Decentralized Control of Discrete Event Systems with Bounded or Unbounded Delay Communication[1]

Stavros Tripakis[2]

VERIMAG Technical Report TR-2004-26
November 2004

Abstract

We introduce problems of decentralized control with delayed communication, where delays are either 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,

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

where the containments are strict, $\mathcal{CC}$ is the set of control problems solvable with a single controller (centralized case) and $\mathcal{DCC}_k$ (resp. $\mathcal{DCUC}$, $\mathcal{DC}$) is the set of problems solvable with two controllers in a k-bounded-delay network (resp. in an unbounded-delay network, without communication). The hierarchy is a result of the following property: controllers which "work" in a given network will also work in a less nondeterministic network. This property does not hold when non-blockingness is introduced. Checking the existence of controllers in the unbounded-delay case or in the case without communication are undecidable problems. However, a related decentralized observation problem with bounded-delay communication is decidable.

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 [9, 4, 22, 18, 17, 6, 5, 24, 8]) and the second class *decentralized control with communication* (e.g., see [23, 2, 16, 15, 14]). 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 [23, 2, 16, 15] has been based on the assumption that controllers can exchange information with *zero delay*, in other words, that the plant cannot perform any action between the transmission and reception of a message among controllers. This assumption, while simplifying 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, we assume that communication is *lossless*, that is, all messages are eventually delivered within a finite (possibly unbounded) delay. Second, we assume that communication is *FIFO*, that is, if message a is sent before message b, then a will be delivered before b. Third, we fix the communication policy to the following simple policy: each controller transmits the events it observes in the exact order it observes them, and nothing else. Fourth,

[1] A preliminary version of this work appeared in [20] and a journal version appeared in [21]. This work has been partially supported by the European IST project "Next TTA" under project No IST-2001-32111 and by CNRS STIC projects *AS RTP 23 "Automates, modèles distribués et temporisés"* and *ACI SI "Control and Observation of Real-Time Open Systems"*.

[2] Verimag, Centre Equation, 2, avenue de Vignate, 38610 Gières, France. E-mail: `Stavros.Tripakis@imag.fr`. Tel: +33 4 56 52 03 69. Fax: +33 4 56 52 03 44.

we consider a simple model of specifications, in terms of *responsiveness* properties, of the form "event a is always followed by event b". Finally, we consider for simplicity the case of only two controllers (this is not an essential assumption). Our framework is described in Section 3. The hierarchy is presented in Section 4.

The results we obtain are as follows.

First, we show that our modeling framework results in the (infinite) hierarchy of control problems expressed by Formula (1) below (containments are strict).

$$\mathcal{CC} = \mathcal{DCC}_0 \supset \mathcal{DCC}_1 \supset \cdots \supset \mathcal{DCUC} \supset \mathcal{DC} \qquad (1)$$

$\mathcal{CC}$ 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. $\mathcal{CC} = \mathcal{DCC}_0$ 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. 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 $k = 0, 1, 2, \cdots$, and that there are problems which can be solved with unbounded-delay communication, but cannot be solved without any communication.

We follow the framework of supervisory control for discrete-event systems (e.g., see [4, 18]) except that we use responsiveness properties to express requirements. Usually, in DES, a *legal* regular language is given, and the objective is to find controllers such that the language generated by the closed-loop system is contained in the legal language. To avoid trivial solutions (e.g., the language of the closed-loop system being empty) an extra requirement is added, namely, *non-blockingness*, which informally states that it is always possible in the closed-loop system to reach an accepting (or *marked*) state. In Section 5, we argue that non-blockingness is meaningless in a setting with communication, since it fails to satisfy a natural property. Indeed, controllers may be non-blocking in a $(k+1)$-bounded delay network, but blocking in a k-bounded delay network. The reason is that non-blockingness cannot distinguish between "cooperative" non-determinism (of the plant) and "adversarial" non-determinism (induced by the network). This is explained in Section 5.

We also provide a set of undecidability and decidability results. Some versions of the decentralized control problem are known to be decidable [18, 17], while others have recently been shown to be undecidable [8, 19]. In particular, checking the existence of (and constructing, if they exist) *non-blocking* controllers [13, 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 [18], 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* [24] controlled system without communication, $L(\cdot)$ is the *unmarked* (i.e., prefix-closed) language of $(G/C_1 \wedge C_2)$ and $L_m(\cdot)$ is the *marked* (or *accepted*) language of $(G/C_1 \wedge C_2)$. In [8], it was shown that checking the existence of decentralized deadlock-free controllers in an ω-regular language setting is undecidable. In [19], 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.[1]

In this paper, we extend the results of [19] and 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 and (2) no communication. 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* notion of [19], to take into account the fact that the observers communicate to each other their observations, and these observations are delivered with bounded delay.

Related work: [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. [14] studied a related problem of decentralized diagnosis with communication. Their model of communication appears to be similar to our unbounded-delay model.

The problem of decentralized control has been also considered in different settings, as in the setting of *reactive modules* [11, 7, 10]. There, the communication policy

[1] It is also worth noting that the setting of [18, 17] is slightly more general than the one considered in [19], in the sense that [18, 17] allow controllers to have their own acceptance conditions (accepting states), whereas in [19] it is assumed that all states of the controllers are accepting.

is not fixed, however, the entire system executes *synchronously*, thus, the communication delay is zero.

2 Preliminaries

N will denote the set of natural numbers. Let Σ be a finite alphabet. Σ^* denotes the set of all finite strings over Σ, ϵ denotes the empty string, and $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. Σ^ω denotes the set of all infinite strings over Σ. Given two strings ρ and ρ', such that ρ is finite, $\rho\rho'$ or $\rho \cdot \rho'$ is the *concatenation* of ρ and ρ'. Given a (finite or infinite) string ρ, a *prefix* of ρ is a finite string π such that $\rho = \pi \cdot \tau$, for some τ. Given a set of strings L, the *prefix-closure* of L (i.e., the set of all prefixes of all strings in L) is denoted by $\mathsf{pref}(L)$. Let ρ be a (finite or infinite) string over Σ. Given $\Gamma \subseteq \Sigma$, we define the *projection* of ρ to Γ, denoted $P_\Gamma(\rho)$, as the string obtained from ρ by erasing all letters not in Γ. For example, if $\Sigma = \{a, b, c\}$ and $\Gamma = \{a, c\}$, then $P_\Gamma(a\,b\,b\,c\,b\,a\,c\,b) = a\,c\,a\,c$. For a set of (finite or infinite) strings L, $P_\Gamma(L) = \{P_\Gamma(\rho) \mid \rho \in L\}$. For $K \subseteq \Gamma^*$ and $\Sigma \supseteq \Gamma$ implicitly assumed, $P_\Gamma^{-1}(K)$ is the inverse projection of K, that is, the greatest subset L of Σ^* such that $P_\Gamma(L) = K$. The length of a string ρ is denoted $|\rho|$. For instance, $|\epsilon| = 0$ and $|ab| = 2$.

A *responsiveness property* over some alphabet Σ is a formula of the form $a \rightsquigarrow b$, where $a, b \in \Sigma$. Consider a (finite or infinite) string ρ over Σ, $\rho = c_0\,c_1\,c_2 \cdots$. We say that ρ *satisfies* $a \rightsquigarrow b$, denoted $\rho \models a \rightsquigarrow b$, if b occurs after every a in ρ, that is, for all i, if $c_i = a$, then there exists $j > i$, such that $c_j = b$. For example, $a\,c\,b\,a\,a\,b$ satisfies $a \rightsquigarrow b$, whereas $a\,c\,c$ does not. Notice that $a \rightsquigarrow b$ does *not* require that b occurs only if a has occurred before. Thus, $b \models a \rightsquigarrow b$. Also notice that *multiple* a's can be "covered" by *a single* b, thus, $a\,a\,b \models a \rightsquigarrow b$. 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 ϕ, denoted $L \models \phi$, if L satisfies every property in ϕ. Note that if $L' \subseteq L$ and $L \models \phi$, then $L' \models \phi$.

Responsiveness captures other properties, such as *invariance* properties, of the form "event a never occurs". This can be expressed by the responsiveness property $a \rightsquigarrow b$, where b is a new event that never occurs. Then, $a \rightsquigarrow b$ is satisfied iff a never occurs.

A *non-deterministic automaton* over an alphabet Σ 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 $\Delta : S \times \Sigma \to 2^S$ is the non-deterministic transition function (Δ 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 be denoted by δ). 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 $\rho = a_1 \cdots a_k \in \Sigma^*$, we define $\Delta(s, \rho)$ to be the set of all states $s' \in S$, for which there exists a sequence of states $s_0, s_1, \ldots, s_k \in S$, such that $s_0 = s$, $s_k = s'$ and $s_{i+1} \in \Delta(s_i, a_{i+1})$. We write $s \xrightarrow{\rho} s'$ if $s' \in \Delta(s, \rho)$. We also write $\Delta(\rho)$ instead of $\Delta(q_0, \rho)$.

A state s of H is *reachable* if there exists some $\rho \in \Sigma^*$ such that $s \in \Delta(\rho)$. If $\Delta(\rho)$ is non-empty, we say that ρ is *generated* by H. Given an infinite string π, we say that π is generated by H if every finite prefix ρ of π is generated by H. A string ρ generated by H is *maximal* if, either ρ is infinite, or ρ 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.

An automaton H as above can be equipped with a set of *marked* states $S_m \subseteq S$. Given such an automaton, its *unmarked* language is defined to be $L(H) = \{\rho \in \Sigma^* \mid \Delta(\rho) \neq \emptyset\}$, and its *marked* language is defined to be $L_m(H) = \{\rho \in \Sigma^* \mid \Delta(\rho) \cap S_m \neq \emptyset\}$.

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

3 Centralized and decentralized control problems

We now define four control problems, namely, centralized control, decentralized control without communication, decentralized control with unbounded-delay communication and decentralized control with bounded-delay communication. First, we have to define what is a plant, what are the controllers, what is communication and what is the effect of one or more controllers on the plant.

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

3.1 Centralized control (CC)

Let $\Sigma_O, \Sigma_C \subseteq \Sigma$. Σ_O models the set of events of the plant that are *observable* by the controller and Σ_C models the set of *controllable* events. The latter can be "disabled" by the controller. The centralized control architecture is depicted in Figure 1.

The controller $C = (S_C, q_{0C}, \Sigma_O, \delta_C, \Sigma_C, \Lambda_C)$ is a re-

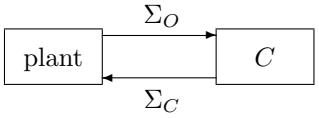

Figure 1: The centralized control architecture.

ceptive deterministic automaton over Σ_O with outputs in Σ_C. The intended meaning is that, when C is in state s, it disables all events in $\Sigma_C - \Lambda_C(s)$. Notice that we do not require C to be finite-state *a-priori*.

The *controlled system*, denoted (G/C), is defined to be the deterministic automaton (S, q_0, Σ, δ), where $S = S_G \times S_C$, $q_0 = (q_{0G}, q_{0C})$, and δ 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 1 (CC problem) *Given a finite-state deterministic automaton G over Σ, a specification ϕ over Σ, and $\Sigma_O, \Sigma_C \subseteq \Sigma$, does there exist a receptive deterministic automaton C over Σ_O with outputs in Σ_C, such that $L_{\max}(G/C) \models \phi$.*

3.2 Decentralized control without communication (DC)

Let $\Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C} \subseteq \Sigma$. Σ_{iO} (resp., Σ_{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 2.

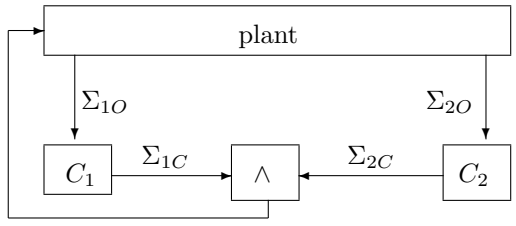

Figure 2: The (conjunctive) decentralized control architecture without communication.

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

The *conjunctively controlled system without communication*, denoted $(G/C_1 \wedge C_2)$, is defined to be the deterministic automaton (S, q_0, Σ, δ), where $S = S_G \times S_{1C} \times S_{2C}$, $q_0 = (q_{0G}, q_{1C}, q_{2C})$, and δ 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 2 (DC problem) *Given a finite-state deterministic automaton G over Σ, a specification ϕ over Σ, and $\Sigma_{1O}, \Sigma_{1C}, \Sigma_{2O}, \Sigma_{2C} \subseteq \Sigma$, do there exist receptive deterministic automata C_i over Σ_{iO} with outputs in Σ_{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 Γ is an ordered list of pairs of the form (a, i), where $a \in \Gamma$ and $i \in \mathsf{N}$. The index i is called the *time-to-live* field and models the remaining time steps until the message is delivered.[2] 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. It models a network where messages a and b have been sent, and a has been sent before b. Thus, by the FIFO property of the network that we assume, a will also be delivered before b. Moreover, the time-to-live field of a is 2, meaning that a will be delivered after 2 time steps. We shall see later how time is counted in our discrete-event model. The empty queue is denoted $\emptyset$.

The first element of a non-empty queue Q is its *head*, denoted $\mathsf{head}(Q)$, and the last element is its *tail*, denoted $\mathsf{tail}(Q)$. If $Q \neq \emptyset$ and $\mathsf{head}(Q) = (a, i)$, 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 $Q = \emptyset$, we let $Q - 1 = \emptyset$.

If $Q \neq \emptyset$ and $\mathsf{head}(Q) = (a, 0)$, then we say that Q is *ready*, and we define $\mathsf{pop}(Q)$ to be the new queue obtained by removing the head of Q. So, $\mathsf{pop}([(a, 0), (b, 3)]) = [(b, 3)]$. By convention, an empty queue is not ready. Let $\max(Q)$ denote the maximum i such that (a, i) 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 $\mathsf{push}(Q, a)$, which takes a queue

[2] In fact, we use the term "time-to-live" in a slightly different way than its standard meaning. In computer networks, "time-to-live" is a counter which is decremented at specific points (e.g., every time the message goes through a switch). When the counter reaches zero, the message is discarded, as it is considered too "old". In our case, when the time-to-live field reaches zero, the message is ready to be delivered and must be delivered before time can elapse.

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

$$\mathsf{push}([(a,1)],b) = \{\ [(a,1),(b,1)],\ [(a,1),(b,2)],\ [(a,1),(b,3)],[(a,1),(b,4)],...\}.$$

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

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

The pop operation models the network delivering a message. The operation $Q-1$ models time elapse and the corresponding "aging" of messages in the network. 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 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, push_k will be used. Notice that, by the FIFO property of the queue (which models the FIFO assumption we make on the network), a message cannot be delivered unless all previous messages in the queue are delivered. For example, $\mathsf{push}([(a,1)],b)$ models the fact that a is sent after b in an unbounded delay network. Since b will be delivered after one time unit, a cannot be delivered earlier. On the other hand, a can be delivered (much) later, after i time units, where i is unknown. In a bounded-delay network, a known bound exists for i.

The following properties follow easily from the definitions.

Lemma 3 *For any queue Q over some alphabet Γ, for any $a \in \Gamma$, and for any $k \in \mathsf{N}$, (1) $\mathsf{push}_k(Q,a) \subseteq \mathsf{push}_{k+1}(Q,a) \subseteq \mathsf{push}(Q,a)$, and (2) for any $Q' \in \mathsf{push}_k(Q,a)$, for any element (b,i) of Q', $i \leq k$.*

3.4 Decentralized control with unbounded-delay communication (DCUC)

Given an event a in some alphabet Σ, $\widehat{a}$ denotes another event, called the *message version* of a ($\widehat{a}$ will model the message sent by a controller that observes a). Given an alphabet Γ, we define $\widehat{\Gamma} = \{\widehat{a} \mid a \in \Gamma\}$. Given $\rho \in \Gamma^*$, $\rho = a_1 \cdots a_l$, $\widehat{\rho}$ denotes the string $\widehat{a_1} \cdots \widehat{a_l} \in \widehat{\Gamma}^*$.

Let $\Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C} \subseteq \Sigma$. Controller C_1 will observe its own observable events, Σ_{1O}, plus the message events it receives from C_2, $\widehat{\Sigma_{2O}}$. In order not to create confusion, we will assume that $\widehat{\Sigma_{1O}}$ and $\widehat{\Sigma_{2O}}$ are disjoint: this can be achieved by renaming if necessary. However, 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 (but possibly unbounded) delay.

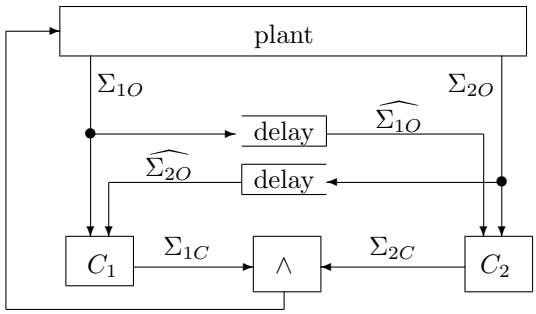

Figure 3: The (conjunctive) decentralized control architecture with communication.

Let $\mathcal{Q}_i$ be the set of all possible queues over $\widehat{\Sigma_{jO}}$, for $i,j = 1,2$, $i \neq j$. 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 Σ'_{iO} with outputs in Σ_{iC}. Let t be a new event, and define $\Pi = \Sigma \cup \widehat{\Sigma_{1O}} \cup \widehat{\Sigma_{2O}} \cup \{\mathsf{t}\}$.

The conjunctive decentralized control architecture with communication is depicted in Figure 3. 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, Π, Δ), where $S = S_G \times S_{1C} \times S_{2C} \times \mathcal{Q}_1 \times \mathcal{Q}_2$, $q_0 = (q, q_{1C}, q_{2C}, \emptyset, \emptyset)$, and Δ is defined as follows. Given states $s = (s_G, s_1, s_2, Q_1, Q_2)$ and $s' = (s'_G, s'_1, s'_2, Q'_1, Q'_2)$, Δ contains the following types of transitions:

1. (*delivery of a message*) $s \xrightarrow{\widehat{a}} s'$: if some queue Q_i is ready with $\mathsf{head}(Q_i) = (\widehat{a}, 0)$, $\widehat{a} \in \widehat{\Sigma_{jO}}$, in which case, $s'_G = s_G$, $Q'_i = \mathsf{pop}(Q_i)$, $Q'_j = Q_j$, $s'_i = \delta_{iC}(s_i, \widehat{a})$, $s'_j = s_j$, where $i,j = 1,2$, $j \neq i$,
2. if no queue is ready,

(a) (*plant transition and time progress*) $s \xrightarrow{b} s'$: 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 $s'_i = \delta_{iC}(s_i, b)$, $Q'_j \in \mathsf{push}(Q_j - 1, \widehat{b})$, otherwise, $s'_i = s_i$, $Q'_j = Q_j - 1$, where $j = 1, 2$, $j \neq i$,

(b) (*time progress*) $s \xrightarrow{\mathsf{t}} s'$: 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 moves according to b, and $\widehat{b}$ is sent to the other controller C_j. Clause 2(b) corresponds to time elapse, without any move of the network or the plant: this happens when the plant is blocked and there is at least one queue which is not ready and non-empty. In this case, time elapses, the queue eventually becomes ready and delivers the message. The special event t models one time step.

Definition 4 (DCUC problem) *Given a finite-state deterministic automaton G over Σ, a specification ϕ over Σ, 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 Σ_{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 (DCC)

Let Σ_{iO}, Σ_{iC}, C_i and Π be as in Section 3.4. In addition, we are given a natural constant $k \in \mathsf{N}$. The conjunctive decentralized control architecture with bounded-delay communication is the same as the one shown in Figure 3. 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 $\mathsf{push}(\cdot, \cdot)$ is replaced by $\mathsf{push}_k(\cdot, \cdot)$, in the definition of the transition function Δ.

Definition 5 (DCC problem) *Given $k \in \mathsf{N}$, a finite-state deterministic automaton G over Σ, a specification ϕ over Σ, 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 Σ_{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

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 $\Sigma_C = \Sigma_{1C} \cup \Sigma_{2C}$. $\mathcal{CC}$ will denote 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 $L_{\max}(G/C) \models \phi$. Classes $\mathcal{DCC}_k$, $\mathcal{DCUC}$ and $\mathcal{DC}$ are defined similarly, w.r.t. the DCC, DCUC and DC problems. We first observe that decentralized control with zero delay is equivalent to centralized control.

Proposition 6 $\mathcal{CC} = \mathcal{DCC}_0$.

Indeed, zero delay means that, each time the plant generates an observable event a, say, $a \in \Sigma_{1O}$, message $\widehat{a}$ is delivered to C_2 before the plant has time to generate another event. But this is equivalent to C_2 directly observing a. Thus, both controllers have same observation capabilities, equivalent to a single controller observing $\Sigma_{1O} \cup \Sigma_{2O}$.

Next, we show that every decentralized control problem that can be solved with communication of unbounded delay or delay at most $(k+1)$ can also be solved if the delay is at most k, using the same controllers. This is to be expected, since a network of delay at most k is more deterministic (i.e., has less behaviors) than a network of at most $k+1$ delay, or a network of unbounded delay.

However, as we shall see in Section 5, this natural property is violated in the framework of non-blockingness.

Lemma 7 *For any $k \in \mathsf{N}$, 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 in the induction step. ∎

Lemma 8 *For any $k \in \mathsf{N}$, 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 claim 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 7, 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 $a \in \widehat{\Sigma_{1O}} \cup \widehat{\Sigma_{2O}} \cup \{\mathtt{t}\}$. We can see that, in this case, the definition of Δ (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 $Q'_2 \in \mathsf{push}_{k+1}(Q_2 - 1, \widehat{a})$. Recall that s is reachable in H_k, that is, Q_2 has been obtained (potentially) using the $\mathsf{push}_k(\cdot,\cdot)$ operator, and not $\mathsf{push}_{k+1}(\cdot,\cdot)$. Therefore, by part 2 of Lemma 3, for every element (b, j) of Q_2, we have $j \leq k$, and $\mathsf{push}_k(Q_2 - 1, \widehat{a})$ is not empty. That is, for some $Q''_2 \in \mathsf{push}_k(Q_2 - 1, \widehat{a})$, 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. ∎

Proposition 9 *For any $k \in \mathsf{N}$, plant G, and controllers C_1, 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: Let $\rho \in L_{\max}(G/C_1 \wedge_k C_2)$. If ρ is infinite, then, by Lemma 7, ρ also belongs in $L_{\max}(G/C_1 \wedge_{k+1} C_2)$ and $L_{\max}(G/C_1 \wedge_\infty C_2)$. If ρ is finite, then it can lead $(G/C_1 \wedge_k C_2)$ to a deadlock state s. By Lemma 7, ρ can also lead $L_{\max}(G/C_1 \wedge_{k+1} C_2)$ and $L_{\max}(G/C_1 \wedge_\infty C_2)$ to s. By Lemma 8, s is a deadlock in $L_{\max}(G/C_1 \wedge_{k+1} C_2)$ and $L_{\max}(G/C_1 \wedge_\infty C_2)$, thus, ρ is a maximal string in the latter systems. ∎

Corollary 10 *For all $k \in \mathsf{N}$, $\mathcal{DCC}_{k+1} \subseteq \mathcal{DCC}_k$ and $\mathcal{DCUC} \subseteq \mathcal{DCC}_k$.*

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

Proposition 11 *$\mathcal{DC} \subseteq \mathcal{DCUC}$.*

Putting together all the above results, we get the inclusions of Formula (1). We now proceed to show that these inclusions are strict.

Proposition 12 *For all $k \in \mathsf{N}$, $\mathcal{DCC}_k - \mathcal{DCC}_{k+1} \neq \emptyset$.*

Proof: We will use the plant depicted in Figure 4. Assume that $u, u_1, ..., u_k$ are uncontrollable and unobservable events, while a, c are controllable by controller C_1 and b is observable by controller C_2. The specification ϕ 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 $\widehat{b}$ to C_1, if b occurs. Controller C_1 will initially enable both a and c. If it receives $\widehat{b}$, it will disable a. It can be seen that these controllers satisfy ϕ in a k-bounded-delay network, because $\widehat{b}$ 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 $\widehat{b}$. 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. ∎

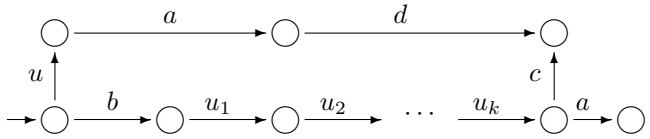

Figure 4: Solvable with a k-bounded-delay network, but not with a $(k+1)$-bounded-delay network.

Corollary 13 *For all $k \in \mathsf{N}$, $\mathcal{DCC}_k - \mathcal{DCUC} \neq \emptyset$.*

Proof: We use the same example as in the proof of Proposition 12 and the fact that $\mathcal{DCUC} \subseteq \mathcal{DCC}_{k+1}$. ∎

Proposition 14 *$\mathcal{DCUC} - \mathcal{DC} \neq \emptyset$.*

Proof: We will use the plant depicted in Figure 5. 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 $\widehat{a}$ or $\widehat{b}$. If this ever happens, then C_1 knows that either a or b occurred, and can enable the corresponding response.[3]

[3] Note that delays can be arbitrary but they are finite, so there can be no infinite behavior with only t events.

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. ∎

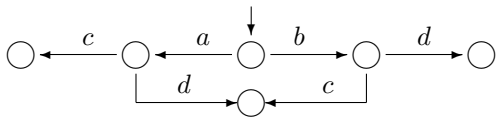

Figure 5: Solvable with an unbounded-delay network, but not without any network.

5 The case of non-blockingness

In supervisory control theory, specifications are often given by considering a *legal* language $E \subseteq \Sigma^*$, which must contain the closed-loop system language, and further requiring that the controllers be *non-blocking*. Informally, non-blockingness states that for any behavior ρ of the closed-loop system, it is *possible* to extend ρ to a behavior accepted by the plant.

In this section, we show that non-blockingness is not an appropriate requirement in the context of decentralized control with bounded-delay communication. Indeed, in such a setting, it is not true that controllers which are non-blocking in a $(k+1)$-bounded delay network are also non-blocking in a k-bounded delay network. We consider this problematic, since it does not meet our expectations. If we look at controllers as players in a "game" against the network, then, since a k-bounded delay network is a "weaker" player than a $(k+1)$-bounded delay network (the former has less choices than the latter), a strategy against the latter should also work against the former. To put it differently, controllers functioning properly in a network where delays are potentially (but not necessarily) large, should also function properly in a network where delays are guaranteed to be small. This property holds in the setting we have considered so far in this paper, as shown by Proposition 9.

Let us begin by defining an alternative decentralized control problem with bounded-delay communication, where requirements are not given as responsiveness properties, but as a legal language plus the requirement of non-blockingness.

The plant will be modeled as a deterministic automaton $G = (S_G, q_0, \Sigma, \delta, S_m)$, equipped with marked states $S_m \subseteq S_G$. The controllers will be modeled as in Section 3.5. The closed-loop system will be defined as in Section 3.5, with the addition that it will now have marked states, in particular, all states in $S_m \times S_{1C} \times S_{2C}$, where S_{iC} is the set of states of controller C_i, for $i = 1, 2$ (notice that controllers do not have marked states). Let $(G/C_1 \wedge_k C_2)$ denote the closed-loop system in a network with bounded delay k. Recall that $L(G/C_1 \wedge_k C_2)$ and $L_m(G/C_1 \wedge_k C_2)$ are, respectively, the unmarked and marked languages of $(G/C_1 \wedge_k C_2)$.

Definition 15 (DCCNB problem) *Given $k \in \mathsf{N}$, a finite-state deterministic automaton G over Σ, equipped with a set of marked states, a regular language $E \subseteq \Sigma^*$, 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 Σ_{iC}, for $i, j = 1, 2$, $j \neq i$, such that*

1. *$P_\Sigma(L_m(G/C_1 \wedge_k C_2)) \subseteq E$ (legal language requirement), and*
2. *$L(G/C_1 \wedge_k C_2) = \mathsf{pref}(L_m(G/C_1 \wedge_k C_2))$ (non-blockingness requirement).*

Theorem 16 *There exists a plant G and controllers C_1, C_2 such that $(G/C_1 \wedge_1 C_2)$ is non-blocking, whereas $(G/C_1 \wedge_0 C_2)$ is blocking.*

Proof: We will use the plant and the controllers shown in Figure 6. Let $\Sigma = \{a, b, c\}$. The marked state of the plant G is the state drawn with a double circle. Thus, G generates the regular language $(a\,b)^*\,c$. Let $\Sigma_{1O} = \{b, c\}$, $\Sigma_{1C} = \{c\}$, $\Sigma_{2O} = \{a\}$, $\Sigma_{2C} = \emptyset$. Controller C_2 plays no other role except transmitting its observations to controller C_1. Controller C_1 disables the controllable event c in all its states, except state 6. Intuitively, C_1 enables c only when it receives message $\widehat{a}$ with a unit delay, that is, when it observes $b\,\widehat{a}$. As long as $\widehat{a}$ is received with zero delay, that is, as long as C_1 observes $\widehat{a}\,b$, c remains disabled.

$(G/C_1 \wedge_1 C_2)$ is depicted in Figure 7. At each state, we show the local state of the plant, the local state of controller C_1 and the contents of the non-empty queues. The local state of controller C_2 is always the same, thus it is omitted. We do not explicitly identify the two queues: their contents suffice to identify them. For example, at state $(0, 3, [(\hat{a}, 1)])$, G is at state 0, C_1 is at state 3, and there is a message $\widehat{a}$ in the receiving queue of C_1 with time-to-live field 1. The marked state of the closed-loop system automaton is state $(2, 7)$.

It can be verified that $(G/C_1 \wedge_1 C_2)$ is non-blocking. Indeed, from every state of the automaton, the marked state $(2, 7)$ can be reached (this is a sufficient condition for non-blockingness).

$(G/C_1 \wedge_0 C_2)$ is obtained from $(G/C_1 \wedge_1 C_2)$ by removing all states (and corresponding transitions) where

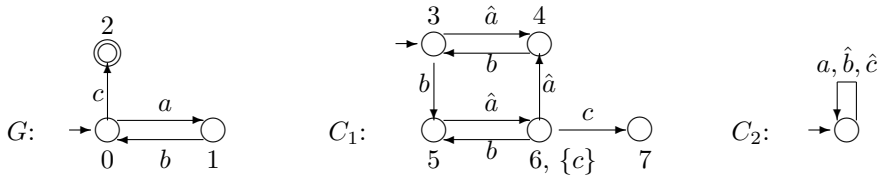

Figure 6: A plant G and two controllers C_1 and C_2.

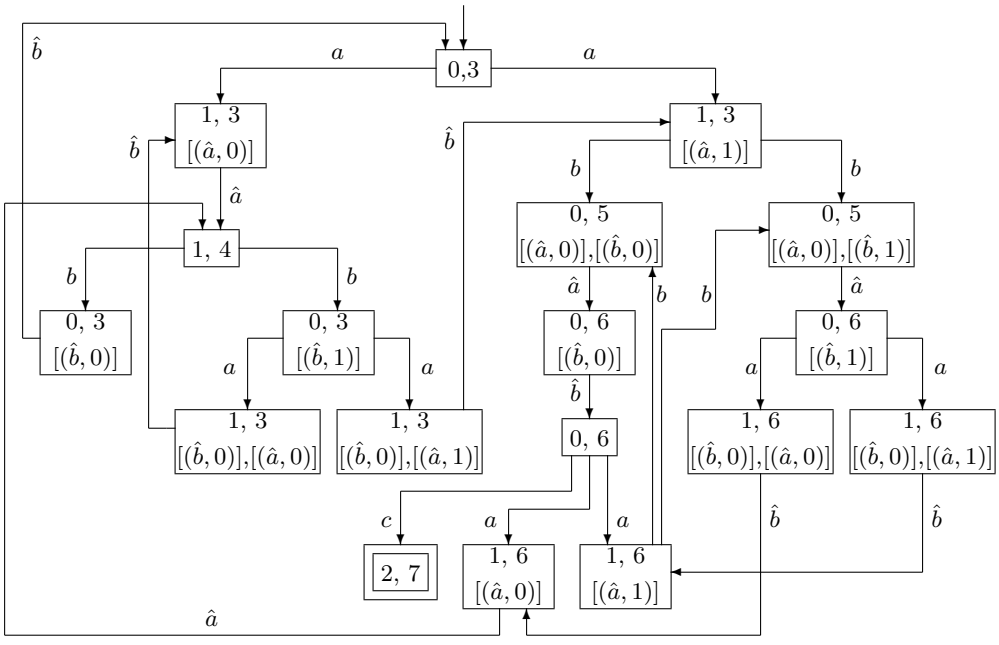

Figure 7: The closed-loop system $(G/C_1 \wedge_1 C_2)$ (where G, C_1, C_2 are as in Figure 6).

some element in some queue has a non-zero time-to-live field. Doing so, we obtain a reachable state space which is a subset of the one for $(G/C_1 \wedge_1 C_2)$. In particular, $(G/C_1 \wedge_0 C_2)$ contains a single cycle, namely, $(0,3) \xrightarrow{a} (1,3,[(\hat{a},0)]) \xrightarrow{\hat{a}} (1,4) \xrightarrow{b} (0,3,[(\hat{b},0)]) \xrightarrow{\hat{a}} (0,3)$. Thus, $L(G/C_1 \wedge_0 C_2) = \mathsf{pref}((a\,\hat{a}\,b\,\hat{b})^*)$. On the other hand, $L_m(G/C_1 \wedge_0 C_2) = \emptyset$, since the marked state is unreachable in $(G/C_1 \wedge_0 C_2)$. Thus, $(G/C_1 \wedge_0 C_2)$ is blocking. ■

6 Undecidability results

In this section, we show that checking existence of controllers in both cases of unbounded-delay communication and no communication are undecidable problems. The proofs are by reduction of an undecidable decentralized observation problem, namely, checking *joint observability* [19]. We first recall the definition of joint observability and state its properties of interest, without proof.

Definition 17 (joint observability) *Given regular languages $K \subseteq L \subseteq \Sigma^*$ over some finite alphabet Σ, and $\Sigma_1, \Sigma_2 \subseteq \Sigma$, K is said to be jointly observable with respect to L and Σ_1, Σ_2, if there exists a total function $f : \Sigma_1^* \times \Sigma_2^* \to \{0,1\}$, such that*

$$\forall \rho \in L\,.\,\big(\rho \in K \iff f(P_{\Sigma_1}(\rho), P_{\Sigma_2}(\rho)) = 1\big).$$

The following lemma comes from [19] and gives necessary and sufficient conditions for joint observability.

Lemma 18 *K is jointly observable with respect to L and Σ_1, Σ_2 iff*

$$\begin{aligned}&\forall \rho, \rho' \in L\,.\\ &(\rho \in K \wedge \rho' \in L - K) \Rightarrow\\ &(P_{\Sigma_1}(\rho) \neq P_{\Sigma_1}(\rho') \vee P_{\Sigma_2}(\rho) \neq P_{\Sigma_2}(\rho')).\end{aligned}$$

The following theorem comes from [19].

Theorem 19 *Checking joint observability is undecidable.*

Based on the above result, we now show undecidability of decentralized control with unbounded-delay communication or without communication. The undecidability result for decentralized control without communication is similar to the result proven in [19]. However, the setting in [19] is slightly different: requirements are expressed by legal languages and non-blockingness. Also, the proofs here use directly the undecidability of joint observability, reducing the latter problem to each of the two control problems.

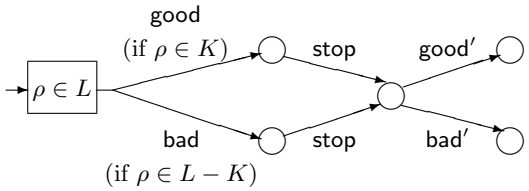

Figure 8: Plant for the proof of Theorem 20.

Theorem 20 *The decentralized control problem with unbounded-delay communication is undecidable.*

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 Γ, alphabets $\Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C} \subseteq \Gamma$, and a property ϕ over Γ, such that K is jointly observable with respect to L, Σ_1, Σ_2 iff there exist controllers C_i over $\Sigma_{iO} \cup \widehat{\Sigma_{jO}}$ with outputs in Σ_{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 $\Gamma = \Sigma \cup \{\mathsf{stop}, \mathsf{good}, \mathsf{good}', \mathsf{bad}, \mathsf{bad}'\}$, where $\mathsf{stop}, \mathsf{good}, \mathsf{good}', \mathsf{bad}, \mathsf{bad}'$ are new events. Let $\Sigma_{1O} = \Sigma_1 \cup \{\mathsf{stop}\}$ and $\Sigma_{2O} = \Sigma_2 \cup \{\mathsf{stop}\}$. Let $\Sigma_{1C} = \{\mathsf{good}', \mathsf{bad}'\}$ and $\Sigma_{2C} = \emptyset$. G is the automaton shown in Figure 8. G initially generates strings in L. At some point, G may decide to stop generating strings in L. Let $\rho \in L$ be the generated string at that point. If $\rho \in K$, G executes good, otherwise, it executes bad. After that, G executes stop and waits for controller 1 to enable either good' or bad'.

Let ϕ be $\{\mathsf{good} \rightsquigarrow \mathsf{good}', \mathsf{bad} \rightsquigarrow \mathsf{bad}'\}$. That is, if G generates a string in K, then we want good' to be enabled, and if G generates a string in $L - K$, then we want 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, Σ_1, Σ_2.

Suppose K is jointly observable with respect to L, Σ_1, Σ_2. Then, there exists a function $f : \Sigma_1^* \times \Sigma_2^* \to \{0,1\}$, 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.[4] C_2 will do nothing, except transmit its observations to C_1. C_1 will initially disable both good' and bad' and wait until it receives $\widehat{\mathsf{stop}}$. 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). If $f(P_{\Sigma_1}(\rho), P_{\Sigma_2}(\rho)) = 1$ then C_1 will enable good', otherwise it will enable bad'.

[4] Note that the controllers may be infinite-state.

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 good, nor bad, thus, ϕ is trivially satisfied. Second, if good ever occurs, then stop will also occur (by maximality, and the fact that stop is uncontrollable), thus, $\widehat{\mathsf{stop}}$ will eventually be received by both controllers (even though this will happen after an unbounded number of steps). Then, good' will be enabled and the specification will be satisfied. The situation is similar if bad occurs.

Now, suppose K is not jointly observable with respect to L, Σ_1, Σ_2, that is, by Lemma 18, 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 ϕ exist. Suppose that G initially performs $\rho\,\mathsf{good}\,\mathsf{stop}$: this can happen, since all events in $\Sigma \cup \{\mathsf{good}, \mathsf{bad}, \mathsf{stop}\}$ are uncontrollable. Also suppose that all events sent by C_2 to C_1 are received by C_1 after C_1 observes stop: this can happen, since the network delay can be greater than the length of ρ. Since ϕ must be satisfied, good' must be enabled after stop. Moreover, bad' cannot be enabled at all after stop, otherwise the specification can be violated.

Suppose C_1 enables good' for the first time (after stop) when it observes $\pi = \sigma_1\,\mathsf{stop}\,\tau$, where τ is a prefix of $\widehat{\sigma_2}\,\widehat{\mathsf{stop}}$, that is, $\widehat{\sigma_2}\,\widehat{\mathsf{stop}} = \tau\,\tau'$. Then, $\rho\,\mathsf{good}\,\mathsf{stop}\,\tau\,\mathsf{good}'\,\tau'$ is a maximal behavior of $(G/C_1 \wedge_\infty C_2)$. We claim that the string $\rho'\,\mathsf{bad}\,\mathsf{stop}\,\tau\,\mathsf{good}'\,\tau'$ is also a maximal behavior of $(G/C_1 \wedge_\infty C_2)$. To see this, note that, having observed π, C_1 is in some state s_1 and $\Lambda_{1C}(s_1) = \{\mathsf{good}'\}$. Suppose G performs $\rho'\,\mathsf{bad}\,\mathsf{stop}$, and all events sent by C_2 to C_1 are received by C_1 after it observes stop. Then, ρ' produces the same observations as ρ, the sequence observed by C_1 is again $\sigma_1\,\mathsf{stop}\,\tau = \pi$. Since C_1 is receptive and deterministic, it reaches state s_1 and enables good'. Thus, $\rho'\,\mathsf{bad}\,\mathsf{stop}\,\tau\,\mathsf{good}'\,\tau'$ is also a maximal behavior of $(G/C_1 \wedge_\infty C_2)$. But this violates the property $\mathsf{bad} \rightsquigarrow \mathsf{bad}'$. Thus, correct controllers cannot exist. ∎

Theorem 21 *The decentralized control problem without communication is undecidable.*

Proof: The proof is similar to the one of Theorem 20, except that a slightly different plant is used, shown in Figure 9. This plant offers the possibility to controller C_2 to transmit its observations to controller C_1, by enabling and disabling controllable events t_i. That is, communication between the two controllers is "simulated" by the plant.

We use the notation of the proof of Theorem 20. Assuming $\Sigma_2 = \{a_1, ..., a_n\}$, let $\Sigma_t = \{t_1, ..., t_n, \mathsf{end}\}$ be a set of new events, and define $\Gamma' = \Sigma \cup \{\mathsf{stop}, \mathsf{good}, \mathsf{good}', \mathsf{bad}, \mathsf{bad}'\} \cup \Sigma_t$. Let $\Sigma_{1O} = \Sigma_1 \cup \{\mathsf{stop}\} \cup \Sigma_t$ and $\Sigma_{2O} = \Sigma_2 \cup \{\mathsf{stop}\} \cup \Sigma_t$. Finally, let $\Sigma_{1C} = \{\mathsf{good}', \mathsf{bad}'\}$ and $\Sigma_{2C} = \Sigma_t$.

The new plant G' is over Γ'. The initial behavior of G' (up to stop) is as the initial behavior of G in the proof of Theorem 20. After stop, G' waits for C_2 to transmit its observation to C_1. C_2 can do this by enabling a sequence $t_{i_1} \cdots t_{i_l}\,\mathsf{end}$, which corresponds to the message "I have observed $a_{i_1} \cdots a_{i_l}$".[5] Finally, C_1 must enable either good' or bad'. The property ϕ is defined to be $\{\mathsf{good} \rightsquigarrow \mathsf{good}', \mathsf{bad} \rightsquigarrow \mathsf{bad}'\}$.

We claim that $(G', \Sigma_{1O}, \Sigma_{2O}, \Sigma_{1C}, \Sigma_{2C}, \phi) \in \mathcal{DC}$ iff K is jointly observable with respect to L, Σ_1, Σ_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 end; second, C_1 waits for $\widehat{\mathsf{end}}$ instead of $\widehat{\mathsf{stop}}$, before it decides.

In the other direction, suppose K is not jointly observable with respect to L, Σ_1, Σ_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 ϕ exist. Suppose G' performs $\rho\,\mathsf{good}\,\mathsf{stop}$ and at this point, for $i = 1, 2$, controller C_i has observed σ_i and is in state s_i. Now, we claim that C_2 transmits some sequence $\tau\,\mathsf{end} \in \Sigma_t^*$. C_2 cannot transmit an infinite sequence, because good' will never take place and ϕ will not be satisfied. Notice that C_2 may enable both end and some t_i at the same time, and it is possible that the plant chooses t_i instead of end. If this happens, C_2 observes t_i and decides what to do next. In any case, C_2 must eventually disable all t_i events and enable end: otherwise, it is possible to have an infinite transmission and good' will never occur. Still, we can see that the sequence τ is not unique. This is not a problem, because for each such sequence τ transmitted to C_1, C_1 must make the correct decision. Therefore, we assume that τ is one of the possible sequences transmitted to C_1.

C_1 enables good' for the first time after stop, once it observes some prefix π of $\tau\,\mathsf{end}$. Now, suppose G' performs $\rho'\,\mathsf{bad}\,\mathsf{stop}$. Since ρ and ρ' yield the same observations to both C_1 and C_2, and the controllers are deterministic and receptive, C_2 will transmit the same sequence(s) to C_1 as when $\rho\,\mathsf{good}\,\mathsf{stop}$ occurred. C_1 will also behave in exactly the same manner and, after observing π, will enable good', violating the specification. Thus, correct controllers cannot exist. ∎

[5] 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.

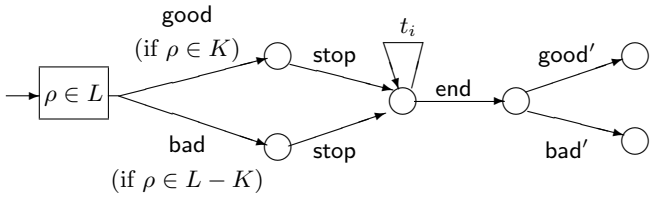

Figure 9: Plant for the proof of Theorem 21.

7 Decidability of a decentralized observation problem with bounded-delay communication

We believe that the decentralized control problem with bounded-delay communication is decidable. Towards such a result, we show decidability of a decentralized observation problem with bounded-delay communication. The latter problem is a modification of the joint observability problem, with bounded-delay communication added between the observers.

7.1 Joint observability with bounded-delay communication

Given regular language L over Σ, subalphabets $\Sigma_1, \Sigma_2 \subseteq \Sigma$, and $k \in \mathsf{N}$, we construct a regular language $L^k_{\Sigma_1,\Sigma_2}$. The latter models the observation of L by two observers which communicate in a k-bounded delay network, as illustrated in Figure 10. Observer i observes all events in Σ_i immediately when they occur, and receives all events in $\widehat{\Sigma_j}$ within a delay of at most k, for $i, j = 1, 2$, $j \neq i$. We use the notation introduced in Section 3 and assume that $\widehat{\Sigma_1}$ and $\widehat{\Sigma_2}$ are disjoint (this can be achieved by renaming, if necessary).

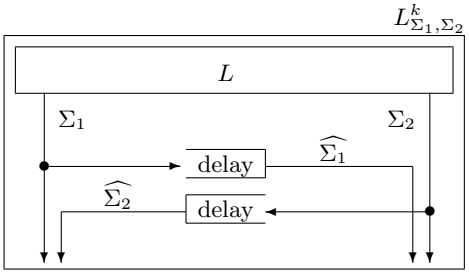

Figure 10: The language L observed by two observers in a bounded-delay network.

Let $(S_L, q_0, \Sigma, \delta, F)$ be a deterministic finite-state automaton equipped with marked states F, which accepts L (i.e., whose marked language is L). Let $\mathcal{Q}_i$ be the set of queues over $\widehat{\Sigma_i}$, for $i = 1, 2$. We define a new automaton, $A^k_{\Sigma_1,\Sigma_2} = (S', q'_0, \Pi, \Delta, F')$, where $S' = S_L \times \mathcal{Q}_1 \times \mathcal{Q}_2$, $q'_0 = (q_0, \emptyset, \emptyset)$, $\Pi = \Sigma \cup \widehat{\Sigma_1} \cup \widehat{\Sigma_2} \cup \{\mathsf{t}\}$, $F' = F \times \{\emptyset\} \times \{\emptyset\}$, and Δ contains the following types of transitions:

1. (*delivery of a message*) $(s, Q_1, Q_2) \xrightarrow{\widehat{a}} (s, Q'_1, Q'_2)$: if Q_i is ready with $\mathsf{head}(Q_i) = (\widehat{a}, 0)$, $\widehat{a} \in \widehat{\Sigma_j}$, in which case, $Q'_i = \mathsf{pop}(Q_i)$, $Q'_j = Q_j$, where $i, j = 1, 2$, $j \neq i$,

2. if no queue is ready,

   (a) (*plant transition and time progress*) $(s, Q_1, Q_2) \xrightarrow{b} (s', Q'_1, Q'_2)$: if $b \in \Sigma$, $s' = \delta(s, b)$ is defined and, for $i = 1, 2$, if $b \in \Sigma_i$, then $Q'_j \in \mathsf{push}_k(Q_j - 1, \widehat{b})$, otherwise, $Q'_j = Q_j - 1$, where $j = 1, 2$, $j \neq i$,

   (b) (*time progress*) $(s, Q_1, Q_2) \xrightarrow{\mathsf{t}} (s, Q_1 - 1, Q_2 - 1)$: if there is no b such that clause 2(a) is satisfied and some queue Q_i, $i = 1, 2$, is non-empty.

The definition of Δ is similar to the one for the decentralized control problem with communication. The t transitions are added to "clear up" the queue of any pending messages at the end of the operation of the plant (an example is given below).

$L^k_{\Sigma_1,\Sigma_2}$ is defined to be the marked language of $A^k_{\Sigma_1,\Sigma_2}$.

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 $L^k_{\Sigma_1,\Sigma_2}$ are the following:

$$\begin{aligned} \pi_1 &= a\,\widehat{a}\,b\,\widehat{b}\,b\,\widehat{b}\,a\,\widehat{a}\,c\,b\,\widehat{b}\,a\,\widehat{a}\,c, \\ \pi_2 &= a\,\widehat{a}\,b\,\widehat{b}\,b\,\widehat{b}\,a\,\widehat{a}\,c\,b\,\widehat{b}\,a\,c\,\mathsf{t}\,\widehat{a}, \\ \pi_3 &= a\,b\,b\,\widehat{a}\,\widehat{b}\,a\,c\,\widehat{b}\,\widehat{a}\,b\,a\,\widehat{b}\,c\,\widehat{a}. \end{aligned}$$

String π_1 corresponds to all messages being delivered immediately (zero delay). String π_2 is the same as π_1, up to the transmission of the last a, which is delayed by two steps. Notice that since operation of the plant ends with the last c, the t event modeling time elapse is necessary to decrement the time-to-live field of pending message $\widehat{a}$ and allow the message to be delivered. In string π_3, the first a is delayed by two steps and the other two a's by one step; the first and third b's are delayed by one step and the second b by two steps; c is not transmitted, since it is not observable by any observer.

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

$$\pi_4 = a\,\widehat{a}\,\widehat{b}\,b\cdots, \qquad \pi_5 = a\,b\,b\,a\,\widehat{a}\cdots.$$

π_4 is not valid since a message cannot be received before it is transmitted. π_5 is not valid since the first a is delayed by three steps whereas the maximum allowed delay is two.

Although the set of all potential states of $A^k_{\Sigma_1,\Sigma_2}$ is infinite, its set of reachable states is finite, as shown below. Thus, the language $L^k_{\Sigma_1,\Sigma_2}$ is regular.

Lemma 22 (queue invariant) *In any reachable state of $A^k_{\Sigma_1,\Sigma_2}$, a queue can contain at most $k+1$ elements. Moreover, if a queue contains $l \geq 1$ elements, then the time-to-live of the head of this queue is at most $k-l+1$.*

Proof: We prove the second part of the lemma, by induction on l. It holds for $l = 1$, by definition of push_k. Assuming it holds for some l, we can prove that it holds for $l' = l+1$ by looking at the definition of the transition relation Δ. Indeed, every time an element is added to the queue using the push_k operator, the time-to-live field of all elements already in the queue is decremented by one. When the queue contains $k+1$ elements, the head of the queue has time-to-live $k-(k+1)+1 = 0$. Thus, by definition of Δ, no more push operations are allowed, until the head is popped. ∎

Corollary 23 *The set of reachable states of $A^k_{\Sigma_1,\Sigma_2}$ is finite.*

Proof: By Lemma 22, a queue in a reachable state of $A^k_{\Sigma_1,\Sigma_2}$ contains at most $k+1$ elements. Thus, the number of reachable states is bounded by $|S_L|\cdot|\Sigma_1|^{k+1}\cdot|\Sigma_2|^{k+1}$. ∎

We now formally define the problem of joint observability with bounded-delay communication.

Definition 24 (bounded-delay joint observability) *Given regular languages K, L over Σ, $K \subseteq L$, $\Sigma_1, \Sigma_2 \subseteq \Sigma$ and $k \in \mathsf{N}$, K is said to be* jointly observable with bounded-delay k with respect to L and Σ_1, Σ_2 *if there exists a total function $g : (\Sigma_1 \cup \widehat{\Sigma_2})^* \times (\Sigma_2 \cup \widehat{\Sigma_1})^* \to \{0,1\}$, such that*

$$\begin{array}{l}\forall \pi \in L^k_{\Sigma_1,\Sigma_2}\ .\\ P_\Sigma(\pi) \in K \iff g(P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi), P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi)) = 1.\end{array}$$

The following lemma gives necessary and sufficient conditions for bounded-delay joint observability.

Lemma 25 *K is jointly observable with bounded-delay k with respect to L and Σ_1, Σ_2 iff*

$$\begin{array}{l}\forall \pi, \pi' \in L^k_{\Sigma_1,\Sigma_2}\ .\\ (P_\Sigma(\pi) \in K \wedge P_\Sigma(\pi') \in L - K) \Rightarrow\\ (P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi) \neq P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi') \vee P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi) \neq P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi')).\end{array} \quad (2)$$

Proof: Assume the negation of Condition (2), that is, assume there exist $\pi, \pi' \in L^k_{\Sigma_1,\Sigma_2}$ such that $P_\Sigma(\pi) \in K$, $P_\Sigma(\pi') \in L - K$, $P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi) = P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi')$ and $P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi) = P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi')$. Let $\sigma_1 = P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi) = P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi')$ and $\sigma_2 = P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi) = P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi')$. Then, $g(\sigma_1,\sigma_2)$ must equal both 1 (because of π) and 0 (because of π'), thus, g cannot exist.

Conversely, assume Condition (2) holds and define, for $\sigma_1 \in (\Sigma_1 \cup \widehat{\Sigma_2})^*$, $\sigma_2 \in (\Sigma_2 \cup \widehat{\Sigma_1})^*$,

$$g(\sigma_1,\sigma_2) = \begin{cases} 1, & \text{if } \exists \pi \in L^k_{\Sigma_1,\Sigma_2}\ .\ P_\Sigma(\pi) \in K \wedge \\ & P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi) = \sigma_1 \wedge P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi) = \sigma_2, \\ 0, & \text{otherwise.}\end{cases}$$

We claim that g solves the bounded-delay decentralized observation problem. Indeed, let $\pi \in L^k_{\Sigma_1,\Sigma_2}$. If $P_\Sigma(\pi) \in K$ then, by definition, $g(P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi), P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi)) = 1$. If $P_\Sigma(\pi) \notin K$ then we claim that $g(P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi), P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi)) = 0$. Otherwise, there must exist $\pi' \in L^k_{\Sigma_1,\Sigma_2}$ such that $P_\Sigma(\pi') \in K$, $P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi') = P_{\Sigma_1\cup\widehat{\Sigma_2}}(\pi)$ and $P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi') = P_{\Sigma_2\cup\widehat{\Sigma_1}}(\pi)$, which contradicts Condition (2). ∎

7.2 Decidability

Theorem 26 *Checking joint observability with bounded-delay k is decidable, for any $k \in \mathsf{N}$.*

In the rest of the section, we prove the theorem. Define $\Gamma_1 = \Sigma_1 \cup \widehat{\Sigma_2}$ and $\Gamma_2 = \Sigma_2 \cup \widehat{\Sigma_1}$. Also, $\Pi = \Sigma \cup \widehat{\Sigma_1} \cup \widehat{\Sigma_2} \cup \{\mathsf{t}\}$, as defined above.

The algorithm for checking bounded-delay joint observability is to build an automaton A and check that the marked language of A is empty. A is constructed as a special product of two finite automata, A_1 and A_2. A_1 generates π and A_2 generates π', so that π and π' contradict Condition (2). To ensure that $P_\Sigma(\pi) \in K$, A_1 is defined to be the automaton generating the language $L_1 = L^k_{\Sigma_1,\Sigma_2} \cap P_\Sigma^{-1}(K)$. To ensure that $P_\Sigma(\pi') \in L-K$, A_2 is defined to be the automaton generating the language $L_2 = L^k_{\Sigma_1,\Sigma_2} \cap P_\Sigma^{-1}(L-K)$. Regular languages are closed under intersection, complementation and inverse projection. Thus, both L_1 and L_2 are regular and A_1, A_2 are finite automata. Let $A_i = (S_i, q_0^i, \Pi, \Delta_i, F_i)$, for $i = 1, 2$.

To ensure that the projections of π and π' on Γ_1 are the same, the two automata synchronize on all transitions labeled by an event in Γ_1. To ensure that the projections of π and π' on Γ_2 are the same, the product automaton is equipped with a FIFO queue storing events in $\Gamma_2 - \Gamma_1$. This queue is simpler than the one defined in Section 3.3, namely, it is just a finite string of events, without time-to-live field. To avoid confusion, we call this queue a *buffer*.

To understand the usage of the buffer, assume for the moment that Γ_1 and Γ_2 are disjoint (this is not necessarily the case, since Σ_1 and Σ_2 are not necessarily disjoint). Initially the buffer is empty. The first automaton among A_1 and A_2 that generates an event in Γ_2, say, A_1, inserts it in the buffer. From this point on, every time A_1 generates an event in Γ_2, it appends it at the end of the buffer, and this until the buffer becomes empty again. We say that A_1 is the *leader*. Every time A_2 generates an event in Γ_2, this event must be the same as the event in the head of the buffer. If this condition is satisfied, A_2 generates the event and removes it from the head of the buffer. Otherwise A_2 cannot generate the event. We say that A_2 is the *follower*. The operation continues in the same way until the buffer becomes empty. The next time some automaton generates an event in Γ_2, this automaton becomes the leader and the other automaton the follower.

For the general case where Γ_1 and Γ_2 are not disjoint, the same rules hold, with the additional condition that the buffer must be empty whenever an event in $\Gamma_1 \cap \Gamma_2$ is generated. This ensures that the follower "catches up" with the leader whenever some event in $\Gamma_1 \cap \Gamma_2$ occurs. The operation must end with the buffer being empty. This ensures that the follower "catches up" with the leader at the end, thus, the projections of π and π' on Γ_2 are identical.

Formally, A is defined to be the automaton (S, q_0, Π, Δ, F), where $S = S_1 \times S_2 \times (\Gamma_2 - \Gamma_1)^* \times \{1, 2\}$, $q_0 = (q_0^1, q_0^2, \epsilon, 1)$, $F = F_1 \times F_2 \times \{\epsilon\} \times \{1, 2\}$. A state s of A is a tuple (s_1, s_2, β, l), consisting of the local states s_1 and s_2 of A_1 and A_2, the contents of the buffer β, and the index of the leader l. When the buffer is empty ($\beta = \epsilon$) the value of l is unimportant.

The transition relation Δ is defined below. To simplify its definition, we introduce the predicate move_1 (respectively, move_2) as a short notation for $s_1' \in \Delta_1(s_1, a) \wedge s_2' = s_2$ (respectively, $s_2' \in \Delta_2(s_2, a) \wedge s_1' = s_1$). Δ contains the following types of transitions:

1. $(s_1, s_2, \epsilon, l) \xrightarrow{a} (s_1', s_2', \epsilon, l)$, such that $a \in \Gamma_1 \cap \Gamma_2$ and $s_1' \in \Delta_1(s_1, a)$ and $s_2' \in \Delta_2(s_2, a)$,
2. $(s_1, s_2, \beta, l) \xrightarrow{a} (s_1', s_2', \beta, l)$, such that $a \in \Gamma_1 - \Gamma_2$ and $s_1' \in \Delta_1(s_1, a)$ and $s_2' \in \Delta_2(s_2, a)$,
3. $(s_1, s_2, \beta, l) \xrightarrow{a} (s_1', s_2', \beta', l')$, such that $a \in \Gamma_2 - \Gamma_1$ and
   - either $\beta = \epsilon$ (no current owner) and
     - either move_1 and $\beta' = a$ and $l' = 1$ (A_1 becomes owner),
     - or move_2 and $\beta' = a$ and $l' = 2$ (A_2 becomes owner),
   - or $\beta \neq \epsilon$ and $l = 1$ (current owner is A_1) and
     - either move_1 and $\beta' = \beta a$ and $l' = l$,
     - or move_2 and $\beta = a\beta'$ and $l' = l$,
   - or $\beta \neq \epsilon$ and $l = 2$ (current owner is A_2) and
     - either move_1 and $\beta = a\beta'$ and $l' = l$,
     - or move_2 and $\beta' = \beta a$ and $l' = l$,
4. $(s_1, s_2, \beta, l) \xrightarrow{a} (s_1', s_2', \beta, l)$, such that $a \in \Pi - (\Gamma_1 \cup \Gamma_2)$ and either move_1 or move_2.

Note that the "either-or" above are exclusive. Also note that the non-determinism in A comes from the non-determinism of A_1 and A_2. Otherwise, the transition relation Δ is deterministically defined.

The first two clauses define transitions labeled by an event in Γ_1. A_1 and A_2 synchronize on all such events. If the event is also in Γ_2 (first clause) then the buffer must be empty. The third clause defines transitions labeled by an event in $\Gamma_2 - \Gamma_1$. Different cases are distinguished, depending on whether an owner currently exists or not. The fourth clause defines transitions labeled by an event in $\Pi - (\Gamma_1 \cup \Gamma_2)$. These are unobservable events, on which no constraints are imposed. The semantics on these events are *interleaving*, that is, either A_1 or A_2 moves, but not both at the same time.

In the rest of the section we will use the following notation, to avoid confusion: $\rightarrow$ will refer to the transition relation Δ of A, while $\rightarrow_1, \rightarrow_2$ will refer to the transition relation Δ_1, Δ_2 of A_1, A_2, respectively.

Lemma 27 *Let $\pi, \pi' \in \Pi^*$, such that $s_1 \xrightarrow{\pi}_1 s_1'$, $s_2 \xrightarrow{\pi'}_2 s_2'$, $P_{\Gamma_1}(\pi) = P_{\Gamma_1}(\pi')$ and $P_{\Gamma_2}(\pi) = P_{\Gamma_2}(\pi')$. Then, there exists $\tau \in \Pi^*$, such that $(s_1, s_2, \epsilon, 1) \xrightarrow{\tau} (s_1', s_2', \epsilon, l')$, for some $l' \in \{1, 2\}$.*

Sketch of proof: We proceed by "composing" the moves of A_1 and A_2 in A, according to π and π', respectively. For instance, assuming $a \in \Gamma_1 \cap \Gamma_2$, $b \in \Gamma_1 - \Gamma_2$, $c \in \Gamma_2 - \Gamma_1$ and $u_1, u_2 \in \Pi - (\Gamma_1 \cup \Gamma_2)$, and letting $pi = u_1\, a\, b\, c$ and $\pi' = u_2\, a\, c\, b$, we get the following possible run in A: $(s_1^0, s_2^0, \epsilon, 1) \xrightarrow{u_1} (s_1^1, s_2^0, \epsilon, 1) \xrightarrow{u_2} (s_1^1, s_2^1, \epsilon, 1) \xrightarrow{a} (s_1^2, s_2^2, \epsilon, 1) \xrightarrow{c}$

$(s_1^2, s_2^3, c, 2) \xrightarrow{b} (s_1^3, s_2^4, c, 2) \xrightarrow{c} (s_1^4, s_2^4, \epsilon, 2)$, where $s_1^0 = s_1$, $s_2^0 = s_2$, $s_1^4 = s_1'$ and $s_2^4 = s_2'$. Note that another run exists, where u_1 and u_2 are executed in the reverse order and the rest of the run does not change. The resulting behavior of A is τ. In the example above, $\tau = u_1\, u_2\, a\, c\, b\, c$. Equality of projections of π and π' on Γ_1 and Γ_2 ensures that the synchronizing transitions can occur and that the buffer is left empty in the end. ■

Lemma 28 *Let $\tau \in \Pi^*$, such that $(s_1, s_2, \beta, l) \xrightarrow{\tau} (s_1', s_2', \beta', l')$. Then, there exist $\pi, \pi' \in \Pi^*$ such that $s_1 \xrightarrow{\pi}_1 s_1'$, $s_2 \xrightarrow{\pi'}_2 s_2'$, and $P_{\Gamma_1}(\pi) = P_{\Gamma_1}(\pi')$. Moreover, if $\beta = \beta' = \epsilon$, then $P_{\Gamma_2}(\pi) = P_{\Gamma_2}(\pi')$.*

Sketch of proof: We proceed by "decomposing" τ into π, corresponding to the steps where automaton A_1 moves and π', corresponding to the steps where automaton A_2 moves. $P_{\Gamma_1}(\pi) = P_{\Gamma_1}(\pi')$ follows by definition of Δ and the fact that the two automata synchronize on events in Γ_1.

If $\beta = \beta' = \epsilon$, then we proceed by induction on the number of substrings of τ, $\tau = \tau_1 \cdots \tau_m$, such that $s_i \xrightarrow{\tau_i} s_{i+1}$, $s_i = (s_1^i, s_2^i, \epsilon, l^i)$, where each τ_i has the property that either the buffer is empty all along $s_i \xrightarrow{\tau_i} s_{i+1}$, or the buffer is empty in s_i and s_{i+1} and non-empty in-between. Then, we "decompose" each τ_i into π_i, π_i', as indicated above. In case the buffer is empty all along $s_i \xrightarrow{\tau_i} s_{i+1}$, no events in $\Gamma_2 - \Gamma_1$ occur in τ_i and $P_{\Gamma_2}(\pi_i) = P_{\Gamma_2}(\pi_i')$ follows from the fact that A_1 and A_2 synchronize on $\Gamma_2 \cap \Gamma_1$. If the buffer is non-empty all along $s_i \xrightarrow{\tau_i} s_{i+1}$, except at s_i and s_{i+1}, then observe that no events in $\Gamma_2 \cap \Gamma_1$ can occur in τ_i and that there is a single owner during this period. $P_{\Gamma_2}(\pi_i) = P_{\Gamma_2}(\pi_i')$ follows from the fact that every event inserted in the buffer by the owner must be removed by the follower, in the same order. ■

Lemma 29 *The set of reachable states of A is finite.*

Proof: We will show that for any reachable state (s_1, s_2, β, l) of A, $|\beta| \leq 2k+1$. Thus, the number of reachable states is at most $|S_1| \cdot |S_2| \cdot |\Gamma_2 - \Gamma_1|^{2k+1} \cdot 2$. (Recall that S_1, S_2 are the sets of states of automata A_1, A_2 out of which A is built.)

Recall that the buffer stores events in $\Gamma_2 - \Gamma_1 = (\Sigma_2 \cup \widehat{\Sigma_1}) - (\Sigma_1 \cup \widehat{\Sigma_2}) = (\Sigma_2 - \Sigma_1) \cup \widehat{\Sigma_1}$ (from the fact that $\widehat{\Sigma_1}$ and $\widehat{\Sigma_2}$ are disjoint). We will show that every event inserted in the buffer by some automaton will be removed after this automaton has performed at most k steps, where a step corresponds to an event in $\Sigma \cup \{\mathtt{t}\}$. This suffices to prove that the size of the buffer never grows above $2k+1$, since in k steps, at most $2k$ events in $(\Sigma_2 - \Sigma_1) \cup \widehat{\Sigma_1}$ can occur. Indeed, every event in $\Sigma_2 - \Sigma_1$ counts as a step and every event in $\widehat{\Sigma_1}$ is generated by an event in Σ_1 occurring at most k steps earlier and also counting as a step.

In what follows we prove the above claim. We assume, without loss of generality, that the event is inserted in the buffer by A_1. This means that A_1 is the leader at this point and will remain the leader until the buffer becomes empty. We distinguish two cases, illustrated in Figure 11.

Case 1: the event is some $\widehat{a} \in \widehat{\Sigma_1}$. At most k steps before A_1 inserts $\widehat{a}$, A_1 must have generated $a \in \Sigma_1$. At that point, A_1 synchronized with A_2, which also generated a. A_2 must generate $\widehat{a}$ at most k steps after generating a. Since A_2 is the follower, it can generate $\widehat{a}$ only after A_1 does so. Upon generating the event, A_2 removes $\widehat{a}$ from the head of the buffer.

Case 2: the event is some $b \in \Sigma_2 - \Sigma_1$. At most k steps after A_1 generates b, it generates $\widehat{b} \in \widehat{\Sigma_2}$, synchronizing with A_2 at that point. In order to generate $\widehat{b}$, A_2 must have generated b at most k steps earlier. Since A_2 is the follower, it can generate b only after A_1 does so. Upon generating the event, A_2 removes b from the head of the buffer. ■

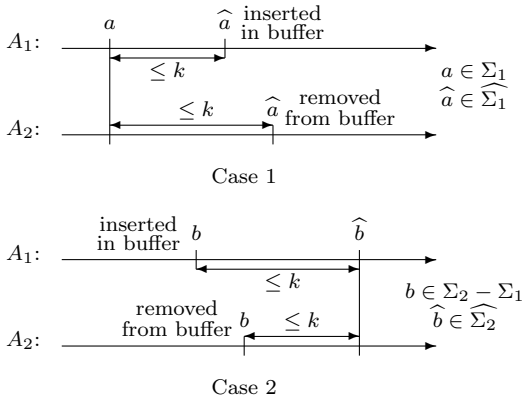

Figure 11: The two cases used in the proof of Lemma 29.

We are now ready to assemble the previous results into a proof of Theorem 26.

Proof: [of Theorem 26] We claim that K is jointly observable with bounded-delay k w.r.t. L and Σ_1, Σ_2 iff $L_m(A) = \emptyset$. Indeed, by Lemma 25, K is jointly observable iff Condition (2) holds. By Lemma 27, if there exist π, π' violating Condition (2), then $L_m(A) \neq \emptyset$. Conversely, if $L_m(A) \neq \emptyset$, then there exist π, π' violating Condition (2), by Lemma 28. Since A is a finite-state automaton, checking $L_m(A) = \emptyset$ is decidable. ■

7.3 Complexity of checking bounded-delay joint observability

Checking $L_m(A) = \emptyset$ is linear in the size of A. From the proof of Lemma 29, the number of states of A is bounded by $|S_1| \cdot |S_2| \cdot |\Gamma_2 - \Gamma_1|^{2k+1} \cdot 2$, where S_1, S_2 are the sets of states of automata A_1, A_2. Recall that A_1 and A_2 are the automata generating the languages $L_1 = L^k_{\Sigma_1,\Sigma_2} \cap P_\Sigma^{-1}(K)$ and $L_2 = L^k_{\Sigma_1,\Sigma_2} \cap P_\Sigma^{-1}(L-K)$, respectively. Assume languages L and K are given as finite-state automata A_L and A_K with sets of states S_L and S_K, respectively. The automaton A_{L-K} accepting $L-K$ can be implemented by building the automaton $A_{\overline{K}}$ accepting the complement of K and then building the product of A_L and $A_{\overline{K}}$ accepting the intersection of L and the complement of K. Because of exponential worst-case cost of complementation, the worst-case size of A_{L-K} is $O(|S_L| \cdot 2^{|S_K|})$. Inverse projection can be implemented by adding "self-loop" transitions which "cover" the missing letters, thus, does not change the number of states of an automaton. $L^k_{\Sigma_1,\Sigma_2}$ is generated by automaton $A^k_{\Sigma_1,\Sigma_2}$ which, by Corollary 23, has at most $|S_L| \cdot |\Sigma_1|^{k+1} \cdot |\Sigma_2|^{k+1}$ states. Thus, $|S_1|$ is bounded by $|S_L| \cdot |\Sigma_1|^{k+1} \cdot |\Sigma_2|^{k+1} \cdot |S_K|$ and $|S_2|$ is $O(|S_L| \cdot |\Sigma_1|^{k+1} \cdot |\Sigma_2|^{k+1} \cdot |S_L| \cdot 2^{|S_K|})$. Putting it all together, along with the fact that $|\Sigma_i| \leq |\Sigma|$ and $|\Gamma_2 - \Gamma_1| \leq 2 \cdot |\Sigma|$, we get that the number of states of A is

$$O(|S_L|^3 \cdot |S_K| \cdot 2^{|S_K|} \cdot |\Sigma|^{6k+5} \cdot 2^{2(k+1)}).$$

8 Summary and perspectives

We have introduced a framework of decentralized control for discrete-event systems with various types of communication: bounded-delay, unbounded-delay or no communication at all. We have shown that, for a fixed, simple communication policy ("transmit everything you observe") a natural hierarchy of control problems arises, where the smaller the network delays are, the more the problems that admit a solution. We have also shown that checking the existence of controllers in the cases of unbounded-delay or no communication are undecidable problems. We conjectured that the problem becomes decidable in the case of bounded-delay communication. Towards such a result, we showed decidability of a related bounded-delay decentralized observation problem.

Apart from proving the conjecture, other perspectives include removing some of the assumptions of our model, namely, the lossless and FIFO properties of the network. We believe that removing these assumptions should not affect the hierarchy or (un)decidability results. The algorithm we presented for checking bounded-delay joint observability has a high complexity, exponential in the delay bound k. It would be interesting to examine whether more efficient algorithms exist. Another direction is to study the synthesis of the communication policy itself.

The decentralized control problems formulated in this paper ask whether controllers exist, not excluding infinite-state controllers. Indeed, as shown in [12], there are problems which can be solved with infinite-state controllers but not with finite-state controllers. It is an interesting open problem to examine the decidability of decentralized control problems where controllers are required to be finite-state.

Acknowledgments: The author 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*, 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. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[4] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on 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 nondeterministic systems under partial observation. *SIAM Journal on Control and Optimization*, 35(2):363–383, 1997.

[7] O. Kupferman and M. Vardi. Synthesizing distributed systems. In *Logic in Computer Science*, 2001.

[8] H. Lamouchi and J. Thistle. Effective control synthesis for DES under partial observations. In *IEEE Conference on Decision and Control*, 2000.

[9] F. Lin and W. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[10] P. Madhusudan and P.S. Thiagarajan. Distributed controller synthesis for local specifications. In *28th ICALP, Crete, Greece*, LNCS 2076, 2001.

[11] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the*

31th IEEE Symposium Foundations of Computer Science, pages 746–757, 1990.

[12] A. Puri, S. Tripakis, and P. Varaiya. Problems and examples of decentralized observation and control for discrete event systems. In *CAV'01 Symposium on Supervisory Control for Discrete Event Systems (SCODES)*, 2001. Also in B. Caillaud, P. Darondeau, L. Lavagno and X. Xie (eds.), *Synthesis and Control of Discrete Event Systems*, Kluwer Academic Publishers, 2002.

[13] P. Ramadge and W. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, January 1989.

[14] L. Ricker and J. van Schuppen. Decentralized failure diagnosis with asynchronous communication between supervisors. In *European Control Conference*, 2001.

[15] S. Ricker and K. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Transactions on Automatic Control*, 45(9), September 2000.

[16] K. Rudie, S. Lafortune, and F. Lin. Minimal communication in a distributed discrete-event control system. In *American Control Conference*, 1999.

[17] K. Rudie and J.C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40(7), 1995.

[18] K. Rudie and W. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37, 1992.

[19] S. Tripakis. Undecidable problems of decentralized observation and control. In *IEEE Conference on Decision and Control (CDC'01)*, 2001.

[20] S. Tripakis. Decentralized control of discrete event systems with bounded or unbounded delay communication. In *6th International Workshop on Discrete Event Systems (WODES'02)*. IEEE CS Press, 2002.

[21] S. Tripakis. Decentralized control of discrete event systems with bounded or unbounded delay communication. *IEEE Transactions on Automatic Control*, 49(9), September 2004.

[22] Y. Willner and M. Heymann. On supervisory control of concurrent discrete-event systems. *International Journal of Control*, 54(5), 1991.

[23] K.C. Wong and J.H. van Schuppen. Decentralized supervisory control of discrete-event systems with communication. In *Workshop On Discrete Event Systems*, 1996.

[24] T. Yoo and S. Lafortune. New results on decentralized supervisory control of discrete-event systems. In *IEEE Conference on Decision and Control*, 2000.