Supervisory control of discrete-event systems in an asynchronous setting

2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)

Supervisory Control of Discrete-Event Systems in an Asynchronous Setting

Aida Rashidinejad[1] Michel Reniers[1] Martin Fabian[2]

***Abstract*— In conventional supervisory control theory, a plant and supervisor are supposed to work synchronously such that enabling an event by the supervisor, execution of it in the plant, and observation of the executed event by the supervisor all occur at once. Therefore, these occurrences are all captured by means of a single event. However, when a supervisor synthesized from conventional supervisory control theory is implemented in real life, it will face problems since exact synchronization can hardly happen in practice due to delayed communications. In this paper, we propose a synthesis technique to achieve a supervisor that does not face the problems caused by inexact synchronization. For this purpose, we first introduce an asynchronous setting in which enablement, execution, and observation of an event do not occur simultaneously but with some delay. We present a model representing the behavior of the plant in the asynchronous setting which we call the asynchronous plant. For the asynchronous plant, we present an algorithm synthesizing an asynchronous supervisor which satisfies (asynchronous) controllability and nonblockingness.**

I. Introduction

Discrete-event systems (DES) are systems with a discrete set of states in which transitions take place in association with instantaneous events. Many types of physical systems can be modeled as DES including manufacturing systems, traffic systems, and communication networks [1], [2]. Such systems are typically supervised in order to satisfy some control requirements. Supervisory Control Theory (SCT) has been developed to automatically synthesize such a supervisor from models of the uncontrolled system and the control requirements [3]. In a supervisory control setting, control commands are sent from the supervisor to the actuators in the plant through the control channel, and events occurring in the plant are observed by the sensors and sent to the supervisor through the observation channel. Several problems that arise in implementation of supervisors have been reported in the literature as briefly discussed in the following [4], [5], [6], [7], [8]:

- avalanche effect: this is the problem of multiple state transitions occurring on the same event, leading to instantaneously passing through intermediate states so that actions from those states are not executed.
- interleave sensitivity: this problem occurs when different interleavings of events can be executed at a state, and the supervisor needs to make decisions based on the order of events that it observes. The problem arises when events may not necessarily be observed in the same order as they have been executed in the plant.
- causality: SCT assumes all events can occur spontaneously in the plant, and the supervisor may only disable a subset of them. However, (in practice) events which are controllable need to be commanded by the supervisor, otherwise they cannot occur.
- choice: SCT achieves a supervisor which is maximally permissive meaning that it gives the plant the greatest amount of freedom within the control requirements. Therefore, in many cases, the supervisor should make a choice between alternative transitions that are possible at a state, and this will be highly dependent on the implementation.
- inexact synchronization: in SCT, a supervisor is synthesized based on the assumption of synchronous interactions between the plant and supervisor. In other words, a supervisor is assumed to immediately observe an event as it is executed by the plant, and the plant receives a control command immediately after it has been sent from a supervisor. However, such a supervisor will be implemented in an asynchronous setting where sending and receiving data are subject to delays. Hence, the synchronous assumption does not typically hold.

Here, we focus on the problem caused by inexact synchronization. Additionally, we take into account the problems of causality and interleave sensitivity by allowing the plant to execute a controllable event only if it is enabled (commanded) by a supervisor, and also allowing consecutive events to be observed in any possible order.

The problem caused by inexact synchronization was first addressed by Balemi [9]. It arises in the situation where the occurrence of an uncontrollable event invalidates a control command (or the selection of a controllable event). This problem is caused by a communication delay where the occurrence of an uncontrollable event is interpreted as a communication delay unit. Consider a state of the plant where both a controllable event and an uncontrollable event are enabled. If there is a communication delay, the supervisor might send the controllable event while the plant transits to another state on the uncontrollable event. Then, the control command may arrive when the plant is in a state that invalidates that command.

To solve the problem caused by communication delays, Balemi introduced the notion of “delay insensitive language”. A language is delay insensitive if a control command is not

* This research has received funding from the European Unions Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 674875.

[1]Aida Rashidinejad and Michel Reniers are with Department of Mechanical Engineering, Eindhoven University of Technology, The Netherlands {a.rashidinejad, m.a.reniers}@tue.nl

[2]Martin Fabian is with Department of Electrical Engineering, Chalmers University of Technology, Sweden fabian@chalmers.se

invalidated by an uncontrollable event. In other words, any control command sent by the supervisor should be accepted by the plant. If a supervised plant is delay insensitive, then the achieved supervisor does not face any problem caused by inexact synchronization. However, this condition is not met by most applications.

Besides the inexact synchronization, the causality problem was also considered by Balemi in [9] where he uses an "input/output semantics" for the plant. In this input/output perspective, controllable events are considered as inputs to the plant and uncontrollable events are the outputs or responses generated by the plant. The condition of delay insensitivity was also used in [10] called "$\Sigma_u - \Sigma_c$-commuting" condition. This condition has been further generalized in [5], [11] for a sequence of uncontrollable events (representing a bounded observation delay).

In [4], a condition for "interleave insensitivity" was introduced. In this case, having an implementable supervisor is limited to applications which require the same control command after any interleaving of a sequence of uncontrollable events. In [5], delay insensitivity and interleave insensitivity are captured in a single definition as "delay interleave insensitivity".

If we assume that a control command is enabled until a disablement command is received from a supervisor, then delays in control and observation channels may have different effects and need to be investigated separately [12], [13]. To consider the effect of delays, in [12], a condition called "bounded-delay implementability" has been introduced. This condition takes into account the effects of observation and control delays on a requirement behavior by achieving a delayed version of the requirement. In case the delayed version of the requirement stays within the requirement (without delays) it can be satisfied in an asynchronous setting as well. Furthermore, in [13] new observability and controllability conditions under delays are introduced as "delay observability" and "delay controllability", respectively. A supervisor can be synthesized for a requirement which satisfies these conditions.

The drawback of the mentioned conditions is that they disqualify many relevant cases that do not satisfy the condition. More recently, a networked supervisory control framework has been introduced in [14] where a networked supervisor is synthesized to deal with communication delays. To indicate that enabling, execution, and observation of events do not occur at the same time, different notions for enabling and observed events are introduced.

In this paper, we introduce an asynchronous supervisory control setting describing the situation where a control command enabled by the supervisor stays in the control channel until being executed in the plant. Also, the observation of an event may occur immediately after execution in the plant or at some point in the future. Our asynchronous supervisory control setting is close to the networked supervisory control framework introduced in [14]. We use the same notations for enabling, execution, and observation of events as discussed in [14]. However, the asynchronous setting presented here is different from the networked supervisory control technique presented in [14] mainly because we do not quantify the amount of delay. The objective of this paper is to present a method to synthesize a supervisor taking into account the following conditions that may exist in practice:

1) A controllable event can be executed in the plant only if it is commanded (enabled) by the supervisor. Also, any event executed in the plant is observable to the supervisor.
2) An uncontrollable event is not commanded (enabled) by a supervisor, and it only occurs spontaneously in the plant.
3) A control command sent by the supervisor may not necessarily be accepted by the plant, and in this case it will stay in the control channel since in the asynchronous setting there is no deadline for an enabling event to reach the plant.
4) The observation of an event, controllable as well as uncontrollable, may occur immediately after being executed in the plant or at some point in the future.
5) Consecutive events that occur in the plant may be observed by the supervisor in any possible order.

The rest of the paper is organized as follows. The conventional supervisory control approach is summarized in Section II. In Section III, we present the asynchronous supervisory control setting, and we introduce an asynchronous composition operator to achieve an asynchronously supervised plant in this setting. Afterwards, in Section IV, we present a method for transforming a plant into an asynchronous plant modeling the behavior of the plant as asynchronously observed and controlled by the supervisor. Based on the asynchronous plant, an algorithm is proposed to synthesize an asynchronous supervisor which guarantees (asynchronous) controllability and nonblockingness. Finally, Section V concludes the paper and discusses future work.

II. Background

A DES G is formally represented as a quintuple

$$G = (A, \Sigma, \delta, a_0, A_m), \tag{1}$$

where A, Σ, $\delta : A \times \Sigma \to A$, $a_0 \in A$, and $A_m \subseteq A$ stand for the set of states, the set of events, the (partial) transition function, the initial state, and the set of marked states, respectively. An automaton with a finite set of states and a finite set of events is called a finite automaton [15]. The notation $\delta(a,\sigma)!$ denotes that δ is defined for state a and event σ, i.e., there is a transition from state a with label σ to some state. The transition function is generalized to words in the usual way: $\delta(a,w) = a'$ means that there is a sequence of subsequent transitions from state a to the state a' that together make up the word $w \in \Sigma^*$. Starting from the initial state, the set of all possible words that may occur in G is called the language of G and is indicated by $L(G) := \{w \in \Sigma^* \mid \delta(a_0,w)!\}$. Furthermore, for a state $a \in A$, the function $Reach(a)$ gives the set of states reachable from state a; $Reach(a) := \{a' \mid \exists w \in \Sigma^*, \delta(a,w) = a'\}$. States from which it is possible to reach a marked state are said to be nonblocking. An automaton is nonblocking when each state reachable from the initial state is nonblocking; for each $a \in Reach(a_0)$, $Reach(a) \cap A_m \neq \varnothing$.

Moreover, in this paper, we frequently use the natural projection operator [1].

Definition 1 (Natural projection): For sets of events Σ and $\Sigma' \subseteq \Sigma$, $P_{\Sigma'} : \Sigma^* \to \Sigma'^*$ is defined as follows: for $e \in \Sigma$ and $w \in \Sigma^*$

$$P_{\Sigma'}(\varepsilon) := \varepsilon,$$

$$P_{\Sigma'}(we) := \begin{cases} P_{\Sigma'}(w)e & \text{if } e \in \Sigma', \\ P_{\Sigma'}(w) & \text{if } e \in \Sigma \setminus \Sigma'. \end{cases}$$

The definition of a natural projection can be extended to a language; given a language $L \subseteq \Sigma^*$, $P_{\Sigma'} : \Sigma^* \to \Sigma'^*$ maps it to a set of words from Σ'^* where $\Sigma' \subseteq \Sigma$ such that $P_{\Sigma'}(L) := \{w' \in \Sigma'^* \mid \exists w \in L, P_{\Sigma'}(w) = w'\}$ [1]. ■

Note that, although natural projection is an operation which is generally defined for languages, it is also possible to apply it on automata [16]. For an automaton with event set Σ, $P_{\Sigma'}$ first replaces all events not from Σ' by ε. Then, using the determinisation algorithm introduced in [15], the achieved automaton becomes deterministic again. For the projection operator, the following lemma holds.

Lemma 1 (Nonblockingness over Projection): If an automaton G with event set Σ and set of states A is nonblocking, then $P_{\Sigma'}(G)$ with event set $\Sigma' \subseteq \Sigma$ is also nonblocking. ■

Proof: Consider a set of reachable states A_r in $P_{\Sigma'}(G)$. Due to the definition of projection on automata, each state of $P_{\Sigma'}(G)$ is a subset of A, and so one can say $A_r \subseteq A$. By construction A_r is not empty and thus contains at least one element, say $a_r \in A$. By definition a_r is reachable in G and since G is nonblocking there is a $w \in \Sigma^*$ such that $\delta(a_r, w) \in A_m$. Then, again by construction and by the properties of the involved determinisation procedure (determinisation makes a state of $P_{\Sigma'}(G)$ marked if it includes at least one of the marked states of G [15]) we have that $P_{\Sigma'}(w)$ allows to reach a marked state in $P_{\Sigma'}(G)$. ■

Lemma 1 will be used in further proofs. From now on, we assume that the plant is given as a finite automaton G in which all events are observable. However, a subset of events may be uncontrollable indicated by $\Sigma_{uc} \subseteq \Sigma$. The set of controllable events is then given by $\Sigma_c = \Sigma \setminus \Sigma_{uc}$. To satisfy nonblockingness, a supervisor is required to be synthesized for the plant G. A conventional supervisor S is also a DES with the same event set as G. The conventional supervisory control setting is depicted in Figure 1.

Here we focus on solving the basic synthesis problem to achieve a supervisor satisfying controllability and nonblockingness. As described in [17], control requirements can be translated to the plant. By applying the basic synthesis, a supervisor is achieved which fulfills the requirements, and it satisfies controllability and nonblockingness.

In conventional SCT, the plant and supervisor are supposed to work synchronously, and the automaton modeling the supervised plant is obtained by applying the synchronous composition operator indicated by $S||G$. Generally, in the synchronous composition of two automata, a shared event can be executed only when it is enabled by both automata, and a non-shared event can be executed if it is enabled by one of the automata. Since S is assumed to have the same event set as G, each event will be executed in $S||G$ only if the supervisor allows it. Since uncontrollable events enabled in G can never be disabled by S, they should always be allowed in $S||G$, and this can be checked by the controllability condition given in Definition 2.

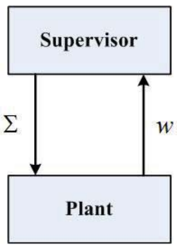
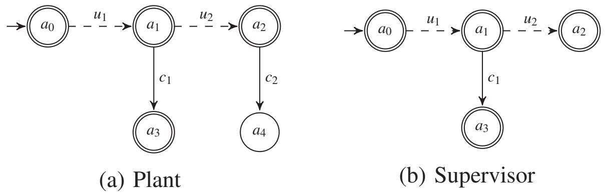


Fig. 1: Conventional supervisory control (synchronous setting).

Definition 2 (Conventional Controllability [2]): For any DES G controlled by a conventional supervisor S, $S||G$ is controllable if for any $w \in L(S||G)$ and $u \in \Sigma_{uc}$, whenever $wu \in L(G)$ then $wu \in L(S||G)$. ■

In the following example, we explain the problem caused by inexact synchronization.

Example 1 (Motivating example from [4]): Consider the plant G, given in Figure 2a, for which u_1 and u_2 are uncontrollable events (indicated by dashed lines), and c_1 and c_2 are controllable events (indicated by solid lines). The state a_4 is blocking (marked states are indicated by double circles) and needs to be avoided by a supervisor. Using conventional supervisory control synthesis, the supervisor depicted in Figure 2b is obtained. As described in [4], the problem appears when after the execution of u_1, S enables c_1; however, u_2 occurs in G which invalidates the control command sent by S (because G cannot accept c_1 after executing u_2). Clearly, this problem does not occur if c_1 could occur after u_2 as well. This is actually the "delay insensitive language" condition introduced in [9]. ■



(a) Plant (b) Supervisor

Fig. 2: The plant and the conventional supervisor from [4].

In order to synthesize a supervisor dealing with the problems that may arise in practice, we need to take into account the five conditions mentioned in Section I while synthesizing a supervisor.

III. Asynchronous Supervisory Control Setting

In the synchronous setting, enabling, execution, and observation of events are assumed to occur simultaneously, and so all of these are indicated by one and the same event. As depicted in Figure 3, to indicate that the enabling, execution, and observation of events do not occur simultaneously in an asynchronous setting, new notions are introduced in Definition 3. Motivated from [14], for each event σ from the plant we introduce unique events σ_e and σ_o representing the related enabling and observed events, respectively.

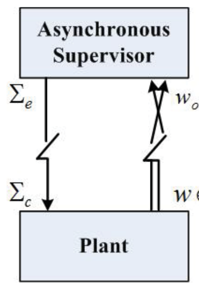


Fig. 3: Asynchronous supervisory control setting.

Definition 3 (Enabling and Observed Events [14]): For a plant $G = (A, \Sigma, \delta, a_0, A_m)$, to each controllable event $\sigma \in \Sigma_c$ an enabling event $\sigma_e \in \Sigma_e$ and to any event $\sigma \in \Sigma$ an observed event $\sigma_o \in \Sigma_o$ are associated. ■

As it is clear from Figure 3, the plant and supervisor do not have the same event set in the proposed asynchronous setting. The event set of an asynchronous supervisor includes only the enabling and observed events. To achieve the supervised plant in the asynchronous setting, we need to define an asynchronous composition operator. To indicate how events may be observed in a supervisory control system, we assume that an event $\sigma \in \Sigma$ executed in the plant will be stored in the observation channel until being observed as $\sigma_o \in \Sigma_o$. Since it is assumed that events may not necessarily be observed in the same order as they have been executed in the plant, the observation channel is represented as a multiset. The multiset given in Definition 4 is a representation of the contents of the observation channel which will be used in determining the occurrences of the observed events.

Definition 4 (Observation Channel Representation): The set M is defined as $M = \Sigma \times \mathbb{N}$. Moreover, we define the following operations for all $m \in M$, and $\sigma, \sigma' \in \Sigma$:

- $m(\sigma)$ denotes the number of events σ in m.
- $[]$ denotes the empty multiset, i.e, the function m with $m(\sigma) = 0$ for all $\sigma \in \Sigma$.
- $|m| = \sum_{\sigma \in \Sigma} m(\sigma)$ denotes the number of elements in m
- $m \uplus [\sigma]$ inserts σ to m. Formally, it denotes the function m' for which $m'(\sigma) = m(\sigma) + 1$ and $m'(\sigma') = m(\sigma')$ for $\sigma' \neq \sigma$.
- $m \setminus [\sigma]$ removes σ from m once. Formally, it denotes the function m' for which $m'(\sigma) = \max(m(\sigma) - 1, 0)$ and $m'(\sigma') = m(\sigma')$ for $\sigma' \neq \sigma$.
- $\sigma \in m$ denotes that σ is present in m, it holds if $m(\sigma) > 0$. ■

Similarly, control commands sent by a supervisor are assumed to be stored in a control channel until being executed in the plant. However, since events will be executed based on the order that they have been commanded, the control channel is represented by a list as given in Definition 5.

Definition 5 (Control Channel Representation): The set L is defined as $L = \Sigma^*$. Moreover, we define the following operations for all $\sigma \in \Sigma$, and $l \in L$:

- *app*(l, σ) adds the element σ to the end of l.
- *head*(l) gives the first element of l (for nonempty lists).
- *tail*(l) denotes the list after removal of its head. ■

Based on the representations of observation and control channels, an operator is presented in Definition 6 to obtain the asynchronously supervised plant. This operator is derived from the "networked supervised plant operator" introduced in [14] where similar representations are proposed for observation and control channels. Note that this asynchronous composition operator is only meaningful if G and AS are in accordance with the presented asynchronous supervisory control setting.

Definition 6 (Asynchronous Composition Operator): Consider the plant $G = (A, \Sigma, \delta, a_0, A_m)$ controlled by the supervisor $AS = (Y, \Sigma_{AS}, \delta_{AS}, y_0, Y_m)$ in an asynchronous setting. Then, the asynchronous composition of G and AS, denoted by $AS|/|G$ gives the asynchronously supervised plant as the following automaton

$$AS|/|G = (Z, \Sigma_{ASP}, \delta_{ASP}, z_0, Z_m),$$

where

$$\begin{aligned} Z &= A \times Y \times M \times L, \\ \Sigma_{ASP} &= \Sigma_{AS} \cup \Sigma, \\ z_0 &= (a_0, y_0, [], \varepsilon), \\ Z_m &= A_m \times Y_m \times M \times L. \end{aligned}$$

Moreover, for $a \in A$, $y \in Y$, $m \in M$, and $l \in L$, $\delta_{ASP} : Z \times \Sigma_{ASP} \to Z$ is defined as follows:

1) When an enabling event $\sigma_e \in \Sigma_e$ is executed in AS, it will be stored in l until being received by G. If $\delta_{AS}(y, \sigma_e)!$:

$$\delta_{ASP}((a, y, m, l), \sigma_e) = (a, \delta_{AS}(y, \sigma_e), m, app(l, \sigma)).$$

2) An uncontrollable event $\sigma \in \Sigma_{uc}$ can be executed in $AS|/|G$ only if it is executed in G. In addition, the event will be put in the observation channel to be received by AS later. If $\delta(a, \sigma)!$:

$$\delta_{ASP}((a, y, m, l), \sigma) = (\delta(a, \sigma), y, m \uplus [\sigma], l).$$

3) A controllable event $\sigma \in \Sigma_c$ can be executed in $AS|/|G$ if the enabling event σ_e was sent by the supervisor, and it is the first enabling event in the control channel to be executed. In addition, the event will be put in the observation channel. If $\delta(a, \sigma)!$ and *head*$(l) = \sigma$:

$$\delta_{ASP}((a, y, m, l), \sigma) = (\delta(a, \sigma), y, m \uplus [\sigma], tail(l)).$$

4) An observed event $\sigma_o \in \Sigma_o$ can be executed in $AS|/|G$ if $\sigma \in m$, subsequently σ is removed from m. If $\delta_{AS}(y,\sigma_o)!$ and $\sigma \in m$:

$$\delta_{ASP}((a,y,m,l),\sigma_o) = (a,\delta_{AS}(y,\sigma_o),m \setminus [\sigma],l). \quad \blacksquare$$

For the asynchronous composition operator presented in Definition 6, the following lemma holds.

Lemma 2 (Asynchronously Supervised Plant Transitions): Considering the asynchronously supervised plant obtained from Definition 6, for any $w \in \Sigma_{ASP}^*$ with $\delta_{ASP}(z_0,w)!$, we have $\delta_{ASP}(z_0,w) = (\delta(a_0,P_\Sigma(w)),\delta_{AS}(y_0,P_{\Sigma_{AS}}(w)),m,l)$ for some $m \in M, l \in L$. $\blacksquare$

Proof: Let $w \in \Sigma_{ASP}^*$ with $\delta_{ASP}(z_0,w)!$. We prove that $\delta_{ASP}(z_0,w) = (\delta(a_0,P_\Sigma(w)),\delta_{AS}(y_0,P_{\Sigma_{AS}}(w)),m,l)$ for some $m \in M, l \in L$ by induction on the structure of w. First, assume that $w = \varepsilon$ then we have $\delta_{ASP}(z_0,w) = (a_0,y_0,[],\varepsilon) = (\delta(a_0,\varepsilon),\delta_{AS}(y_0,\varepsilon),[],\varepsilon)$. Now, assume that $w = v\sigma$ for some $v \in \Sigma_{ASP}^*$ with $\delta_{ASP}(z_0,v)!$. By induction we have $\delta_{ASP}(z_0,v) = (\delta(a_0,P_\Sigma(v)),\delta_{AS}(y_0,P_{\Sigma_{AS}}(v)),m',l')$ for some $m' \in M$ and $l' \in L$. It is sufficient to prove that $\delta_{ASP}(z_0,v\sigma) = (\delta(a_0,P_\Sigma(v\sigma)),\delta_{AS}(y_0,P_{\Sigma_{AS}}(v\sigma)),m,l)$ for some $m \in M$ and $l \in L$. For $\sigma \in \Sigma_{ASP}$ one of the following cases could occur; if $\sigma \in \Sigma$, then $\delta_{ASP}(z_0,v\sigma) = (\delta(\delta(a_0,P_\Sigma(v)),\sigma),\delta_{AS}(y_0,P_{\Sigma_{AS}}(v)),m,l)$, and if $\sigma \in \Sigma_{AS}$, then $\delta_{ASP}(z_0,v\sigma) = (\delta(a_0,P_\Sigma(v)),\delta_{AS}(\delta_{AS}(y_0,P_{\Sigma_{AS}}(v)),\sigma),m,l)$ where in each case due to Definition 1 we get $\delta_{ASP}(z_0,v\sigma) = (\delta(a_0,P_\Sigma(v\sigma)),\delta_{AS}(y_0,P_{\Sigma_{AS}}(v\sigma)),m,l)$. $\blacksquare$

Lemma 2 will help us with the proofs to come. Note that the definition of nonblockingness and controllability stays the same as given in Section II. However, because of introducing new sets of enabling and observed events, the formal definition of conventional controllability needs to be adapted for the asynchronous setting given as asynchronous controllability in Definition 7. A supervisor is (asynchronously) controllable for the plant if any uncontrollable event that could occur in the plant can be executed in the asynchronously supervised plant.

Definition 7 (Asynchronous Controllability): Consider the plant G with event set Σ. Then, supervisor AS is asynchronously controllable for G if for all $w \in L(AS|/|G)$ and $u \in \Sigma_{uc}$, if $P_\Sigma(w)u \in L(G)$ then $wu \in L(AS|/|G)$. $\blacksquare$

In the proposed asynchronous setting, by definition, an uncontrollable plant event can occur whenever it is enabled in the plant, even though the plant is controlled by a supervisor. Therefore, as we prove in Property 1, the asynchronous controllability condition is always guaranteed by the definition of the asynchronous composition operator. Note that, although the Σ_o events are uncontrollable, they are not enabled in the plant and therefor not considered as such in asynchronous controllability.

Property 1 (Controllable Asynchronous Supervisor): For any AS and G, AS is asynchronously controllable for G. $\blacksquare$

Proof: Take $w \in L(AS|/|G)$ and $u \in \Sigma_{uc}$ such that $P_\Sigma(w)u \in L(G)$. Then, we need to prove that $wu \in L(AS|/|G)$. For $w \in L(AS|/|G)$, from Lemma 2, we have $\delta_{ASP}(z_0,w) = (\delta(a_0,P_\Sigma(w)),\delta_{AS}(y_0,P_{\Sigma_{AS}}(w)),m,l)$ for some $m \in M, l \in L$. Then, from item 2 of Definition 6, we know that u occurs in $AS|/|G$ only if it is enabled by the plant where due to the assumption we have $\delta(a_0,P_\Sigma(w)u)!$. So from $\delta_{ASP}(z_0,w)$, the event u can occur which results in $\delta_{ASP}(z_0,wu) = (\delta(a_0,P_\Sigma(w)u),\delta_{AS}(y_0,P_{\Sigma_{AS}}(w)),m \uplus [u],l)$. $\blacksquare$

Problem Statement In the following, for a given plant G, we aim to find an asynchronous supervisor AS such that $AS|/|G$ is nonblocking.

IV. Synthesis

In a real implementation, a supervisor determines the enabling commands based on the observations that it receives. Moreover, it could be possible that although the supervisor has sent a control command, the plant executes an uncontrollable event (or simply ignores the command because it cannot execute it). The control command will then stay in the control channel and it may, in some cases, block other controllable events waiting in the control channel to be executed in the plant. To achieve an asynchronous supervisor providing controllability and nonblockingness in the asynchronous setting, we do the synthesis on the "asynchronous plant" which is indicated in Figure 4.

The asynchronous plant is a model for how events are executed in the plant based on enabling events, and also how observations of the executed events may occur in the asynchronous setting. To achieve the asynchronous plant automaton, we need to determine all possible cases that the control commands (enabling events) could have been sent in the asynchronous setting. Since enabling events are based on observations, we first start by presenting a model for how the plant is actually observed in an asynchronous setting which we call the "observed plant", and it is shown in Figure 4. The formal definition of the observed plant is presented in Definition 8. To achieve a finite representation, the size of the observation channel is considered to be limited to N_o.

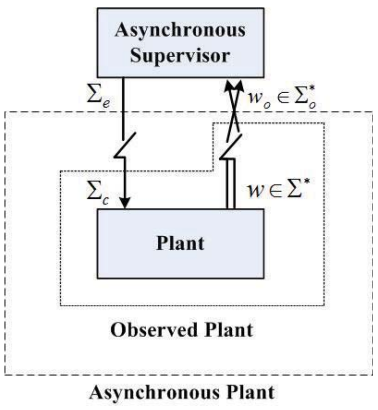


Fig. 4: Asynchronous plant and observed plant.

Definition 8 (Observed Plant): For a plant $G = (A, \Sigma, \delta, a_0, A_m)$, we define

$$\Upsilon(G,N_o) := (Q,\Sigma_{OP},\delta_{OP},q_0,Q_m),$$

where
$$Q = A \times M, \quad \Sigma_{OP} = \Sigma \cup \Sigma_o,$$
$$q_0 = (a_o, []), \quad Q_m = A_m \times M.$$

The states of $\Upsilon(G, N_o)$ depend on the current states of the plant and of the medium. Initially, no event has occurred yet, and thus the medium is empty. Whenever an event occurs in the plant, it will be stored in the medium until it is observed.

For $a \in A$, $m \in M$ and $\sigma \in \Sigma$, the transition function $\delta_{OP} : Q \times \Sigma_{OP} \to Q$ is defined as follows:

1) If $\delta(a, \sigma)!$ and $|m| < N_o$
$$\delta_{OP}((a, m), \sigma) = (\delta(a, \sigma), m \uplus [\sigma]).$$
2) If $\sigma \in m$
$$\delta_{OP}((a, m), \sigma_o) = (a, m \setminus [\sigma]).$$

Note that when there are multiple events in the medium, these can be observed in all possible orders. ■

Example 2: Let us again consider the plant from Example 1. The observed plant obtained from Definition 8 is given in Figure 5. ■

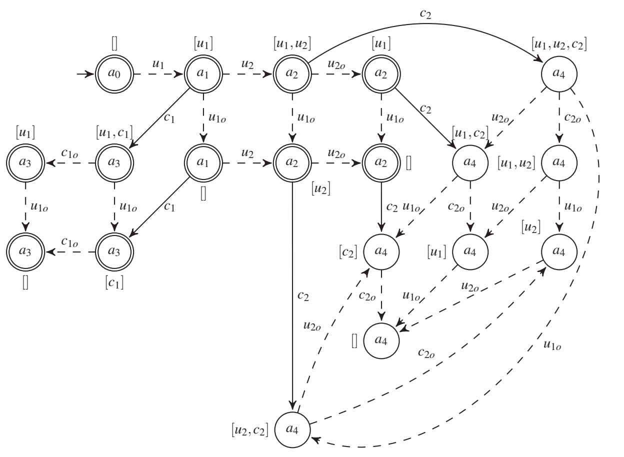


Fig. 5: Observed plant for G from Example 1.

The next step is to determine all feasible enabling events that could have been sent in the asynchronous setting based on the observed plant. However, since enabling events are only related to controllable events, we leave out the uncontrollable events of the observed plant. We also use the plant model to determine how events will be executed in the plant, and how the observations can occur in the asynchronous plant. The asynchronous plant automaton is achieved using the operator given in Definition 9. To obtain a finite automaton, the size of observation and control channels are limited to N_o and N_c, respectively. These limitations are required to guarantee the finiteness of the set of states in the presence of an event-loop.

Definition 9 (Asynchronous Plant Operator): For a given plant, $G = (A, \Sigma, \delta, a_0, A_m)$ and constants N_c and N_o, Π gives the asynchronous plant as the following automaton:
$$\Pi(G, N_c, N_o) = (X, \Sigma_{ASP}, \delta_{AP}, x_0, X_m), \qquad (2)$$

Let $OP' = P_{\Sigma_{OP} \setminus \Sigma_{uc}}(\Upsilon(G, N_o)) = (Q', \Sigma_{OP}, \delta_{OP'}, q'_0, Q'_m)$, and
$$X = A \times Q' \times M \times L,$$
$$x_0 = (a_0, q'_0, [], \varepsilon),$$
$$X_m = A_m \times Q' \times M \times L.$$

For $a \in A$, $q' \in Q'$, $m \in M$ and $l \in L$, the transition function $\delta_{AP} : X \times \Sigma_{ASP} \to X$ is defined as follows:

1) If $\delta_{OP'}(q', \sigma)!$, $\sigma \in \Sigma_c$ and $|l| < N_c$
$$\delta_{AP}((a, q', m, l), \sigma_e) = (a, \delta_{OP'}(q', \sigma), m, app(l, \sigma)).$$
2) If $\delta(a, \sigma)!$, $head(l) = \sigma, \sigma \in \Sigma_c$ and $|m| < N_o$
$$\delta_{AP}((a, q', m, l), \sigma) = (\delta(a, \sigma), q', m \uplus [\sigma], tail(l)).$$
3) If $\delta(a, \sigma)!, \sigma \in \Sigma_{uc}$ and $|m| < N_o$
$$\delta_{AP}((a, q', m, l), \sigma) = (\delta(a, \sigma), q', m \uplus [\sigma], l).$$
4) If $\sigma \in m$, $\delta_{OP'}(q', \sigma_o)!$
$$\delta_{AP}((a, q', m, l), \sigma_o) = (a, \delta_{OP'}(q', \sigma_o), m \setminus [\sigma], l).$$
5) If $\sigma \in m$, $\neg \delta_{OP'}(q', \sigma_o)!$
$$\delta_{AP}((a, q', m, l), \sigma_o) = (a, q', m \setminus [\sigma], l). \quad ■$$

Proposition 1 (Finite Asynchronous Plant): For a given plant G which is being supervised and observed through the control and observation channels with limited capacities N_c and N_o, respectively, $\Pi(G, N_c, N_o)$ is a finite automaton. ■

Proof: $\Pi(G, N_c, N_o)$ is finite if it has a finite set of states and a finite set of events. Let us first prove that X is finite. Due to Definition 9, $X = A \times Q' \times M \times L$. To prove that X is a finite set, it is sufficient to guarantee that A, Q', M and L are finite sets because as proved in [18] the Cartesian product of finite sets is finite. A is a finite set since we assumed that the plant is modeled by a finite automaton. For each $q' \in Q'$, we know that $q' \subseteq Q$. So, we should prove that Q is a finite set. $Q = A \times M$ is finite since A and M are finite as the maximum size of M is limited to a finite number N_o. Finally, L is finite since its size is limited to N_c. ■

Example 3: For the plant given in Example 1, we use the projected observed plant OP' depicted in Figure 6 to determine the occurrences of the enabling events in the asynchronous plant for which we do not need the state information of OP. The asynchronous plant is given in Figure 7. Each state indicates the current states of G and OP', and also the events existing in the control and observation channels which are not shown in the figure due to lack of space. ■

The asynchronous plant determines all the feasible enabling commands, executions of events in the plant, and observations of them. An asynchronous supervisor is then synthesized for the asynchronous plant to determine which of the enabling events need to be disabled. The synthesis algorithm is presented in Algorithm 1 in which we use the following additional concepts:

- *Blocking*(AP) is the set of blocking states in AP.
- *Uncon*$_{AP}(BS)$ is the smallest set of states such that
  1) $BS \subseteq Uncon_{AP}(BS)$;

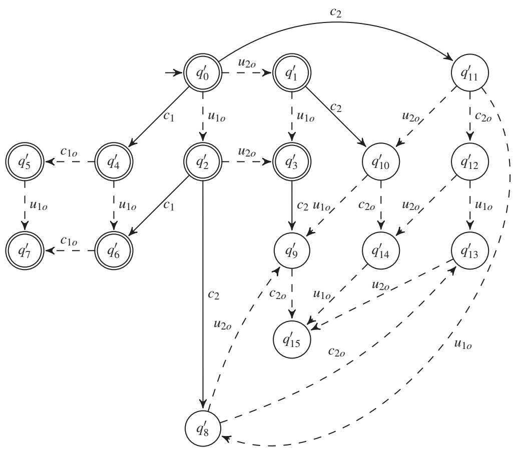


Fig. 6: OP' for G from Example 1.

2) if $\delta_{AP}(x,\sigma) \in Uncon_{AP}(BS)$ for some $x \in X$ and $\sigma \in \Sigma \cup \Sigma_o$, then $x \in Uncon_{AP}(BS)$;

Intuitively this set provides all the states from which a state from BS can be reached in an uncontrollable way.

- Events executed in the plant are unknown to a supervisor until it receives the observations. Therefore, while doing synthesis we need to be careful that the plant events are unobservable in the asynchronous plant, and a supervisor should make the same decision for any two words that it cannot distinguish between. To consider this issue in the synthesis algorithm, we use the function $OBS_{AP}(x) = \{x' \in X \mid \exists w, w' \in \Sigma_{AP}^*, \delta_{AP}(x_0,w) = x \wedge \delta_{AP}(x_0,w') = x' \wedge P_{\Sigma_e \cup \Sigma_o}(w) = P_{\Sigma_e \cup \Sigma_o}(w')\}$ which gives the set of states reachable through the same observation (observationally equivalent states). For instance, $OBS_{AP}(x_0) = \{x_0, x_1, x_2\}$ in Figure 7.

Algorithm 1 Asynchronous supervisory control synthesis
Input: $AP = (X, \Sigma_{ASP}, \delta_{AP}, x_0, X_m)$, Σ_{uc}, Σ_c
Output: $AS = (Y, \Sigma_{AS}, \delta_{AS}, y_0, Y_m)$ or no result

```
 1: $AS \leftarrow AP$
 2: $BS \leftarrow Blocking(AS)$
 3: while $x_0 \notin BS \wedge BS \neq \varnothing$ do
 4:     for $y \in Y \wedge \sigma \in \Sigma_e$ do
 5:         if $\delta_{AS}(y,\sigma) \in Uncon_{AP}(BS)$ then
 6:             for $y' \in OBS_{AP}(y)$ do
 7:                 $\delta_{AS}(y',\sigma) \leftarrow$ undefined
 8:     $AS \leftarrow Reach(AS)$
 9:     $BS \leftarrow Blocking(AS)$
10: if $x_0 \in BS$ then
11:     no result
12: $AS \leftarrow P_{\Sigma_{ASP} \setminus \Sigma}(AS)$
```

Additionally, we could also have assumed that some events from Σ_e are unobservable. In this case, there would be more states which become observationally equivalent, and so the resulting supervisor could be more restrictive since a control command should be disabled at all observationally equivalent states if it needs to be disabled at one of them.

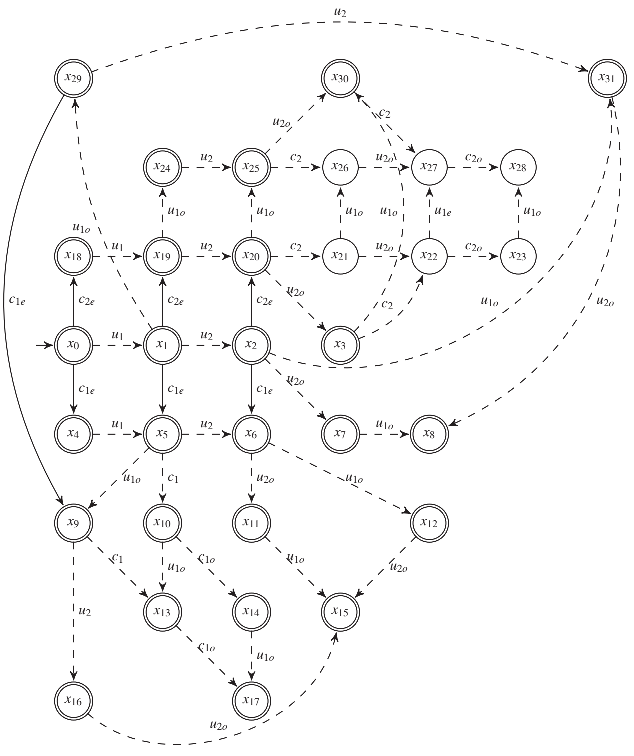


Fig. 7: Asynchronous plant for G from Example 1.

Starting from $AS = AP$, Algorithm 1 changes AS by disabling transitions at line 7, and delivering the reachable part at line 8.

Lemma 3 (Algorithm Termination): The synthesis algorithm presented in Algorithm 1 terminates. ■

Proof: Let the output of the algorithm at iteration i be indicated by $AS(i)$. In each iteration, say iteration i, of Algorithm 1 at least one of its reachable states (from the nonempty set BS) is removed (by making all edges leading into those states undefined (line 7)). Since the automaton is finite state initially, this can only be done finitely often. ■

Example 4: Let us reconsider the plant from Example 1. By applying Algorithm 1, we achieve the asynchronous supervisor given in Figure 8. Note that if c_2 was replaced by c_1 in the plant (Figure 2a), then no result exists, precisely because in the asynchronous setting observation of u_2 is not immediate. ■

Note that for any asynchronous supervisor resulting from Algorithm 1, the asynchronously supervised plant is a finite automaton. Moreover, as discussed in Property 1, controllability of the synthesized asynchronous supervisor is already guaranteed. In Theorem 1, we prove that the asynchronous supervisor obtained from Algorithm 1 guarantees nonblockingness.

Theorem 1 (Nonblocking Asynchronous Supervisor): For

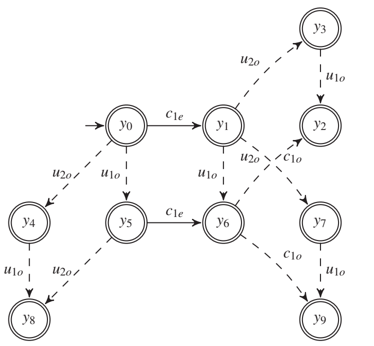


Fig. 8: Asynchronous supervisor for G from Example 1.

$G = (A, \Sigma, \delta, a_0, A_m)$ and $AS = (Y, \Sigma_{AS}, \delta_{AS}, y_0, Y_m)$ achieved from Algorithm 1, $AS|/|G$ is nonblocking. ■

Proof: $AS|/|G = (Z, \Sigma_{ASP}, \delta_{ASP}, z_0, Z_m)$ is nonblocking if for all $z \in Reach(z_0)$ there exists a word $w \in \Sigma_{ASP}^*$ such that $\delta_{ASP}(z, w) \in Z_m$. Take $z \in Reach(z_0)$ where $z = (a, y, m, l)$, then we need to find $w \in \Sigma_{ASP}^*$ for which $\delta(a, P_\Sigma(w)) \in A_m$ and $\delta_{AS}(y, P_{\Sigma_{AS}}(w)) \in Y_m$ since $Z_m = A_m \times Y_m \times M \times L$. From Lemma 2, we know that $\delta_{ASP}(z_0, w) = (\delta(a_0, P_\Sigma(w)), \delta_{AS}(y_0, P_{\Sigma_{AS}}(w)), m, l)$ for some $m \in M, l \in L$. We also have AS is nonblocking when Algorithm 1 terminates successfully which means that there is no blocking state to be removed ($BS = \varnothing$) and also due to Property 1 the projection operator does not change the nonblockingness of an automaton. So, we can say that for $y \in Reach(y_0)$ there exists a word $w' \in \Sigma_{AS}^*$ such that $\delta_{AS}(y, w') \in Y_m$. From Algorithm 1, we have $AS \subseteq P_{\Sigma_{AS}}(AP)$ since we start the algorithm from $AS = AP$ and we remove transitions leading to blocking states, and finally we use the projection to leave out the Σ events. The state $y \subseteq X$ is a set of observationally equivalent states of AP. Since $\delta_{AS}(y, w')!$, we can say that $\forall x \in y, \exists w \in \Sigma_{ASP}^*, \delta_{AP}(x, w) \in X_m$ because otherwise all observationally equivalent transitions have been removed and $\neg\delta_{AS}(y, w')!$. Take $w \in \Sigma_{ASP}^*, \delta_{AP}(x, w) \in X_m$, we only need to prove that $\delta(a, P_\Sigma(w)) \in A_m$. $\delta_{AP}(x, w) \in X_m, X_m = A_m \times Q' \times M \times L$. So, $\delta_{AP}(x, w) \in X_m$ implies that $\delta(a, P_\Sigma(w)) \in A_m$. ■

V. Conclusions and Future Work

In this paper, we investigate the problem of inexact synchronization that may cause a conventionally synthesized supervisor to fail in practice. For this purpose, we first present an asynchronous supervisory control setting in which control commands may arrive in the plant after some delay, and events executed in the plant may not be immediately observed. We also assume that events executed in the plant in some order could be observed in a different order. Moreover, we assume that the plant can execute controllable events only if they are commanded by the supervisor. On the other hand, the plant is free to execute a control command sent by the supervisor or ignore it. In the asynchronous setting, uncontrollable events may be executed in the plant spontaneously, and a supervisor has no role in the occurrence of them. Therefore, controllability is always guaranteed by the definition of asynchronous composition. Furthermore, we present a method for achieving an automaton called the asynchronous plant representing the behavior of the plant in the asynchronous setting. Finally, a synthesis algorithm is presented for obtaining an asynchronous supervisor guaranteeing nonblockingness.

In this paper, we solved the basic synthesis problem to provide nonblockingness. To synthesize an asynchronous supervisor satisfying control requirements, we can first translate the requirements to the plant, and then apply the basic synthesis technique.

For cases with large state spaces, we must deal with the scalability problem of the asynchronous plant, which we aim to consider in future research.

References

[1] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[2] W. M. Wonham, "Supervisory control of discrete-event systems," *Encyclopedia of Systems and Control*, pp. 1396–1404, 2015.

[3] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[4] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, vol. 3. IEEE, 1998, pp. 3305–3310.

[5] F. Basile and P. Chiacchio, "On the implementation of supervised control of discrete event systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 4, pp. 725–739, 2007.

[6] A. B. Leal, D. L. Da Cruz, and M. d. S. Hounsell, "Supervisory control implementation into programmable logic controllers," in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. IEEE, 2009, pp. 1–7.

[7] J. Zaytoon and B. Riera, "Synthesis and implementation of logic controllers–a review," *Annual reviews in control*, vol. 43, pp. 152–168, 2017.

[8] L. Prenzel and J. Provost, "PLC implementation of symbolic, modular supervisory controllers," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 304–309, 2018.

[9] S. Balemi, "Communication delays in connections of input/output discrete event processes," in *1992 31st IEEE Conference on Decision and Control*. IEEE, 1992, pp. 3374–3379.

[10] P. Malik, "Generating controllers from discrete-event models," 2002.

[11] S.-J. Park and K.-H. Cho, "Delay-robust supervisory control of discrete-event systems with bounded communication delays," *IEEE Transactions on Automatic Control*, vol. 51, no. 5, pp. 911–915, 2006.

[12] S. Xu and R. Kumar, "Asynchronous implementation of synchronous discrete event control," in *Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on*. IEEE, 2008, pp. 181–186.

[13] F. Lin, "Control of networked discrete event systems: Dealing with communication delays and losses," *SIAM Journal on Control and Optimization*, vol. 52, no. 2, pp. 1276–1298, 2014.

[14] A. Rashidinejad, M. Reniers, and L. Feng, "Supervisory control of timed discrete-event systems subject to communication delays and non-fifo observations," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 456 – 463, 2018, 14th IFAC Workshop on Discrete Event Systems WODES 2018.

[15] J. E. Hopcroft, R. Motwani, and J. D. Ullman, "Introduction to automata theory, languages, and computation," *ACM SIGACT News*, vol. 32, no. 1, pp. 60–65, 2001.

[16] S. Ware and R. Malik, "The use of language projection for compositional verification of discrete event systems," in *2008 9th International Workshop on Discrete Event Systems*. IEEE, 2008, pp. 322–327.

[17] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[18] T. Jech, *Set theory*. Springer Science & Business Media, 2013.