

# Decentralized Control of Discrete Event Systems with Bounded or Unbounded Delay Communication\*

Stavros Tripakis<sup>†</sup>

## Abstract

We introduce problems of decentralized control with communication, where we explicitly model communication delays. We distinguish two communication models, where delays are, respectively, unbounded, or bounded by a given constant  $k$ . In the  $k$ -bounded-delay model, between the transmission of a message and its reception, the plant can execute at most  $k$  events. In the unbounded-delay model, the plant can execute any number of events between transmission and reception. We show that our framework yields an infinite hierarchy of control problems,

$$\mathcal{CC} = \mathcal{DCC}_0 \supset \mathcal{DCC}_1 \supset \mathcal{DCC}_2 \supset \dots \supset \mathcal{DCUC} \supset \mathcal{DC},$$

where  $\mathcal{CC}$  is the set of control problems solvable with a single controller (centralized case),  $\mathcal{DCC}_k$  is the set of problems solvable with two controllers in a  $k$ -bounded-delay network,  $\mathcal{DCUC}$  is the set of problems solvable with two controllers in an unbounded-delay network, and  $\mathcal{DC}$  is the set of problems solvable with two controllers without communication. The above containments are strict. We prove undecidability of checking the existence of controllers in the unbounded-delay case, or in the case without any communication. Finally, we prove that a decentralized observation problem with bounded-delay communication is decidable.

**Keywords:** discrete-event systems, decentralized control, communication delays, decidability.

## 1 Introduction

Decentralized supervisory control for discrete-event systems has been studied in both (1) the case where the controllers do not communicate at run time, and (2) the case where the controllers can exchange information at run time. We call the first class of problems *decentralized control without communication* (e.g., see [8, 4, 15, 13, 12, 6, 5, 17, 7]) and the second class *decentralized control with communication* (e.g., see [16, 2, 11, 10]). Both classes are worth studying: decentralized control without communication is sometimes imposed, in the case where no network is available; on the other hand, communication is often necessary, in the case where the controllers do not have enough local information to achieve their objective.

So far, most of the work on decentralized control with communication [16, 2, 11, 10] has been based on the assumption that controllers can exchange information with *zero delay*, in

---

\*This work has been partially supported by the European IST project “Next TTA” under project No IST-2001-32111.

<sup>†</sup>VERIMAG, Centre Equation, 2, avenue de Vignate, 38610 Gières, FRANCE. E-mail: tripakis@imag.fr.

other words, that the plant cannot perform any action between the transmission and reception of a message among controllers. This assumption, though it helps the study of *what* the communication policy should be (for example, how can transmissions be reduced so that only absolutely necessary information is communicated), is often unrealistic in practice, where controllers must function in a network with delays.

In this paper, we study problems of decentralized control with communication, where the communication delays are explicitly modeled and taken into account. In particular, we distinguish two communication models, namely, where delays are either bounded by a given constant  $k$ , or unbounded. In the  $k$ -bounded-delay model, between the transmission of a message and its reception, the plant can execute at most  $k$  events. In the unbounded-delay model, the plant can execute any number of events between transmission and reception.

We make a number of assumptions. First, communication is *lossless*, that is, all messages are eventually delivered within a finite (possibly unbounded) delay. Second, communication is *FIFO*, that is, if message  $a$  is sent before message  $b$ , then  $a$  will be delivered before  $b$ . Third, the communication policy is fixed to the following simple policy: each controller transmits the events it observes in the exact order it observes them, and nothing else. Fourth, we consider a simple model of specifications, in terms of *responsiveness* properties, of the form “event  $a$  is always followed by event  $b$ ” (the reason will be clarified below). Finally, we consider for simplicity the case of only two controllers (this is not an essential assumption).

Our results are of two types. First, we show that our modeling framework results in the (infinite) hierarchy of control problems, shown in Figure 1. Second, we provide a set of undecidability and decidability results. We now discuss our results in more detail.

Figure 1 is to be interpreted as follows.  $\mathcal{CC}$  denotes the class of control problems that can be solved with a central controller.  $\mathcal{DCC}_k$  denotes the class of control problems that can be solved with two controllers with  $k$ -bounded-delay communication.  $\mathcal{DCUC}$  denotes the class of control problems that can be solved with two controllers with unbounded-delay communication.  $\mathcal{DC}$  denotes the class of control problems that can be solved with two controllers without communication. Then, we show that

$$\mathcal{CC} = \mathcal{DCC}_0 \supset \mathcal{DCC}_1 \supset \mathcal{DCC}_2 \supset \cdots \supset \mathcal{DCUC} \supset \mathcal{DC},$$

where the inclusions are strict.

$\mathcal{CC} = \mathcal{DCC}_0$  means that every problem that can be solved with a single controller can also be solved with two controllers communicating with zero-delay, and vice-versa (recall that we assume the “transmit everything you observe” policy).  $\mathcal{DCC}_{k+1} \subseteq \mathcal{DCC}_k$  means that every problem that can be solved with  $(k+1)$ -bounded-delay communication can also be solved with  $k$ -bounded-delay communication, in fact, using the same controllers. (Note that this is not necessarily true in other frameworks, for instance, see Footnote 5 in Section 4.) The other inclusions are similar. The fact that the inclusions are strict means that there are problems which can be solved in a  $k$ -bounded-delay network, but cannot be solved in a  $(k+1)$ -bounded-delay network, for  $k = 0, 1, 2, \dots$ , and that there are problems which can be solved with unbounded-delay communication, but cannot be solved in the absence of any communication.

Regarding decidability, a number of results already exist. Some versions of the decentralized control problem are decidable [13, 12], while others have recently been shown to be undecidable [7, 14]. In particular, checking the existence of (and constructing, if they exist) *non-blocking* controllers [9, 3], such that  $A \subseteq L(G/C_1 \wedge C_2) \subseteq E$  (resp.,  $L_m(G/C_1 \wedge C_2) = E$ ),

is shown to be decidable in [13], where  $A$  and  $E$  are given regular languages,  $G$  is a (finite-state) plant,  $C_1$  and  $C_2$  are the controllers,  $(G/C_1 \wedge C_2)$  is the *conjunctively* [17] controlled system without communication,  $L(\cdot)$  is the *unmarked* (prefix-closed) language of  $(G/C_1 \wedge C_2)$  and  $L_m(\cdot)$  is the *marked* language of  $(G/C_1 \wedge C_2)$ . In [14], it was shown that checking the existence of non-blocking controllers, such that  $L_m(G/C_1 \wedge C_2) \subseteq E$ , is undecidable.<sup>1</sup>

In this paper, we show that it is undecidable to check the existence of two controllers such that a set of responsiveness properties is satisfied, in both cases of (1) unbounded-delay communication, (2) no communication. Both proofs are based on the reduction of a decentralized observation problem to control problems (1) and (2). The decentralized observation problem is the problem of checking *joint observability* [14]. The reduction consists in showing that, if we knew how to solve the decentralized control problem (1) or (2), then we would know how to check joint observability. However, checking joint observability is shown to be undecidable in [14].

We believe that the decentralized control problem with bounded-delay communication is decidable. Towards such a result, we prove decidability of *joint observability with bounded-delay communication*. The latter is a modification of the joint observability definition, to take into account the fact that the observers communicate to each other their observations, and these observations are delivered with bounded delay.

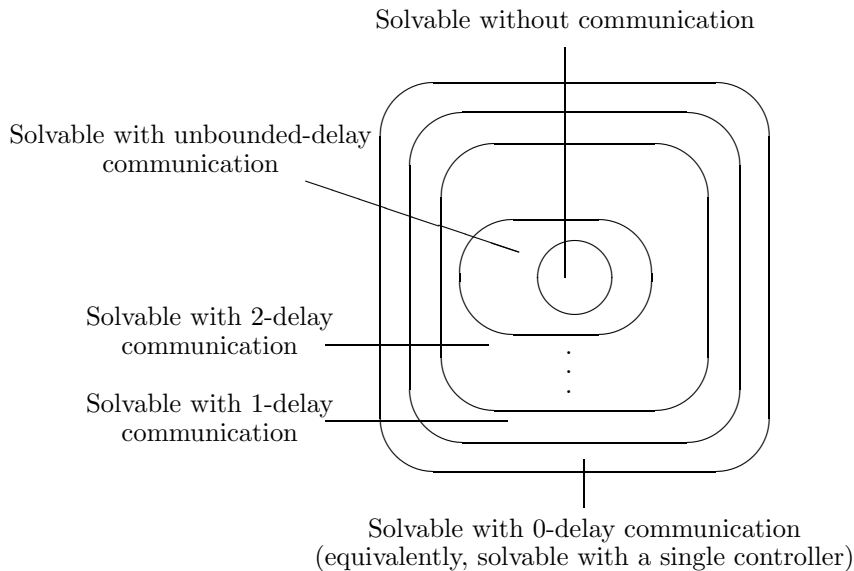


Figure 1: Hierarchy of decentralized control problems with bounded communication.

The paper is organized as follows. Section 2 contains preliminary definitions. In Section 3 we define centralized and decentralized control problems. In Section 4 we provide the results on the hierarchy. Section 5 contains the undecidability results. In Section 6 we define joint observability with bounded-delay communication and prove its decidability. In Section 7 we present our conclusions.

<sup>1</sup>It is also worth noting that the setting of [13, 12] is slightly more general than the one considered in [14], in the sense that [13, 12] allow controllers to have their own acceptance conditions (accepting states), whereas in [14] it is assumed that all states of the controllers are accepting.

## 2 Preliminaries

$\mathbb{N}$  will denote the set of natural numbers.

Let  $\Sigma$  be a finite alphabet.  $\Sigma^*$  denotes the set of all finite strings over  $\Sigma$ ,  $\epsilon$  denotes the empty string, and  $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$ .  $\Sigma^\omega$  denotes the set of all infinite strings over  $\Sigma$ .

Given two strings  $\rho$  and  $\rho'$ , such that  $\rho$  is finite,  $\rho\rho'$  or  $\rho \cdot \rho'$  is the *concatenation* of  $\rho$  and  $\rho'$ . Given a (finite or infinite) string  $\rho$ , a *prefix* of  $\rho$  is a finite string  $\pi$  such that  $\rho = \pi \cdot \tau$ , for some  $\tau$ . We say that  $\pi$  is a *strict prefix* of  $\rho$  if  $\tau \neq \epsilon$ .

Let  $\rho$  be a (finite or infinite) string over  $\Sigma$ . Given  $\Gamma \subseteq \Sigma$ , we define the *projection* of  $\rho$  to  $\Gamma$ , denoted  $P_\Gamma(\rho)$ , as the string obtained from  $\rho$  by erasing all letters not in  $\Gamma$ . For example, if  $\Sigma = \{a, b, c\}$  and  $\Gamma = \{a, c\}$ , then  $P_\Gamma(abbcbaacb) = acac$ . For a set of (finite or infinite) strings  $L$ ,  $P_\Gamma(L) = \{P_\Gamma(\rho) \mid \rho \in L\}$ .

A *responsiveness property* over some alphabet  $\Sigma$  is a formula of the form  $a \rightsquigarrow b$ , where  $a, b \in \Sigma$ . Consider a (finite or infinite) string  $\rho$  over  $\Sigma$ ,  $\rho = c_0 c_1 c_2 \dots$ . We say that  $\rho$  *satisfies*  $a \rightsquigarrow b$ , denoted  $\rho \models a \rightsquigarrow b$ , if  $b$  occurs after every  $a$  in  $\rho$ , that is, for all  $i$ , if  $c_i = a$ , then there exists  $j > i$ , such that  $c_j = b$ . For example,  $acbaab$  satisfies  $a \rightsquigarrow b$ , whereas  $acc$  does not.<sup>2</sup> A set of (finite or infinite) strings  $L$  satisfies a property if every string in  $L$  satisfies the property. A *specification* is a set of properties.  $L$  satisfies a specification  $\phi$ , denoted  $L \models \phi$ , if  $L$  satisfies every property in  $\phi$ . Note that if  $L' \subseteq L$  and  $L \models \phi$ , then  $L' \models \phi$ .

A *non-deterministic automaton* over an alphabet  $\Sigma$  is a tuple  $H = (S, q_0, \Sigma, \Delta)$ , where  $S$  is the set of states,  $q_0 \in S$  is the initial state, and  $\Delta : S \times \Sigma \rightarrow 2^S$  is the non-deterministic transition function ( $\Delta$  is a total function, which may return  $\emptyset$ ). We write  $s \xrightarrow{a} s'$  if  $s' \in \Delta(s, a)$ . If, for all  $s \in S$ ,  $a \in \Sigma$ ,  $\Delta(s, a)$  contains at most one element, the automaton is called *deterministic* (in this case, the transition function will often be denoted by  $\delta$ ). If  $\Delta(s, a)$  is never empty, the automaton is called *receptive*. A state  $s$  is a *deadlock* if for all  $a \in \Sigma$ ,  $\Delta(s, a) = \emptyset$ . Given a finite string  $\rho = a_1 \dots a_k \in \Sigma^*$ , we define  $\Delta(\rho)$  to be the set of all states  $s \in S$ , for which there exists a sequence of states  $s_0, s_1, \dots, s_k \in S$ , such that  $s_0 = q_0$ ,  $s_k = s$  and  $s_{i+1} \in \Delta(s_i, a_{i+1})$ . If  $\Delta(\rho)$  is non-empty, we say that  $\rho$  is *generated* by  $H$ . Given an infinite string  $\pi$ , we say that  $\pi$  is generated by  $H$  if every finite prefix  $\rho$  of  $\pi$  is generated by  $H$ . A string  $\rho$  generated by  $H$  is *maximal* if, either  $\rho$  is infinite, or  $\rho$  is finite and for some  $s \in \Delta(\rho)$ ,  $s$  is a deadlock.  $L_{\max}(H)$  is the set of all (finite or infinite) maximal strings generated by  $H$ .

A *deterministic automaton over  $\Sigma$  with outputs in  $\Gamma$* , where  $\Gamma$  is an alphabet (not necessarily related to  $\Sigma$ ), is a tuple  $C = (S, q_0, \Sigma, \delta, \Gamma, \Lambda)$ , such that  $(S, q_0, \Sigma, \delta)$  is a deterministic automaton over  $\Sigma$ , and  $\Lambda : S \rightarrow 2^\Gamma$  is the output function (total).

## 3 Centralized and decentralized control problems

We will define four control problems, namely, centralized control, decentralized control without communication, decentralized control with unbounded-delay communication and decentralized control with bounded-delay communication. To do that, we have to define what is a plant, what

---

<sup>2</sup>We choose to model specifications by responsiveness properties, because these are simple properties that capture the essence of our results. With a little bit of extra modeling, responsiveness can express other properties as well. For instance, an *invariance* property such as “event  $a$  never occurs” can be equivalently expressed by the responsiveness property  $a \rightsquigarrow b$ , where  $b$  is a new letter that never occurs. Then,  $a \rightsquigarrow b$  is satisfied iff  $a$  never occurs.

are the controllers, what is communication and what is the effect of one or more controllers (communicating or not) on the plant.

We will fix an alphabet  $\Sigma$ , to be used through the whole section. In all cases, the plant will be modeled as a finite-state deterministic automaton  $G$  over  $\Sigma$ ,  $G = (S_G, q_{0G}, \Sigma, \delta_G)$ . The controllers will be modeled as receptive deterministic automata with outputs.

### 3.1 Centralized control

Let  $\Sigma_O, \Sigma_C \subseteq \Sigma$ .  $\Sigma_O$  models the set of events of the plant that are *observable* by the controller and  $\Sigma_C$  models the set of *controllable* events (i.e., the events that can be disabled by the controller). The centralized control architecture is depicted in Figure 2.

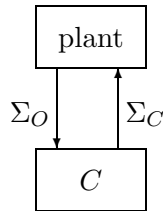


Figure 2: The centralized control architecture.

The controller  $C = (S_C, q_{0C}, \Sigma_O, \delta_C, \Sigma_C, \Lambda_C)$  is a receptive deterministic automaton over  $\Sigma_O$  with outputs in  $\Sigma_C$ . The intended meaning is that, when  $C$  is in state  $s$ , it “allows” only events in  $\Lambda_C(s)$ , i.e., disables all events in  $\Sigma_C - \Lambda_C(s)$ . Note that  $\Lambda_C(s)$  might be empty (no controllable event enabled) or equal to  $\Sigma_C$  (no controllable event disabled).

The *controlled system*, denoted  $(G/C)$ , is defined to be the deterministic automaton  $(S, q_0, \Sigma, \delta)$ , where  $S = S_G \times S_C$ ,  $q_0 = (q_{0G}, q_{0C})$ , and  $\delta$  is defined as follows. Given  $s = (s_G, s_C) \in S$  and  $a \in \Sigma$ ,

- if  $\delta_G(s_G, a)$  is undefined, then  $\delta(s, a)$  is undefined,
- if  $\delta_G(s_G, a)$  is defined and  $a \in \Sigma_C - \Lambda_C(s_C)$ , then  $\delta(s, a)$  is undefined,
- otherwise,  $\delta(s, a) = (\delta_G(s_G, a), s'_C)$ , where  $s'_C = \delta_C(s_C, a)$  if  $a \in \Sigma_O$  and  $s'_C = s_C$  if  $a \notin \Sigma_O$ .

**Definition 3.1 (centralized control problem)** *Given a finite-state deterministic automaton  $G$  over  $\Sigma$ , a specification  $\phi$  over  $\Sigma$ , and  $\Sigma_O, \Sigma_C \subseteq \Sigma$ , does there exist a receptive deterministic automaton  $C$  over  $\Sigma_O$  with outputs in  $\Sigma_C$ , such that  $L_{\max}(G/C) \models \phi$ .*

### 3.2 Decentralized control without communication

Let  $\Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C} \subseteq \Sigma$ .  $\Sigma_{iO}$  (resp.,  $\Sigma_{iC}$ ) is the set of events observable (resp., controllable) to controller  $i$ . In case some event  $a \in \Sigma_{1C} \cap \Sigma_{2C}$ , that is,  $a$  is controllable by both controllers, the *conjunctive* decision policy is assumed, that is, the event is enabled iff both controllers enable it. The conjunctive decentralized control architecture without communication is depicted in Figure 3.

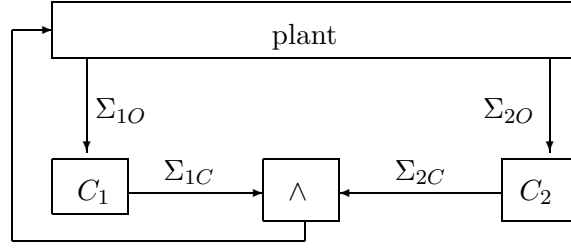


Figure 3: The (conjunctive) decentralized control architecture without communication.

For  $i = 1, 2$ , controller  $C_i = (S_{iC}, q_{iC}, \Sigma_{iO}, \delta_{iC}, \Sigma_{iC}, \Lambda_{iC})$  is a receptive deterministic automaton over  $\Sigma_{iO}$  with outputs in  $\Sigma_{iC}$ .

The *conjunctively controlled system without communication*, denoted  $(G/C_1 \wedge C_2)$ , is defined to be the deterministic automaton  $(S, q_0, \Sigma, \delta)$ , where  $S = S_G \times S_{1C} \times S_{2C}$ ,  $q_0 = (q_{0G}, q_{1C}, q_{2C})$ , and  $\delta$  is defined as follows. Given  $s = (s_G, s_1, s_2) \in S$  and  $a \in \Sigma$ :

- if  $\delta_G(s_G, a)$  is undefined then  $\delta(s, a)$  is undefined,
- if  $\delta_G(s_G, a)$  is defined and there is some  $i = 1, 2$  such that  $a \in \Sigma_{iC} - \Lambda_{iC}(s_i)$  then  $\delta(s, a)$  is undefined,
- otherwise,  $\delta(s, a) = (\delta_G(s_G, a), s'_1, s'_2)$ , where, for  $i = 1, 2$ ,  $s'_i = \delta_{iC}(s_i, a)$  if  $a \in \Sigma_{iO}$  and  $s'_i = s_i$  if  $a \notin \Sigma_{iO}$ .

**Definition 3.2 (decentralized control problem without communication)** *Given a finite-state deterministic automaton  $G$  over  $\Sigma$ , a specification  $\phi$  over  $\Sigma$ , and  $\Sigma_{1O}, \Sigma_{1C}, \Sigma_{2O}, \Sigma_{2C} \subseteq \Sigma$ , do there exist receptive deterministic automata  $C_i$  over  $\Sigma_{iO}$  with outputs in  $\Sigma_{iC}$ , for  $i = 1, 2$ , such that  $L_{\max}(G/C_1 \wedge C_2) \models \phi$ .*

### 3.3 Queues of bounded or unbounded delay

Before defining the decentralized control problems with communication, we introduce the useful notion of a *FIFO queue with delays* (queue, for short). A queue over some alphabet  $\Gamma$  is an ordered list of pairs  $(a, i)$ , where  $a \in \Gamma$  and  $i \in \mathbf{N}$ .<sup>3</sup> We require that if  $(a, i)$  is before  $(b, j)$  in a queue, then  $i \leq j$ . For example,  $[(a, 2), (b, 3)]$  is a queue with two elements. We denote by  $\square$  the empty queue.

The first element of a non-empty queue  $Q$  is its *head*, denoted  $\text{head}(Q)$ , and the last element is its *tail*, denoted  $\text{tail}(Q)$ . If  $Q \neq \square$  and  $\text{head}(Q) = (a, i)$ , where  $i > 0$ , then  $Q - 1$  denotes the new queue obtained by decrementing all indices in the elements of  $Q$  by one. For example, if  $Q = [(a, 2), (b, 3)]$  then  $Q - 1 = [(a, 1), (b, 2)]$ . By convention, if  $Q = \square$ , we let  $Q - 1 = \square$ . If  $Q \neq \square$  and  $\text{head}(Q) = (a, 0)$ , then we say that  $Q$  is *ready*, and we define  $\text{pop}(Q)$  to be the new queue obtained by removing the head of  $Q$ . So,  $\text{pop}([(a, 2), (b, 3)]) = [(b, 3)]$ . By convention, an empty queue is not ready. Let  $\text{max}(Q)$  denote the maximum  $i$  such that  $(a, i)$

<sup>3</sup>The index  $i$  can be seen as the *time-to-live* field, that is, how many time steps remain until the message is delivered.

is in  $Q$ . If  $Q$  is empty, we let  $\max(Q) = 0$ . Notice that, by definition, the tail of  $Q$  is some element  $(b, \max(Q))$ .

We define the operator  $\text{push}(Q, a)$ , which takes a queue  $Q$  and a message  $a$  and returns an infinite set of queues,  $\text{push}(Q, a) = \{Q_{\max(Q)}, Q_{\max(Q)+1}, \dots\}$ , such that, for each  $i \geq \max(Q)$ ,  $Q_i$  is obtained by appending the new tail  $(a, i)$  to  $Q$ . For example,

$$\text{push}([(b, 1)], a) = \{[(b, 1), (a, 1)], [(b, 1), (a, 2)], [(b, 1), (a, 3)], \dots\}.$$

We also define the parameterized operator  $\text{push}_k(Q, a)$ , for  $k \in \mathbb{N}$ , which returns the finite set of queues,  $\text{push}_k(Q, a) = \{Q_{\max(Q)}, \dots, Q_k\}$  (if  $\max(Q) > k$ , then  $\text{push}_k(Q, a) = []$ ). For example,

$$\text{push}_2([(b, 1)], a) = \{[(b, 1), (a, 1)], [(b, 1), (a, 2)]\}.$$

The **pop** operation models the network delivering a message.  $Q - 1$  models the messages “aging” by one time step. The **push** operations model the network *scheduling* a message to be delivered later on: since it is not known exactly after how many steps the message will be delivered, both **push** and  $\text{push}_k$  are non-deterministic. In an unbounded-delay network, **push** will be used, since a message may be delivered after an arbitrary (though finite) number of steps. On the other hand, in a network where a message is guaranteed to be delivered after at most  $k$  steps,  $\text{push}_k$  will be used. Notice that, by the FIFO property of the queue, a message cannot be delivered before all previous messages in the queue are delivered.

**Lemma 3.3** *For any queue  $Q$  over some alphabet  $\Gamma$ , for any  $a \in \Gamma$ , and for any  $k \in \mathbb{N}$ ,*

1.  $\text{push}_k(Q, a) \subseteq \text{push}_{k+1}(Q, a) \subseteq \text{push}(Q, a)$ ,
2. for any  $Q' \in \text{push}_k(Q, a)$ , for any element  $(b, i)$  of  $Q'$ ,  $i \leq k$ .

### 3.4 Decentralized control with unbounded-delay communication

Let us first introduce some notation. Given an event  $a$  in some alphabet  $\Sigma$ ,  $\hat{a}$  denotes another event, called the *message version* of  $a$  ( $\hat{a}$  will model the message sent by a controller that observes  $a$ ). Given an alphabet  $\Gamma$ , we define  $\hat{\Gamma} = \{\hat{a} \mid a \in \Gamma\}$ . Given  $\rho \in \Gamma^*$ ,  $\rho = a_1 \cdots a_l$ ,  $\hat{\rho}$  denotes the string  $\hat{a}_1 \cdots \hat{a}_l \in \hat{\Gamma}^*$ .

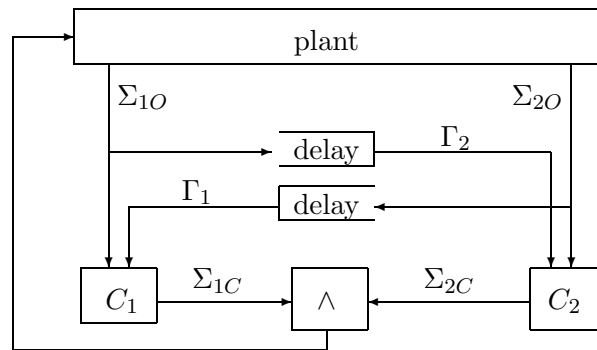


Figure 4: The (conjunctive) decentralized control architecture with communication.

Let  $\Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C} \subseteq \Sigma$ . The idea is that controller  $C_1$  will observe its own observable events,  $\Sigma_{1O}$ , plus the message events it receives from  $C_2$ ,  $\widehat{\Sigma}_{2O}$ . Note that if  $a \in \Sigma_{1O} \cap \Sigma_{2O}$  (i.e.,  $a$  is observable by both  $C_1$  and  $C_2$ ) then  $C_1$  will observe  $a$  directly (the moment it occurs) and will later receive  $\widehat{a}$  (the message sent by  $C_2$ ). The situation is symmetric for  $C_2$ . All message events are received in order, without loss, and within some finite (though unbounded) delay.

Let  $\mathcal{Q}_i$  be the set of all possible queues over  $\widehat{\Sigma}_{jO}$ , for  $i, j = 1, 2, i \neq j$ . That is, a queue  $Q_1 \in \mathcal{Q}_1$  will hold the messages sent from  $C_2$  to  $C_1$ , and  $Q_2 \in \mathcal{Q}_2$  will hold the messages sent from  $C_1$  to  $C_2$ . Let  $\Sigma'_{1O} = \Sigma_{1O} \cup \widehat{\Sigma}_{2O}$  and  $\Sigma'_{2O} = \Sigma_{2O} \cup \widehat{\Sigma}_{1O}$ . For  $i = 1, 2$ , controller  $C_i = (S_{iC}, q_{iC}, \Sigma'_{iO}, \delta_{iC}, \Sigma_{iC}, \Lambda_{iC})$  is a receptive deterministic automaton over  $\Sigma'_{iO}$  with outputs in  $\Sigma_{iC}$ . Let  $\mathbf{t}$  be a new event, and define  $\Sigma' = \Sigma \cup \widehat{\Sigma}_{1O} \cup \widehat{\Sigma}_{2O} \cup \{\mathbf{t}\}$ .

The conjunctive decentralized control architecture with communication is depicted in Figure 4. Note that, since we fix the communication policy to “transmit everything you observe”, we do not need to model communication actions of the controllers explicitly. Instead, every event observed by one controller is automatically transmitted to the other controller.

The *conjunctively controlled system with unbounded-delay communication*, denoted  $(G/C_1 \wedge_{\infty} C_2)$ , is defined to be the non-deterministic automaton  $(S, q_0, \Sigma', \Delta)$ , where  $S = S_G \times S_{1C} \times S_{2C} \times \mathcal{Q}_1 \times \mathcal{Q}_2$ ,  $q_0 = (q, q_{1C}, q_{2C}, [], [])$ , and  $\Delta$  is defined as follows. Given states  $s = (s_G, s_1, s_2, Q_1, Q_2)$  and  $s' = (s'_G, s'_1, s'_2, Q'_1, Q'_2)$ ,  $\Delta$  contains the following types of transitions:

1.  $s \xrightarrow{\widehat{a}} s'$ : if some queue  $Q_i$  is ready with  $\text{head}(Q_i) = (\widehat{a}, 0)$ , in which case,  $s'_G = s_G$ ,  $Q'_i = \text{pop}(Q_i)$ ,  $Q'_j = Q_j$ ,  $s'_i = \delta_{iC}(s_i, \widehat{a})$ ,  $s'_j = s_j$ , where  $i, j = 1, 2, j \neq i$ ,
2. if no queue is ready,
  - (a)  $s \xrightarrow{b} s'$ : if for some  $b \in \Sigma$ ,  $\delta_G(s_G, b)$  is defined and there is no  $i = 1, 2$  such that  $b \in \Sigma_{iC} - \Lambda(s_i)$ , in which case,  $s'_G = \delta_G(s_G, b)$  and, for  $i = 1, 2$ , if  $b \in \Sigma_{iO}$ , then  $s'_i = \delta_{iC}(s_i, b)$ ,  $Q'_j \in \text{push}(Q_j - 1, \widehat{b})$ , otherwise,  $s'_i = s_i$ ,  $Q'_j = Q_j - 1$ , where  $j = 1, 2, j \neq i$ ,
  - (b)  $s \xrightarrow{\mathbf{t}} s'$ : if there is no  $b$  such that clause 2(a) is satisfied, and some queue is non-empty, in which case,  $s'_G = s_G$ ,  $s'_i = s_i$  and  $Q'_i = Q_i - 1$ , for  $i = 1, 2$ .

Clause 1 corresponds to the case where a message is delivered from one of the queues: this happens as soon as a queue is ready. Clause 2(a) corresponds to the case where the plant moves: every such move is assumed to take one time step, so that it results in the aging of both queues by one step; moreover, if the plant executes  $b$ , then, for each controller  $C_i$ , if  $b$  is observable by  $C_i$ , then  $C_i$  will move according to  $b$ , and  $\widehat{b}$  will be (automatically) sent to the other controller  $C_j$ . Clause 2(b) corresponds to the case where the plant is blocked and there is at least one queue which is non-empty but not ready: in this case, we let time elapse, so that the queue eventually becomes ready and delivers the message. Time elapse is modeled by the special event  $\mathbf{t}$  which takes one time step.<sup>4</sup>

<sup>4</sup>We could have given the definition of  $(G/C_1 \wedge_{\infty} C_2)$  in another way, using queues whose elements are simply events (and not events with a time-to-live field) and adding some *fairness* constraints to express the fact that every message is eventually delivered. However, we chose the above formulation, because it extends very easily to the bounded-delay case, as we shall see in Section 3.5.

**Definition 3.4 (decentralized control problem with unbounded-delay communication)**

Given a finite-state deterministic automaton  $G$  over  $\Sigma$ , a specification  $\phi$  over  $\Sigma$ , and  $\Sigma_{1O}, \Sigma_{1C}, \Sigma_{2O}, \Sigma_{2C} \subseteq \Sigma$ , do there exist receptive deterministic automata  $C_i$  over  $\Sigma_{iO} \cup \widehat{\Sigma_{jO}}$  with outputs in  $\Sigma_{iC}$ , for  $i, j = 1, 2, j \neq i$ , such that  $L_{\max}(G/C_1 \wedge_{\infty} C_2) \models \phi$ .

**3.5 Decentralized control with bounded-delay communication**

Let  $\Sigma_{iO}, \Sigma_{iC}, C_i$  and  $\Sigma'$  be as in Section 3.4. In addition, we are given a natural constant  $k \in \mathbb{N}$ . The conjunctive decentralized control architecture with bounded-delay communication is the same as the one shown in Figure 4. The difference is that delays in this case are bounded by  $k$ .

The *conjunctively controlled system with  $k$ -bounded-delay communication*, denoted  $(G/C_1 \wedge_k C_2)$ , is defined in the same way as  $(G/C_1 \wedge_{\infty} C_2)$ , except that  $\text{push}(\cdot, \cdot)$  is replaced by  $\text{push}_k(\cdot, \cdot)$ , in the definition of the transition function  $\Delta$ .

**Definition 3.5 (decentralized control problem with bounded-delay communication)**

Given  $k \in \mathbb{N}$ , a finite-state deterministic automaton  $G$  over  $\Sigma$ , a specification  $\phi$  over  $\Sigma$ , and  $\Sigma_{1O}, \Sigma_{1C}, \Sigma_{2O}, \Sigma_{2C} \subseteq \Sigma$ , do there exist receptive deterministic automata  $C_i$  over  $\Sigma_{iO} \cup \widehat{\Sigma_{jO}}$  with outputs in  $\Sigma_{iC}$ , for  $i, j = 1, 2, j \neq i$ , such that  $L_{\max}(G/C_1 \wedge_k C_2) \models \phi$ .

**4 A hierarchy of centralized and decentralized control problems**

In this section, we present a number of results which give rise to the hierarchy of centralized and decentralized control problems depicted in Figure 1.

Let us first introduce some notation. We will represent a decentralized control problem by a tuple  $(G, \Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C}, \phi)$ . To be able to compare, we will also represent a centralized control problem by the same type of tuple, with the convention that  $\Sigma_O = \Sigma_{1O} \cup \Sigma_{2O}$  and  $\Sigma_C = \Sigma_{1C} \cup \Sigma_{2C}$ .

We will denote by  $\mathcal{CC}$  the class of all control problems  $(G, \Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C}, \phi)$  for which there exists a centralized solution, that is, a controller  $C$  over  $\Sigma_{1O} \cup \Sigma_{2O}$  with outputs in  $\Sigma_{1C} \cup \Sigma_{2C}$ , such that  $L_{\max}(G/C) \models \phi$ . Similarly, we will denote by  $\mathcal{DCC}_k$ ,  $\mathcal{DCUC}$  and  $\mathcal{DC}$  the classes of control problems for which there exists, respectively, a decentralized solution with  $k$ -bounded-delay communication, a decentralized solution with unbounded-delay communication and a decentralized solution without communication.

We first show that the decentralized control problem with zero-delay communication is equivalent to the centralized control problem.

**Proposition 4.1**  $\mathcal{CC} = \mathcal{DCC}_0$ .

**Sketch of proof** Zero delay means that, each time the plant generates an observable event  $a$ , say,  $a \in \Sigma_{1O}$ , the message  $\hat{a}$  will be delivered to  $C_2$  before the plant has time to generate another event. This is essentially the same as if  $a$  was directly observable by  $C_2$ : indeed, a sequence of two immediately successive moves  $a, \hat{a}$  by the two controllers, is equivalent to a

single move  $a$  by a single controller. ■

The second set of results shows that every decentralized control problem that can be solved with  $(k + 1)$ -bounded-delay communication (resp., unbounded-delay communication) can also be solved with  $k$ -bounded-delay communication, for any  $k \in \mathbb{N}$ , and using the same controllers. This is to be expected, since a network of at most  $k$  delay is more deterministic than a network of at most  $k + 1$  delay, or a network of unbounded delay. Therefore, if controllers exist for the  $(k + 1)$ -bounded-delay network (or the unbounded-delay network), the same controllers must work in a  $k$ -bounded-delay network, since all the behaviors of the latter are also behaviors of the former.<sup>5</sup> We formalize these observations in what follows.

**Lemma 4.2** *For any  $k \in \mathbb{N}$ , plant  $G$ , and controllers  $C_1, C_2$ , if  $s \xrightarrow{a} s'$  is a transition in  $(G/C_1 \wedge_k C_2)$ , then it is also a transition in  $(G/C_1 \wedge_{k+1} C_2)$  and in  $(G/C_1 \wedge_\infty C_2)$ .*

**Proof** By induction, with basis the fact that all three systems,  $(G/C_1 \wedge_k C_2)$ ,  $(G/C_1 \wedge_{k+1} C_2)$  and  $(G/C_1 \wedge_\infty C_2)$ , have the same initial state, and using part 1 of Lemma 3.3 in the induction step. ■

**Lemma 4.3** *For any  $k \in \mathbb{N}$ ,  $G$ ,  $C_1$  and  $C_2$ , if  $s$  is a reachable deadlock state in  $(G/C_1 \wedge_k C_2)$ , then it is also a reachable deadlock state in  $(G/C_1 \wedge_{k+1} C_2)$  and  $(G/C_1 \wedge_\infty C_2)$ .*

**Proof** We prove the implication only for  $(G/C_1 \wedge_{k+1} C_2)$ . The proof for  $(G/C_1 \wedge_\infty C_2)$  is similar. For ease of notation, we define  $H_k = (G/C_1 \wedge_k C_2)$  and  $H_{k+1} = (G/C_1 \wedge_{k+1} C_2)$ .

Consider a reachable deadlock state  $s$  in  $H_k$ . By Lemma 4.2,  $s$  is reachable in  $H_{k+1}$  as well. Assume  $s$  is not a deadlock in  $H_{k+1}$ , that is,  $H_{k+1}$  has some transition  $s \xrightarrow{a} s'$ . Let  $s = (s_G, s_1, s_2, Q_1, Q_2)$  and  $s' = (s'_G, s'_1, s'_2, Q'_1, Q'_2)$ .

Consider first the case  $a \in \widehat{\Sigma}_{1O} \cup \widehat{\Sigma}_{2O} \cup \{t\}$ . We can see that, in this case, the definition of  $\Delta$  (clauses 1 or 2(b)) is the same for both  $H_k$  and  $H_{k+1}$ , thus,  $s \xrightarrow{a} s'$  is also a transition in  $H_k$ , which contradicts that  $s$  is a deadlock in  $H_k$ .

Consider now the case  $a \in \Sigma$  (clause 2(a)). The definitions of the transition relation differ between  $H_k$  and  $H_{k+1}$  only in case  $a \in \Sigma_{1O} \cup \Sigma_{2O}$ . Assume  $a \in \Sigma_{1O} - \Sigma_{2O}$  (the other cases are similar). Then, we have  $Q'_1 = Q_1 - 1$  and  $Q'_2 \in \text{push}_{k+1}(Q_2 - 1, \hat{a})$ . Recall that  $s$  is reachable in  $H_k$ , that is,  $Q_2$  has been obtained (potentially) using the  $\text{push}_k(\cdot, \cdot)$  operator, and not  $\text{push}_{k+1}(\cdot, \cdot)$ . Therefore, by part 2 of Lemma 3.3, for every element  $(b, j)$  of  $Q_2$ , we have  $j \leq k$ , and  $\text{push}_k(Q_2 - 1, \hat{a})$  is not empty. That is, for some  $Q''_2 \in \text{push}_k(Q_2 - 1, \hat{a})$ ,  $H_k$  has a transition  $s \xrightarrow{a} s''$ , where  $s'' = (s'_G, s'_1, s'_2, Q'_1, Q''_2)$ . This contradicts the hypothesis that  $s$  is a deadlock in  $H_k$ . ■

---

<sup>5</sup>Surprisingly, this intuition is incorrect in some modeling frameworks. For instance, if we require *non-blockingness* (the requirement that from each reachable state it is possible to reach an accepting state [9, 3]), then there are examples of controllers and plant which are non-blocking on a  $(k + 1)$ -bounded-delay network, but blocking on a  $k$ -bounded-delay network. This happens when the controllers make the wrong decision (e.g., keeping the system in a loop without ever reaching an accepting state) when they receive messages early, but the correct decision when the messages are received late. Then, an artifact of the definition of non-blockingness results in an implicit assumption, namely, that messages *cannot keep arriving early*, therefore eliminating the blocking behaviors in the  $(k + 1)$ -bounded case. This cannot happen in the  $k$ -bounded case, where messages *never* arrive late enough.

**Proposition 4.4** For any  $k \in \mathbb{N}$ ,  $G$ ,  $C_1$  and  $C_2$ ,  $L_{\max}(G/C_1 \wedge_k C_2) \subseteq L_{\max}(G/C_1 \wedge_{k+1} C_2) \subseteq L_{\max}(G/C_1 \wedge_{\infty} C_2)$ .

**Proof** For ease of notation, let  $H_k = (G/C_1 \wedge_k C_2)$ ,  $H_{k+1} = (G/C_1 \wedge_{k+1} C_2)$  and  $H_{\infty} = (G/C_1 \wedge_{\infty} C_2)$ , and let  $\Delta_k$ ,  $\Delta_{k+1}$  and  $\Delta_{\infty}$  be the transition functions of  $H_k$ ,  $H_{k+1}$  and  $H_{\infty}$ , respectively.

Let  $\rho \in L_{\max}(H_k)$ . This means that  $\rho$  is a maximal string generated by  $H_k$ .

Consider the case where  $\rho$  is infinite. By Lemma 4.2, every transition of  $H_k$  is also a transition of  $H_{k+1}$  and  $H_{\infty}$ , therefore,  $\rho$  is also generated by  $H_{k+1}$  and  $H_{\infty}$ .

Consider the case where  $\rho$  is finite. Since  $\rho$  is maximal in  $H_k$ , there exists some  $s \in \Delta_k(\rho)$ , such that  $s$  is a deadlock in  $H_k$ , that is, for every  $a$ ,  $\Delta_k(s, a) = \emptyset$ . By Lemma 4.2,  $s$  is a reachable state of both  $H_{k+1}$  and  $H_{\infty}$ , that is,  $\rho$  is also generated by  $H_{k+1}$  and  $H_{\infty}$ . By Lemma 4.3,  $s$  is a deadlock in  $H_{k+1}$  and  $H_{\infty}$ , that is,  $\rho$  is maximal also in  $H_{k+1}$  and  $H_{\infty}$ . ■

**Corollary 4.5** For all  $k \in \mathbb{N}$ ,  $DCC_{k+1} \subseteq DCC_k$  and  $DCUC \subseteq DCC_k$ .

**Proof** Follows from Proposition 4.4 and the fact that if  $L' \subseteq L$  and  $L \models \phi$ , then  $L' \models \phi$ . ■

We next observe that every decentralized control problem that can be solved without any communication, can also be solved with unbounded-delay communication. This is easy to see, since any controllers that work without exchanging any information, will also work on any network, simply by ignoring all messages they receive.

**Proposition 4.6**  $DC \subseteq DCUC$ .

Putting together all the above results, we get the following inclusions:

$$CC = DCC_0 \supseteq DCC_1 \supseteq DCC_2 \supseteq \dots \supseteq DCUC \supseteq DC.$$

Before proceeding to show that the above inclusions are strict, it is worth noting that, because of Lemma 4.2, the above inclusions will continue to hold in any other specification framework, say  $\models'$ , provided  $\models'$  is *monotonic* with respect to the transition graphs of systems, that is, if  $H \models' \phi$  and the transition graph of  $G$  is a subgraph of the transition graph of  $H$ , then  $G \models' \phi$ .

**Proposition 4.7** For all  $k \in \mathbb{N}$ ,  $DCC_k - DCC_{k+1} \neq \emptyset$ .

**Proof** We will use the plant depicted in Figure 5. Assume that  $u, u_1, \dots, u_k$  are uncontrollable and unobservable events, while  $a, c$  are controllable by controller  $C_1$  and  $b$  is observable by controller  $C_2$ . The specification  $\phi$  is  $\{u \rightsquigarrow d, b \rightsquigarrow c\}$ . In other words, we want to keep  $a$  initially enabled, in case  $u$  occurs, but disable it if  $b$  occurs. We can build correct controllers in a  $k$ -bounded-delay network. Controller  $C_2$  will do nothing, except transmit  $\hat{b}$  to  $C_1$ , if  $b$  occurs. Controller  $C_1$  will initially enable both  $a$  and  $c$ . If it receives  $\hat{b}$ , it will disable  $a$ . It can be seen that these controllers satisfy  $\phi$  in a  $k$ -bounded-delay network, because  $\hat{b}$  will be received by  $C_1$  at the latest right after  $u_k$  occurs, and before the “illegal”  $a$  can occur.

However, in a network where delays can be more than  $k$ , the illegal  $a$  may happen before  $C_1$  has received  $\hat{b}$ . If  $C_1$  decides to disable  $a$  right from the start (i.e., without observing anything), then  $u \rightsquigarrow d$  will be violated if the plant performs  $u$ . ■

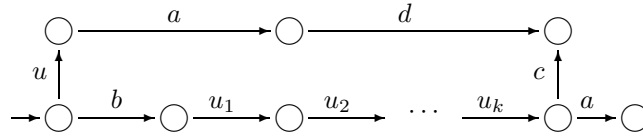


Figure 5: Solvable with a  $k$ -bounded-delay network, but not with a  $(k + 1)$ -bounded-delay network.

The example in the proof of Proposition 4.7, along with the fact that  $DCUC \subseteq DCC_{k+1}$ , can be used to prove the following.

**Corollary 4.8** For all  $k \in \mathbb{N}$ ,  $DCC_k - DCUC \neq \emptyset$ .

**Proof** We use the same example as in the proof of Proposition 4.7 and the fact that  $DCUC \subseteq DCC_{k+1}$ . ■

**Proposition 4.9**  $DCUC - DC \neq \emptyset$ .

**Proof** We will use the plant depicted in Figure 6. Assume that events  $a$  and  $b$  are observable by  $C_2$  and events  $c$  and  $d$  are controllable and observable by  $C_1$ . Let the specification be  $\phi = \{a \rightsquigarrow c, b \rightsquigarrow d\}$ . That is, we want to disable  $d$  if  $a$  occurs and  $c$  if  $b$  occurs. If the controllers can communicate, then  $C_1$  can initially disable both  $c$  and  $d$  and wait, until it receives  $\hat{a}$  or  $\hat{b}$ . If this ever happens, then  $C_1$  knows that either  $a$  or  $b$  occurred, and can enable the corresponding response.<sup>6</sup>

In a setting without communication, however,  $C_1$  cannot possibly know which of  $c, d$  to disable. It cannot disable both, since no response will ever be given, then. It cannot enable both either, since this may result in an incorrect response. ■

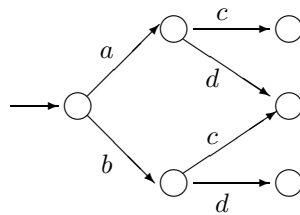


Figure 6: Solvable with an unbounded-delay network, but not without any network.

<sup>6</sup>Note that delays can be arbitrary but they are finite, so there can be no infinite behavior with only  $t$  events.

## 5 Undecidability results

In this section, we present two undecidability results. First, undecidability of the decentralized control problem with unbounded-delay communication. Second, undecidability of the decentralized control problem without communication.

In the proofs, we are going to use the notion of *joint observability*, introduced and shown to be undecidable in [14]. For completeness, we recall these results.

**Definition 5.1 (joint observability)** *Given regular languages  $K \subseteq L \subseteq \Sigma^*$  over some alphabet  $\Sigma$ , and  $\Sigma_1, \Sigma_2 \subseteq \Sigma$ ,  $K$  is said to be jointly observable with respect to  $L$  and  $\Sigma_1, \Sigma_2$ , if there exist no strings  $\rho \in K, \rho' \in L - K$ , such that for  $i = 1, 2$ ,  $P_{\Sigma_i}(\rho) = P_{\Sigma_i}(\rho')$ .*

**Theorem 5.2** *Checking joint observability is undecidable [14].*

### 5.1 Undecidability of the decentralized control problem with unbounded-delay communication

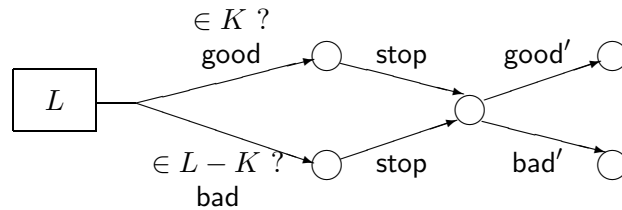


Figure 7: Plant for the proof of Proposition 5.3.

**Proposition 5.3** *The decentralized control problem with unbounded-delay communication is undecidable.*

**Sketch of proof** Consider regular languages  $K \subseteq L \subseteq \Sigma^*$  and  $\Sigma_1, \Sigma_2 \subseteq \Sigma$ . We will define a plant  $G$  over a new alphabet  $\Gamma$ , alphabets  $\Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C} \subseteq \Gamma$ , and a property  $\phi$  over  $\Gamma$ , such that  $K$  is jointly observable with respect to  $L, \Sigma_1, \Sigma_2$  iff there exist controllers  $C_i$  over  $\Sigma_{iO} \cup \widehat{\Sigma_{jO}}$  with outputs in  $\Sigma_{iC}$ ,  $i, j = 1, 2$ ,  $i \neq j$ , such that  $L_{\max}(G/C_1 \wedge_{\infty} C_2) \models \phi$ . Since checking joint observability is undecidable, checking the existence of such controllers is also undecidable.

Let  $\Gamma = \Sigma \cup \{\text{stop}, \text{good}, \text{good}', \text{bad}, \text{bad}'\}$ , where  $\text{stop}, \text{good}, \text{good}', \text{bad}, \text{bad}'$  are new events. Let  $\Sigma_{1O} = \Sigma_1 \cup \{\text{stop}\}$  and  $\Sigma_{2O} = \Sigma_2 \cup \{\text{stop}\}$ . Let  $\Sigma_{1C} = \{\text{good}', \text{bad}'\}$  and  $\Sigma_{2C} = \emptyset$ .  $G$  is the automaton shown in Figure 7.  $G$  initially generates strings in  $L$ . At some point,  $G$  may decide to stop generating strings in  $L$ . Suppose that at that point, the generated string is  $\rho \in L$ . Then, if  $\rho \in K$ ,  $G$  executes  $\text{good}$ , otherwise, it executes  $\text{bad}$ .<sup>7</sup> After that,  $G$  executes  $\text{stop}$  and waits for controller 1 to enable either  $\text{good}'$  or  $\text{bad}'$ .

<sup>7</sup>We can always build the part  $G$  that generates strings in  $L$  in such a way, such that at every state  $s$ , we know, first, whether the string generated up to  $s$  is in  $L$  or in  $\Sigma^* - L$  (in the latter case we allow neither  $\text{good}$  nor  $\text{bad}$  to occur), and, second, whether this string is in  $K$  or in  $L - K$  (in the first case we allow  $\text{good}$  and in the second case we allow  $\text{bad}$ ).

The specification  $\phi$  is defined to be  $\{\text{good} \rightsquigarrow \text{good}', \text{bad} \rightsquigarrow \text{bad}'\}$ . That is, if  $G$  generates a string in  $K$ , then we want  $\text{good}'$  to be enabled, and if  $G$  generates a string in  $L - K$ , then we want  $\text{bad}'$  to be enabled. In other words, we are “asking” the controllers to decide whether the initially generated string was in  $K$  or not.

We claim that  $(G, \Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C}, \phi) \in \mathcal{DCUC}$  iff  $K$  is jointly observable with respect to  $L, \Sigma_1, \Sigma_2$ .

Suppose  $K$  is jointly observable with respect to  $L, \Sigma_1, \Sigma_2$ . Then, by Lemma 2.1 of [14], we know that there exists a function  $f : \Sigma_1^* \times \Sigma_2^* \rightarrow \{0, 1\}$ , such that for any  $\rho \in L$ ,  $f(P_{\Sigma_1}(\rho), P_{\Sigma_2}(\rho)) = 1$  iff  $\rho \in K$ . Controllers  $C_1, C_2$  can be constructed as follows.<sup>8</sup>  $C_2$  will do nothing, except transmit its observations to  $C_1$ .  $C_1$  will initially disable both  $\text{good}'$  and  $\text{bad}'$  and wait until it receives  $\widehat{\text{stop}}$ . At that point, and supposing that  $G$  generated  $\rho \in L$ ,  $C_1$  knows both  $P_{\Sigma_1}(\rho)$  (from its own observations) and  $P_{\Sigma_2}(\rho)$  (from what it received from  $C_2$ ). Then,  $C_1$  will compute  $f(P_{\Sigma_1}(\rho), P_{\Sigma_2}(\rho))$ . If the function gives 1,  $C_1$  will enable  $\text{good}'$ , otherwise, it will enable  $\text{bad}'$ .

To see that the above construction yields correct controllers, observe the following. First, every infinite behavior in  $(G/C_1 \wedge_{\infty} C_2)$  cannot contain neither  $\text{good}$ , nor  $\text{bad}$ , thus,  $\phi$  is trivially satisfied. Second, if  $\text{good}$  ever occurs, then  $\text{stop}$  will also occur (by maximality, and the fact that  $\text{stop}$  is uncontrollable), thus,  $\widehat{\text{stop}}$  will eventually be received by both controllers (even though this will happen after an unbounded number of steps). Then,  $\text{good}'$  will be enabled, and the specification will be satisfied. The situation is similar if  $\text{bad}$  occurs.

Now, suppose  $K$  is not jointly observable with respect to  $L, \Sigma_1, \Sigma_2$ , that is, there exist  $\rho \in K, \rho' \in L - K$ , such that  $P_{\Sigma_i}(\rho) = P_{\Sigma_i}(\rho') = \sigma_i$ , for  $i = 1, 2$ . Assume that controllers satisfying  $\phi$  exist. Suppose that  $G$  initially performs  $\rho \text{good stop}$ : this can happen, since all events in  $\Sigma \cup \{\text{good}, \text{bad}, \text{stop}\}$  are uncontrollable. Also suppose that all events sent by  $C_2$  to  $C_1$  are received by  $C_1$  after  $C_1$  observes  $\text{stop}$ : this can happen, since the network delay can be greater than the length of  $\rho$ . Since  $\phi$  must be satisfied, some time after  $\text{stop}$ ,  $\text{good}'$  must be enabled. Moreover,  $\text{bad}'$  cannot be enabled at all after  $\text{stop}$ , otherwise the specification can be violated. Suppose  $C_1$  enables  $\text{good}'$  for the first time after  $\text{stop}$  when it observes some string  $\pi = \sigma_1 \text{stop} \tau$ , where  $\tau$  is a prefix of  $\widehat{\sigma_2 \text{stop}}$ , that is,  $\widehat{\sigma_2 \text{stop}} = \tau \tau'$ .

Then,  $\rho \text{good stop} \tau \text{good}' \tau'$  is a maximal behavior of  $(G/C_1 \wedge_{\infty} C_2)$ . We claim that the string  $\rho' \text{bad stop} \tau \text{good}' \tau'$  is also a maximal behavior of  $(G/C_1 \wedge_{\infty} C_2)$ . To see this, observe that, having observed  $\pi$ ,  $C_1$  is in some state  $s_1$  and  $\Lambda_{1C}(s_1) = \{\text{good}'\}$ . Suppose  $G$  performs  $\rho' \text{bad stop}$ , and all events sent by  $C_2$  to  $C_1$  are received by  $C_1$  after it observes  $\text{stop}$ . Then,  $\rho'$  produces the same observations as  $\rho$ , the sequence observed by  $C_1$  is again  $\sigma_1 \text{stop} \tau = \pi$ . Since  $C_1$  is receptive and deterministic, it reaches state  $s_1$  and enables  $\text{good}'$ . Thus,  $\rho' \text{bad stop} \tau \text{good}' \tau'$  is also a maximal behavior of  $(G/C_1 \wedge_{\infty} C_2)$ . But this violates the property  $\text{bad} \rightsquigarrow \text{bad}'$ . Thus, correct controllers cannot exist. ■

## 5.2 Undecidability of the decentralized control problem without communication

**Proposition 5.4** *The decentralized control problem without communication is undecidable.*

<sup>8</sup>Note that we construct *infinite-state* controllers.

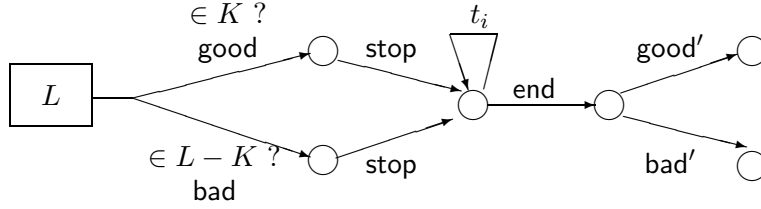


Figure 8: Plant for the proof of Proposition 5.4.

**Sketch of proof** The proof is similar to the one of Proposition 5.3, except that a slightly different plant is used, shown in Figure 8. This plant offers the possibility to controller  $C_2$  to transmit its observations to controller  $C_1$ , by enabling and disabling some controllable events accordingly. That is, communication between the two controllers is “simulated” by the plant itself.

We use the notation of the proof of Proposition 5.3. Assuming  $\Sigma_2 = \{a_1, \dots, a_n\}$ , let  $\Sigma_t = \{t_1, \dots, t_n, \text{end}\}$  be a set of new events, and define  $\Gamma' = \Sigma \cup \{\text{stop}, \text{good}, \text{good}', \text{bad}, \text{bad}'\} \cup \Sigma_t$ . Let  $\Sigma_{1O} = \Sigma_1 \cup \{\text{stop}\} \cup \Sigma_t$  and  $\Sigma_{2O} = \Sigma_2 \cup \{\text{stop}\} \cup \Sigma_t$ . Finally, let  $\Sigma_{1C} = \{\text{good}', \text{bad}'\}$  and  $\Sigma_{2C} = \Sigma_t$ .

The new plant  $G'$  is over  $\Gamma'$ . The initial behavior of  $G'$  (up to  $\text{stop}$ ) is similar to the initial behavior of  $G$ :  $G'$  generates strings in  $L$ , until it may decide to stop and execute  $\text{good}$  (if upon stopping it has generated a string in  $K$ ) or  $\text{bad}$  (otherwise), followed by  $\text{stop}$ . At that point,  $G'$  waits for  $C_2$  to transmit its observation to  $C_1$ .  $C_2$  can do this by enabling a sequence  $t_{i_1} \cdots t_{i_l} \text{end}$ , which corresponds to the message “I have observed  $a_{i_1} \cdots a_{i_l}$ ”.<sup>9</sup> Finally,  $C_1$  must enable either  $\text{good}'$  or  $\text{bad}'$ . The property  $\phi$  is defined to be  $\{\text{good} \rightsquigarrow \text{good}', \text{bad} \rightsquigarrow \text{bad}'\}$ .

We claim that  $(G', \Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C}, \phi) \in \mathcal{DCUC}$  iff  $K$  is jointly observable with respect to  $L, \Sigma_1, \Sigma_2$ . The first direction, where we assume  $K$  jointly observable and construct the controllers, is almost identical to the previous proof, with two differences: first,  $C_2$  explicitly transmits what it observed to  $C_1$ , using the sequence of  $t_i$ 's followed by  $\text{end}$ ; second,  $C_1$  waits for  $\widehat{\text{end}}$  instead of  $\widehat{\text{stop}}$ , before it decides.

In the other direction, suppose  $K$  is not jointly observable with respect to  $L, \Sigma_1, \Sigma_2$ . Then, there exist  $\rho \in K, \rho' \in L - K$ , such that  $P_{\Sigma_i}(\rho) = P_{\Sigma_i}(\rho') = \sigma_i$ , for  $i = 1, 2$ . Assume that controllers satisfying  $\phi$  exist. Suppose  $G'$  performs  $\rho \text{good stop}$ . At that point, for  $i = 1, 2$ , controller  $C_i$  has observed  $\sigma_i$  and is in state  $s_i$ . Then,  $C_2$  transmits some sequence  $\tau \text{end} \in \Sigma_t^*$ :  $C_2$  cannot transmit an infinite sequence, since then  $\text{good}'$  will never take place and  $\phi$  will not be satisfied.<sup>10</sup>  $C_1$  enables  $\text{good}'$  for the first time after  $\text{stop}$ , once it observes some prefix  $\pi$  of  $\tau \text{end}$ . Let  $C_1$  be in state  $s'_1$  when this happens.

<sup>9</sup>Enabling a sequence of controllable and observable events  $c_1 \cdots c_m$  can be easily done by starting at a state  $s_1$  with all events disabled except  $c_1$ , then, when  $c_1$  is observed, moving to state  $s_2$  where all events are disabled except  $c_2$ , and so on.

<sup>10</sup>It may be the case that  $C_2$  enables both  $\text{end}$  and some  $t_i$  at the same time. Then, it is possible that the plant decides to execute  $t_i$  instead of  $\text{end}$ :  $C_2$  then observes  $t_i$  and decides what to do next. Notice, however, that  $C_2$  cannot keep enabling  $t_i$  events forever, because then an infinite transmission is possible and  $\text{good}'$  never occurs.

Still, we can see that the sequence  $\tau$  is not always unique. This is not a problem, because for each such sequence  $\tau$  transmitted to  $C_1$ ,  $C_1$  must make the correct decision. Therefore, we can assume that  $\tau$  is one of the possible sequences to be transmitted to  $C_1$ .

Now, suppose  $G'$  performs  $\rho'$  bad stop. Since  $\rho$  and  $\rho'$  yield the same observations to both  $C_1$  and  $C_2$ , and the controllers are deterministic and receptive,  $C_2$  will transmit exactly the same sequence to  $C_1$  as when  $\rho$  good stop occurred, and  $C_1$  will behave in exactly the same manner, thus enabling good' incorrectly. ■

## 6 Towards decidability of decentralized control with bounded-delay communication

We believe that solving the decentralized control problem with bounded-delay communication is decidable. Towards such a result, we provide a decidability proof of a simpler problem of decentralized observation with bounded-delay communication. Observation lies at the heart of any control problem, thus, showing decidability of the former may help in understanding how to solve the latter.

We first start with some definitions. Consider an alphabet  $\Sigma$  and  $\Sigma_i \subseteq \Sigma$ , for  $i = 1, 2$ . Given  $a \in \Sigma_i$ ,  $\hat{a}^j$  is another event, called the *indexed* message event, which models the message sent by observer  $i$  to observer  $j$ , for  $i, j = 1, 2$ ,  $i \neq j$ . Let  $\Gamma_j = \{\hat{a}^j \mid a \in \Sigma_i, \text{ for } i, j = 1, 2, i \neq j\}$ . For example, if  $\Sigma_1 = \{a, b\}$  and  $\Sigma_2 = \{a, c\}$ , then  $\Gamma_1 = \{\hat{a}^2, \hat{b}^2\}$  and  $\Gamma_2 = \{\hat{a}^1, \hat{c}^1\}$ . Note that  $\Gamma_1$  and  $\Gamma_2$  are disjoint.

Given a regular language  $L$  over  $\Sigma$ ,  $\Sigma_1, \Sigma_2 \subseteq \Sigma$ , and  $k \in \mathbb{N}$ , we will construct the language  $L_{\Sigma_1, \Sigma_2}^k$ , which models the observation of  $L$  by two observers, where observer  $i$  observes all events in  $\Sigma_i$  immediately when they occur, and receives all events in  $\Sigma_j$  as indexed messages in  $\Gamma_i$ , within a delay of at most  $k$ .

Let  $A = (S, q_0, \Sigma, \delta, F)$  be a deterministic automaton *accepting*  $L$ , where  $F \subseteq S$  is the set of accepting states of  $A$  (we use the standard notion of acceptance of a finite string  $\rho$ , where  $\rho$  must lead  $A$  to an accepting state). Let  $\mathcal{Q}_i$  be the set of queues over  $\Gamma_i$ , for  $i = 1, 2$ . We define a new automaton,  $A_{\Sigma_1, \Sigma_2}^k = (S', q'_0, \Sigma', \Delta, F')$ , where  $S' = S \times \mathcal{Q}_1 \times \mathcal{Q}_2$ ,  $q'_0 = (q_0, [], [])$ ,  $\Sigma' = \Sigma \cup \Gamma_1 \cup \Gamma_2 \cup \{t\}$ ,  $F' = F \times \{[]\} \times \{[]\}$ , and  $\Delta$  contains the following types of transitions:

1.  $(s, Q_1, Q_2) \xrightarrow{\hat{a}^j} (s, Q'_1, Q'_2)$ : if  $Q_i$  is ready with  $\text{head}(Q_i) = (\hat{a}^j, 0)$ , in which case,  $Q'_i = \text{pop}(Q_i)$ ,  $Q'_j = Q_j$ , where  $i, j = 1, 2$ ,  $j \neq i$ ,
2. if no queue is ready,
  - (a)  $(s, Q_1, Q_2) \xrightarrow{b} (s', Q'_1, Q'_2)$ : if  $b \in \Sigma$ ,  $s' = \delta(s, b)$  is defined and, for  $i = 1, 2$ , if  $b \in \Sigma_i$ , then  $Q'_j \in \text{push}_k(Q_j - 1, \hat{b}^j)$ , otherwise,  $Q'_j = Q_j - 1$ , where  $j = 1, 2$ ,  $j \neq i$ ,
  - (b)  $(s, Q_1, Q_2) \xrightarrow{t} (s, Q_1 - 1, Q_2 - 1)$ : if there is no  $b$  such that clause 2(a) is satisfied and some queue  $Q_i$ ,  $i = 1, 2$ , is non-empty.

$L_{\Sigma_1, \Sigma_2}^k$  is defined to be the language accepted by  $A_{\Sigma_1, \Sigma_2}^k$ , that is, the set of all finite strings  $\pi \in (\Sigma \cup \Gamma_1 \cup \Gamma_2 \cup \{t\})^*$ , such that  $A_{\Sigma_1, \Sigma_2}^k$  has a sequence of transitions labeled with  $\pi$  to some state  $(s, [], [])$ , where  $s \in F$ .

For example, let  $L = \{a b b a c b a c\}$  and  $\Sigma_1 = \{a\}$ ,  $\Sigma_2 = \{b\}$ ,  $k = 2$ . Then, possible strings of  $L_{\Sigma_1, \Sigma_2}^k$  are the following:

$$\pi_1 = a \hat{a}^2 b \hat{b}^1 b \hat{b}^1 a \hat{a}^2 c b \hat{b}^1 a \hat{a}^2 c, \quad \pi_2 = a b b \hat{a}^2 \hat{b}^1 a c \hat{b}^1 \hat{a}^2 b a \hat{b}^1 c \hat{a}^2.$$

On the other hand, the following strings do not belong in  $L_{\Sigma_1, \Sigma_2}^k$ :

$$\pi_3 = a \widehat{a}^2 \widehat{b}^1 b \cdots, \quad \pi_4 = a b b a \widehat{a}^2 \cdots.$$

$\pi_3$  is not valid since  $\widehat{b}^1$  appears before  $b$ .  $\pi_4$  is not valid since there are  $3 > 2$  events in  $\Sigma$  between the first  $a$  and the corresponding  $\widehat{a}^2$ .

Note that, although the set of all potential states of  $A_{\Sigma_1, \Sigma_2}^k$  is infinite, its set of reachable states is finite. This is because no queue of  $A_{\Sigma_1, \Sigma_2}^k$  can contain more than  $k + 1$  elements. Indeed, by part 2 of Lemma 3.3, for every element in a queue, its time-to-live field is at most  $k$ . Every time a new element is added in the queue, the age of all elements already in the queue is decremented by 1. Since any new element has age at most  $k$ , after  $k$  additions at the most, the queue will be ready and its head must be removed. Therefore,  $A_{\Sigma_1, \Sigma_2}^k$  is a finite-state automaton, and  $L_{\Sigma_1, \Sigma_2}^k$  is regular.

**Definition 6.1 (joint observability with bounded-delay communication)** *Given regular languages  $K, L$  over  $\Sigma$ ,  $K \subseteq L$ ,  $\Sigma_1, \Sigma_2 \subseteq \Sigma$  and  $k \in \mathbb{N}$ ,  $K$  is said to be jointly observable with bounded-delay  $k$  with respect to  $L$  and  $\Sigma_1, \Sigma_2$  if there exist no  $\pi, \pi' \in L_{\Gamma_1, \Gamma_2}^k$ , such that*

$$P_{\Sigma}(\pi) \in K \wedge P_{\Sigma}(\pi') \in L - K \wedge \bigwedge_{i=1,2} P_{\Sigma_i \cup \Gamma_i}(\pi) = P_{\Sigma_i \cup \Gamma_i}(\pi'). \quad (1)$$

The intuition is the following. Suppose the system generates some behavior  $\rho \in L$ . Through a  $k$ -bounded-delay network, this results in some behavior  $\pi \in L_{\Sigma_1, \Sigma_2}^k$ . Observer  $i$  observes  $P_{\Sigma_i \cup \Gamma_i}(\pi)$ . Then, the two observers get together and try to decide whether  $\rho \in K$  or  $\rho \notin K$ , that is, whether  $P_{\Sigma}(\pi) \in K$  or  $P_{\Sigma}(\pi) \in L - K$ . Clearly, if there exist  $\pi$  and  $\pi'$  satisfying condition (1), then the two observers cannot make a decision.

**Proposition 6.2** *Checking joint observability with bounded-delay  $k$  is decidable, for any  $k \in \mathbb{N}$ .*

**Sketch of proof** Let  $A_K$  be the automaton accepting  $K$  and  $A_{L-K}$  be the automaton accepting  $L - K$ . Both  $A_K$  and  $A_{L-K}$  are finite state, since  $K$  and  $L - K$  are regular.

Let  $A_1 = A_{\Sigma_1, \Sigma_2}^k \parallel_{\Sigma} A_K$ , that is,  $A_1$  is the product of  $A_{\Sigma_1, \Sigma_2}^k$  and  $A_K$ , where the two automata *synchronize* on all transitions labeled with an event in  $\Sigma$ , and *interleave* on all other transitions. Let  $A_2 = A_{\Sigma_1, \Sigma_2}^k \parallel_{\Sigma} A_{L-K}$ . That is,  $A_1$  (resp.,  $A_2$ ) is constructed in such a way, so that it accepts exactly the strings  $\pi \in L_{\Sigma_1, \Sigma_2}^k$ , such that  $P_{\Sigma}(\pi) \in K$  (resp., such that  $P_{\Sigma}(\pi) \in L - K$ ).

Then, the idea is to build a special product of  $A_1$  and  $A_2$ , call it  $B$ , such that every string  $\tau$  accepted by  $B$  can be “decomposed” into two strings  $\pi$  and  $\pi'$ , accepted by  $A_1$  and  $A_2$ , respectively, such that, for  $i = 1, 2$ ,  $P_{\Sigma_i \cup \Gamma_i}(\pi) = P_{\Sigma_i \cup \Gamma_i}(\pi')$ . And vice-versa: for every two such strings, there is a string  $\tau$  accepted by  $B$ . Then, checking condition (1) boils down to a reachability analysis of  $B$ , which is decidable, provided the set of reachable states of  $B$  is finite.

$B$  will be the product of  $A_1$  and  $A_2$ , where, first,  $A_1$  and  $A_2$  synchronize on all events in  $\Sigma_1 \cup \Gamma_1$ .  $B$  will also use two FIFO queues,  $Q_1$  and  $Q_2$ , containing events in  $\Sigma_2 \cup \Gamma_2$  (these queues need not contain time-to-live indices). Initially both queues will be empty. The rule is that, for  $i, j = 1, 2$ ,  $i \neq j$ , every time  $A_i$  executes an event  $a \in \Sigma_2 \cup \Gamma_2$ , one of the following conditions must hold:

1. either  $Q_i$  is empty, in which case  $a$  is appended at the end of  $Q_j$ ,
2. or  $\text{head}(Q_i) = a$ , in which case  $a$  is popped from  $Q_i$ .

All other events (i.e.,  $\{\mathbf{t}\} \cup \Sigma - (\Sigma_1 \cup \Sigma_2)$ ) occur in  $B$  in an interleaving fashion. A state of  $B$  is accepting iff both  $A_1$  and  $A_2$  are in an accepting state and both  $Q_1$  and  $Q_2$  are empty.

We claim that there is a string accepted by  $B$  iff  $K$  is not jointly observable with bounded-delay  $k$  with respect to  $L$  and  $\Sigma_1, \Sigma_2$ .

Assume  $K$  is not jointly observable, that is, there exist  $\pi, \pi' \in L_{\Sigma_1, \Sigma_2}^k$ , satisfying condition (1). Then,  $\pi$  is accepted by  $A_1$  and  $\pi'$  is accepted by  $A_2$ . We can construct an execution accepted by  $B$  as follows.  $A_1$  moves according to  $\pi$ , until it reaches a state where it has to perform  $a \in \Sigma_1 \cup \Gamma_1$ . At that point,  $A_1$  must synchronize with  $A_2$  on  $a$ , so  $A_2$  must “catch up” and be able to perform  $a$ , too. Then,  $A_2$  moves according to  $\pi'$ , until the synchronization on  $a$  can take place: we know that this is always possible, that is,  $A_2$  will reach  $a$  before it reaches any other  $b \in \Sigma_1 \cup \Gamma_1$ , because  $P_{\Sigma_1 \cup \Gamma_1}(\pi) = P_{\Sigma_1 \cup \Gamma_1}(\pi')$ . Therefore, a deadlock due to synchronization cannot occur. Moreover, since  $P_{\Sigma_2 \cup \Gamma_2}(\pi) = P_{\Sigma_2 \cup \Gamma_2}(\pi')$ , when  $A_1$  has finished executing  $\pi$  and  $A_2$  has finished executing  $\pi'$ , both queues will be empty. At that point,  $B$  has reached an accepting state.

In the other direction, assume  $\tau$  is a string accepted by  $B$ . Then,  $\tau$  must correspond to the “shuffling” of two strings,  $\pi$  accepted by  $A_1$ , and  $\pi'$  accepted by  $A_2$ , such that  $P_{\Sigma_1 \cup \Gamma_1}(\pi) = P_{\Sigma_1 \cup \Gamma_1}(\pi')$  (since the two automata synchronize on  $\Sigma_1 \cup \Gamma_1$ ). Moreover, since the queues are empty at the end of  $\tau$ ,  $P_{\Sigma_2 \cup \Gamma_2}(\pi) = P_{\Sigma_2 \cup \Gamma_2}(\pi')$ . Finally, by definition of  $A_1$  and  $A_2$ ,  $P_{\Sigma}(\pi) \in K$  and  $P_{\Sigma}(\pi') \in L - K$ . Thus,  $K$  is not jointly observable.

It remains to show that the set of reachable states of  $B$  is finite. This holds, provided both queues remain bounded in length. To show this, we use the following argument. Let  $a \in \Sigma_2 \cup \Gamma_2$ , and suppose, at some point during execution of  $B$ ,  $a$  is appended in  $Q_1$  by  $A_2$ . We will show that, unless  $B$  stops executing events in  $\Sigma_2 \cup \Gamma_2$ ,  $a$  will be popped from  $Q_1$  after at most  $k$  such events have occurred.

First, note that, at the point where  $a$  occurs,  $Q_2$  must be empty: otherwise,  $A_2$  would pop the head of  $Q_2$  instead of appending  $a$  to  $Q_1$  (rule 2 above). We distinguish two cases:

- $a \in \Sigma_2$ : There are two possibilities:
  - Either  $A_2$  executes  $\hat{a}^1$  after  $a$  (note that, if  $A_2$  keeps moving,  $\hat{a}^1$  is bound to occur after  $A_2$  executes at most  $k$  events in  $\Sigma$ ). Since  $\hat{a}^1 \in \Gamma_1$ ,  $A_1$  must synchronize with  $A_2$  on  $\hat{a}^1$ . In order for  $A_1$  to execute  $\hat{a}^1$ , it must perform  $a$  before, therefore, must pop  $a$  from  $Q_1$ . (Note that,  $A_1$  could not have performed  $a$  before  $A_2$  performed  $a$ , because then  $Q_2$  would not be empty.)
  - Or  $A_2$  stops moving before it executes  $\hat{a}^1$ . Then, nothing is appended in  $Q_1$  anymore.
- $a \in \Gamma_2$ , i.e.,  $a = \hat{b}^2$ , for some  $b \in \Sigma_1$ : This means that  $A_2$  has executed  $b$  before, and, since  $b \in \Sigma_1$ ,  $A_1$  and  $A_2$  synchronized on  $b$ . There are two possibilities:
  - Either  $A_1$  executes  $\hat{b}^2$  after  $b$  (if  $A_1$  keeps moving after  $b$ ,  $\hat{b}^2$  is bound to occur after  $A_1$  executes at most  $k$  events in  $\Sigma$ ). At that point,  $Q_1$  is not empty: it contains at least  $\hat{b}^2$ . If  $\hat{b}^2$  is at the head of the list,  $A_1$  will pop it. Otherwise,  $A_1$  cannot execute  $\hat{b}^2$ .

- Or  $A_1$  stops moving before it executes  $\hat{b}^2$ . Then,  $A_2$  cannot keep appending events in  $Q_1$ . Indeed, it cannot keep appending events in  $\Sigma_2$ , since after a bounded number of steps it must execute their corresponding messages in  $\Gamma_1$  and synchronize on them with  $A_1$ . It cannot keep appending events in  $\Gamma_2$  either, since it must execute events in  $\Sigma_1$  before that, and again synchronize on them with  $A_1$ .

Thus, we have shown that  $Q_1$  remains bounded. The argument is symmetric in case  $a$  is appended in  $Q_2$  by  $A_1$ , and proves boundedness of  $Q_2$ . ■

## 7 Conclusions

We have introduced a discrete-event modeling framework which allows to take into account delays in decentralized control with communication. We have shown that this framework results in a natural hierarchy of control problems, the “easiest” being the centralized control problems (equivalent to zero-delay decentralized control problems), with progressively harder the problems with bounded-delay communication, unbounded-delay communication, or no communication at all. We have shown that the latter two problems are undecidable. Finally, we have introduced a decentralized observation problem with bounded-delay communication, and shown that it is decidable. We would like to extend this decidability result to cover decentralized control with bounded-delay communication.

To our knowledge, decentralized control with delayed communication has not been studied before in the context of discrete-event systems. [1] studied a related centralized control problem, namely, the problem of synthesizing a single controller when there are delays in the *input/output* interaction between plant and controller (i.e., an event generated by the plant is not immediately observed by the controller, and similarly with controller outputs), and provided necessary and sufficient conditions for the existence of a controller, in the restricted case of plants generating a so-called *memoryless* language.

**Acknowledgments** I would like to thank Karen Rudie and Eugene Asarin.

## References

- [1] S. Balemi. Communication delays in connections of input/output discrete event systems. In *Proceedings of 31st IEEE Conference on Decision and Control*, pages 3374–3379, 1992.
- [2] G. Barrett and S. Lafortune. On the synthesis of communicating controllers with decentralized information structures for discrete-event systems. In *IEEE Conference on Decision and Control*, 1998.
- [3] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.
- [4] R. Cieslak, C. Desclaux, A.S. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Automatic Control*, 33:249–260, 1988.

- [5] S. Jiang and R. Kumar. Decentralized control of discrete event systems with specializations to local control and concurrent systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):653–660, October 2000.
- [6] R. Kumar and M.A. Shayman. Centralized and decentralized supervisory control of non-deterministic systems under partial observation. *SIAM Journal on Control and Optimization*, 35(2):363–383, 1997.
- [7] H. Lamouchi and J. Thistle. Effective control synthesis for DES under partial observations. In *IEEE Conference on Decision and Control*, 2000.
- [8] F. Lin and W.M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.
- [9] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, January 1989.
- [10] S.L. Ricker and K. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Transactions on Automatic Control*, 45(9), September 2000.
- [11] K. Rudie, S. Lafortune, and F. Lin. Minimal communication in a distributed discrete-event control system. In *American Control Conference*, 1999.
- [12] K. Rudie and J.C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40(7), 1995.
- [13] K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37, 1992.
- [14] S. Tripakis. Undecidable problems of decentralized observation and control. In *IEEE Conference on Decision and Control*, 2001.
- [15] Y. Willner and M. Heymann. Supervisory control of concurrent discrete-event systems. *International Journal of Control*, 54(5), 1991.
- [16] K.C. Wong and J.H. van Schuppen. Decentralized supervisory control of discrete-event systems with communication. In *WODES*, 1996.
- [17] T. Yoo and S. Lafortune. New results on decentralized supervisory control of discrete-event systems. In *IEEE Conference on Decision and Control*, 2000.

## A The problem with non-blockingness

We try to illustrate the problem with replacing the responsiveness requirement with non-blockingness, in the definitions of the decentralized problems.

The problem is the following. We know that if controllers meet a specification in a  $(k + 1)$ -bounded network, then they also meet the specification in a  $k$ -bounded network. However, there are controllers that are non-blocking in a  $(k + 1)$ -bounded network, but blocking in a  $k$ -bounded network. This might seem counter-intuitive, since the behaviors of the  $k$ -bounded network are contained in the behaviors of the  $(k + 1)$ -bounded network (i.e., the latter is less

deterministic). But it is true and has to do with the linear (versus branching) nature of the definition of non-blockingness.

Here is an example.

Consider a plant  $G$  generating the regular language

$$(abc)^* \cdot d.$$

Suppose  $\{b, c\}$  are observable to controller 1, and  $\{d\}$  is controllable by controller 1. Controller 2 will not play any role here, so we'll ignore it.

When you add a 2-bounded network, the behavior of the plant becomes:

$$(a\hat{a}bc + ab\hat{a}c + abc\hat{a})^* \cdot d.$$

That is, the reception of  $\hat{a}$  can be delayed either 0, or 1, or 2 steps.

Controller 1 repeatedly sees one of three strings, namely,  $\hat{a}bc$ ,  $b\hat{a}c$ , or  $bc\hat{a}$ . Suppose that (for some twisted reason) the controller disables  $d$  initially and until it sees  $bc\hat{a}$ . That is, as long as it sees either  $\hat{a}bc$  or  $b\hat{a}c$ , it goes back to its initial state. Clearly, this controller is blocking in a 1-bounded network, since in such a network,  $bc\hat{a}$  will never happen (the number of events between  $a$  and  $\hat{a}$  can be at most 1).

However, in the 2-bounded network,  $bc\hat{a}$  is *always possible*. That is, from each state of the system, it is always possible that in the future  $bc\hat{a}$  happens. Then, once the controller sees that, it will enable  $d$ . So it seems that the controller is non-blocking in the 2-bounded network.