified by showing that supervisory control for the following systems has been investigated in the literature. Control of a rapid thermal multiprocessor [1], databases [2], chemical plants [8], feature interaction in telephone networks [11], theme park vehicles [3], a controller for traffic lights [5].

Synchronous product is a standard way of constructing large-scale systems as a composition of potentially a large number of smaller systems. Formally, a synchronous product of languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ is the language

$$L_1 \| L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2) \subseteq \Sigma^*,$$

where $P_i : \Sigma^* \to \Sigma_i^*$ are natural projections, for $i = 1, 2$. A *(natural) projection* $P : \Sigma^* \to \Sigma_0^*$, where $\Sigma_0 \subseteq \Sigma$, is a homomorphism defined so that $P(a) = \varepsilon$ for $a \in \Sigma \setminus \Sigma_0$,

Fig. 48.4 Generators G_1 and G_2

and $P(a) = a$ for $a \in \Sigma_0$. The projection of a word is thus uniquely determined by projections of its letters. The *inverse image* of P is denoted by $P^{-1} : \Sigma_0^* \to 2^{\Sigma^*}$.

Example 48.5. Let $P : \{a,b,c\}^* \to \{a,b\}^*$ be a projection. Then the projection of a word *abcba* is $P(abcba) = abba$. On the other hand, the inverse image of *abba* is $P^{-1}(abba) = \{c^i ac^j bc^k bc^\ell ac^m \mid i,j,k,\ell,m \geq 0\}$.

For two generators $G_1 = (Q_1, \Sigma_1, f_1, q_{01}, Q_{m1})$ and $G_2 = (Q_2, \Sigma_2, f_2, q_{02}, Q_{m2})$, the generator $G_1 \| G_2$ is defined as the accessible part (i.e., the part of the state set which can be reached from the initial state) of the generator $(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f, (q_{01}, q_{02}), Q_{m1} \times Q_{m2})$, where

$$f((x,y),e) = \begin{cases} (f_1(x,e), f_2(y,e)), & \text{if } f_1(x,e) \in Q_1 \text{ and } f_2(y,e) \in Q_2, \\ (f_1(x,e), y), & \text{if } f_1(x,e) \in Q_1 \text{ and } e \notin \Sigma_2, \\ (x, f_2(y,e)), & \text{if } e \notin \Sigma_1 \text{ and } f_2(y,e) \in Q_2, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

It is known that the relation to the synchronous product of languages is as follows: $L(G_1 \| G_2) = L(G_1) \| L(G_2)$ and $L_m(G_1 \| G_2) = L_m(G_1) \| L_m(G_2)$.

Example 48.6. Consider two generators G_1 and G_2 depicted in Fig. 48.4. Note that the only event shared by the generators is the event a, and that events b and d are private in the respective generators. Then the synchronous product (also called parallel composition) $G_1 \| G_2$ is depicted in Fig. 48.5.

Recall that a generator G is nonblocking if $\overline{L_m(G)} = L(G)$, that is, if every generated word from $L(G)$ can be prolongated to a marked word from $L_m(G)$. Otherwise, we say that the system is blocking, which typically arises in discrete-event systems formed by the synchronous product. It is well known that the synchronous product of two nonblocking generators G_1 and G_2 can be blocking.

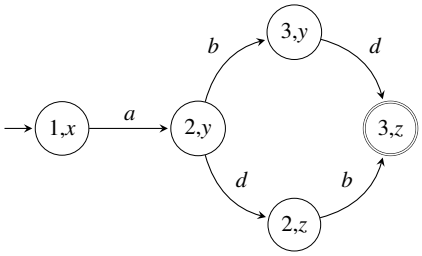

Fig. 48.5 Synchronous product $G_1 \| G_2$

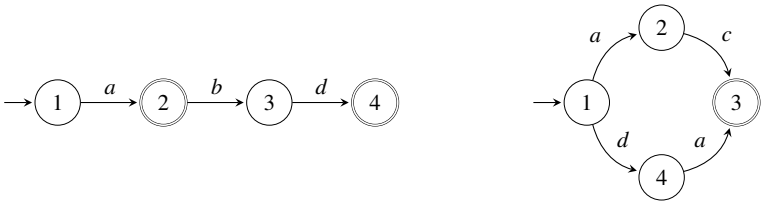

Fig. 48.6 Generators G_1 and G_2

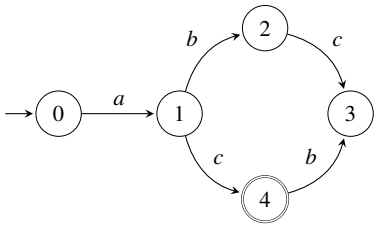

Fig. 48.7 Synchronous product $G_1 \| G_2$

Example 48.7. Consider nonblocking generators G_1 and G_2 depicted in Fig. 48.6. Their synchronous product, depicted in Fig. 48.7, is blocking because no marked state is accessible from state 3.

References

1. S. Balemi, G. J. Hoffmann, P. Gyugi, H. Wong-Toi, and G.F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Trans. Automat. Control*, 38(7):1040–1059, 1993.
2. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, 2008.
3. S. T. J. Forschelen, J. M. Mortel-Fronczak, R. Su, and J. E. Rooda. Application of supervisory control theory to theme park vehicles. *Discrete Event Dyn. Syst.*, 22(4):511–540, 2012.
4. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2006.
5. R. P. Kurshan. *Computer-aided verification of coordinating processes: the automata-theoretic approach*. Princeton University Press, 1994.
6. P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987.
7. P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proc. of IEEE*, 77(1):81–98, 1989.
8. L. Sang-Heon. *Structural decentralised control of concurrent discrete-event systems*. PhD thesis, Australian National University, Canberra, 1998.
9. C. Seatzu, M. Silva, and J. H. van Schuppen, editors. *Control of Discrete-Event Systems*, volume 433 of *Lecture Notes in Control and Information Sciences*. Springer London, 2013.
10. M. Sipser. *Introduction to the Theory of Computation*. Course Technology, 2005.
11. J. G. Thistle, R. P. Malhamé, H. H. Hoang, and S. Lafortune. Feature interaction modelling, detection and resolution: A supervisory control approach. In *FIW*, pages 93–107, 1997.
12. W. M. Wonham. Supervisory control of discrete-event systems. Lecture notes, Department of electrical and computer engineering, University of Toronto, 2009.