Proceedings of the
46th IEEE Conference on Decision and Control
New Orleans, LA, USA, Dec. 12-14, 2007

Mind the Gap: Expanding Communication Options in Decentralized Discrete-Event Control

S.L. Ricker and B. Caillaud

Abstract— **Frameworks that incorporate communication into decentralized supervisory control theory address the following problem: find locations in the evolution of the plant behavior where some supervisors send information so that a supervisor that was unable to make the correct control decision prior to receiving external information, is now capable of making the correct control decision. Proposed solutions to this problem identify an earliest and a latest placement where such communication results in the synthesis of a correct control solution.**

In addition to a first and last communication opportunity, there may be a selection of intermediate possibilities where communication would produce the correct control solution. We present a computable procedure to identify a broader range of suitable communication locations.

I. Introduction

The role of communication (with no delay) in decentralized discrete-event control problems has been explored under a variety of communication protocols. The basic idea of this class of problems is that a designated communicating supervisor follows a communication protocol and provides information to the supervisor taking a specific control decision, providing the latter supervisor with enough information to definitively make the correct control decision. The design of a communication protocol for decentralized control problems requires the construction of a non-empty set of communications that will admit a correct control solution [14], [10], [2]. We are interested in bridging the gap between the "communicate as late as possible"[1] strategy of [2] and the "communicate as early as possible"[2] strategy in our earlier work [10]. A communication sublanguage of the plant language is the set of sequences that precede the occurrence of a communication event, whether that event be a special communication event or the last observed event for the communicating supervisor or a set of the communicating supervisor's local state estimates.

Communication is introduced into a decentralized supervisory control problem when the language requiring control does not satisfy co-observability [12], a condition that determines whether or not the set of control decisions for the collection of decentralized supervisors will allow *all* the correct control decisions. One of the challenges in developing decentralized communication protocols lies in the isolation of sequences that violate co-observability. The basic idea for designing a communication protocol is to augment (with designated communicated information) the content of sequences that were previously not co-observable, so that after communication occurs, the sequences are co-observable. We propose a new computable strategy for identifying communication sublanguages, where communication options lie between the first and last communication opportunities (inclusive).

This work was supported by NSERC.

L. Ricker is with the Department of Mathematics and Computer Science, Mount Allison University, Sackville, NB, Canada lricker@mta.ca. B. Caillaud is with IRISA-INRIA, Rennes, France. Benoit.Caillaud@irisa.fr

[1]Communication occurs only along an illegal sequence.

[2]Communication occurs either along an illegal or legal sequence.

II. Background

A. Supervisory Control

Supervisory control, as formulated in [8], [9], assumes that both the system requiring control, called the *plant*, and the desired behavior, often called the *legal* behavior, are described by formal languages over a given *alphabet*, denoted by Σ. The analysis in this paper is restricted to *regular* languages, a class of languages that can be described by finite automata. The goal of the control problem is to synthesize a *supervisor* of the plant such that, based on its observations of the plant behavior, the supervisor permits the occurrence of only the legal behavior (or possibly some subset of legal behavior) by issuing either an *enable* or *disable* command for different behaviors. The synthesis of the resulting control policy (determining which behaviors to enable or to disable) is further complicated by the fact that there are some behaviors that a supervisor cannot disable.

Decentralized supervisory control problems [5], [12] involve the synthesis of n supervisors (for $n \geq 2$), each of whom has only a partial view of the plant. That is, some of the behavior in the plant is *unobservable* to each supervisor. We use I to represent the set of n decentralized supervisors $\{1, \ldots, n\}$. The ability to synthesize a control policy relies on the existence of *at least one* supervisor that can make the correct control decision to keep the plant performing within the legal behavior. The remainder of this section will focus on the notation for synthesizing decentralized control strategies.

For any strings $s, t \in \Sigma^*$, where Σ^* is the Kleene closure of Σ, we say that t is a *prefix* of s, denoted $t \preceq s$, if $\exists w \in \Sigma^*$ such that s=tw. When $t \preceq s$, we also define $s \setminus t = w$. $L \subseteq \Sigma^*$, the *prefix-closure* of L is a language, denoted by $\overline{L}$, consisting of all prefixes of strings of L: $\overline{L} := \{t \in \Sigma^* \mid t \preceq s\}$. Because every string is a prefix of itself, $L \subseteq \overline{L}$. A language is said to be *prefix-closed* if $L = \overline{L}$. We assume that the plant language is prefix-closed.

The partial view that a decentralized supervisor i has of the plant is described by the set of observable events, denoted $\Sigma_{i,o} \subset \Sigma$ for $i \in I$. To describe a supervisor's view of the system behavior, we use the *canonical projection* $P_i : \Sigma^* \rightarrow \Sigma_{i,o}^*$, for $i \in I$. This operator effectively

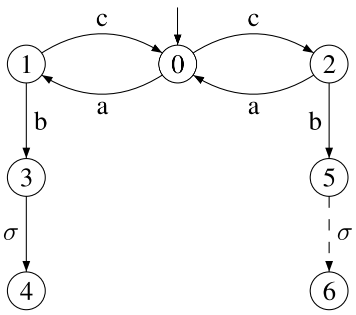

Fig. 1. A joint plant M_L and legal automaton M_K. The collection of all edges is M_L, whereas the collection of solid edges is M_K.

"erases" those events σ from a string t that are not found in $\Sigma_{i,o}$: $P_i(\varepsilon) = \varepsilon$, where ε denotes the empty string; $P_i(\sigma) = \varepsilon$ if $\sigma \in \Sigma \setminus \Sigma_{i,o}$; $P_i(\sigma) = \sigma$ if $\sigma \in \Sigma_{i,o}$; and $P_i(t\sigma) = P_i(t)P_i(\sigma)$, $t \in \Sigma^*, \sigma \in \Sigma$. Thus if the plant generates sequence t, then $P_i(t)$ indicates the sequence of events observed by the decentralized supervisor i. The *inverse projection* of P_i is the mapping from $\Sigma_{i,o}^*$ to 2^{Σ^*}: $P_i^{-1}(t) = \{u \mid P_i(u) = t\}$.

To establish supervision on the plant, we partition the set of events Σ into the disjoint sets Σ_c, *controllable* events, and Σ_{uc}, *uncontrollable* events. Controllable events are those events whose occurrence is preventable (i.e., may be disabled). Uncontrollable events are those events which cannot be prevented and are deemed permanently enabled. The set of controllable events is further partitioned into (not necessarily disjoint) subsets of events that are controlled by supervisor i: $\Sigma_{i,c} \subseteq \Sigma_c$ (for $i \in I$), where $\Sigma_c = \cup_{i=1}^{n} \Sigma_{i,c}$. We denote the set of supervisors for which $\sigma \in \Sigma_{i,c}$ by I_σ. Finally, we denote the cardinality of I_σ by $|I_\sigma|$.

Formally, a deterministic finite automaton is a tuple:

$$M = (Q, \Sigma, \delta, q_0, F),$$

where M is the name of the automaton; Q is its set of states; Σ is the finite alphabet; δ is a *transition function*, a partial function δ: $\Sigma \times Q \to Q$; $q_0 \in Q$ is the *initial state*; and $F \subseteq Q$ is a set of final states. Let the language generated by M be denoted by $\mathcal{L}(M)$. The automaton representing the plant is denoted by M_L, whereas the automaton representing the legal behavior is denoted by M_K. An example of a plant and associated legal behavior, where $I = \{1, 2\}$, is shown in figure 1. For this example, $\Sigma = \{a, b, c, \sigma\}$ and let $\Sigma_{1,o} = \{a, b, \sigma\}$, $\Sigma_{2,o} = \{b, c, \sigma\}$ and $\Sigma_{1,c} = \{a, b, \sigma\}$, $\Sigma_{2,c} = \{c\}$. Therefore $I_a = I_b = I_\sigma = \{1\}$ and $I_c = \{2\}$.

Decentralized supervisors can be synthesized if the corresponding DES is co-observable [12].

A prefix-closed language K is *co-observable* with respect to L and P_i, for $i = 1, \ldots, n$, if for all $t \in \overline{K}$ and for all $\sigma \in \Sigma$,

$$\begin{aligned}&(t\sigma \notin \overline{K}) \wedge (t\sigma \in L) \Rightarrow \exists i \in I_\sigma \\ &\text{such that } P_i^{-1}[P_i(t)]\sigma \cap \overline{K} = \emptyset\end{aligned} \tag{1}$$

It must be the case that based only on its partial view of a sequence, a decentralized supervisor can make the correct control decision.

The equivalent condition in the case of a single (centralized) supervisor is called *observability*. The observations of a centralized supervisor are captured by a projection operator $P : \Sigma^* \to \Sigma_o^*$. A prefix-closed language K is said to be *observable* with respect to L, P, and Σ_c if for all $t \in \overline{K}$ and for all $\sigma \in \Sigma_c$,

$$(t\sigma \notin \overline{K}) \wedge (t\sigma \in L) \Rightarrow P^{-1}[P(t)] \cap \overline{K} \neq \emptyset \tag{2}$$

For purposes of designing a decentralized communication protocol, K must be observable but must not be co-observable.

In figure 1, let $K = \mathcal{L}(M_K)$ and $L = \mathcal{L}(M_L)$. It is the case that K is observable, since all illegal sequences can be distinguished from all legal sequences. But K is *not* co-observable since it is possible to find sequences, such as $t = accacab$, where $t\sigma \in L \setminus K$, and for $I = \{1, 2\}$ it is the case that $P_1^{-1}[P_1(t)]\sigma \cap K = \{caacab\sigma\} \neq \emptyset$ and $P_2^{-1}[P_2(t)]\sigma \cap K = \{accaacab\sigma\} \neq \emptyset$.

B. Labels

Synchronization between the plant and the decentralized supervisors is defined via *synchronization vectors* [1], here referred to as *labels*.

We begin by introducing an augmented set of supervisors $I_0 = I \cup \{0\}$, where 0 represents the plant. A label $\ell : I_0 \longrightarrow \Sigma \cup \{\varepsilon\}$ is a mapping from each supervisor to either an event from Σ or ε. The *support of a label*, denoted by $||\ell||$, is $\{i \mid \ell(i) \neq \varepsilon\}$.

It is possible to define a partial order relation on labels: $\ell_1 \leq \ell_2$ iff $\forall i \in I_0$, $\ell_1(i) \neq \varepsilon \Rightarrow \ell_1(i) = \ell_2(i)$. This partial order relation subsumes a greatest lower bound $\wedge$ and a partial least upper bound $\vee$. The greatest lower bound is computed as follows:

$$\forall i \in I_0, \quad (\ell_1 \wedge \ell_2)(i) = \begin{cases} \ell_1(i), & \text{if } \ell_1(i) = \ell_2(i); \\ \varepsilon, & \text{otherwise.} \end{cases}$$

The least upper bound, $\ell_1 \vee \ell_2$, is defined whenever ℓ_1 and ℓ_2 are compatible. Two labels ℓ_1, ℓ_2 are *compatible*, denoted $\ell_1 \uparrow \ell_2$, iff $\forall i \in I_0,\ \ell_1(i) = \varepsilon$ or $\ell_2(i) = \varepsilon$ or $\ell_1(i) = \ell_2(i)$.

The least upper bound is computed as follows:

$$\forall i \in I_0,\ (\ell_1 \vee \ell_2)(i) = \begin{cases} \ell_1(i), & \text{if } \ell_1(i) \neq \varepsilon; \\ \ell_2(i), & \text{if } \ell_2(i) \neq \varepsilon; \\ \varepsilon, & \text{otherwise.} \end{cases}$$

Two labels, ℓ_1 and ℓ_2 are *independent*, denoted $\ell_1 | \ell_2$, iff

$$\forall i \in I_0\ \ell_1(i) = \varepsilon \text{ or } \ell_2(i) = \varepsilon.$$

Let $\ell_1 \leq \ell_2$, then $\ell_2 \backslash \ell_1$ is the least label such that $\ell_1 \vee (\ell_2 \backslash \ell_1) = \ell_2$. The computation of $\ell_2 \backslash \ell_1$ is straightforward:

$$\forall i \in I_0, \quad \ell_2 \backslash \ell_1(i) = \begin{cases} \ell_2(i), & \text{if } \ell_1(i) = \varepsilon; \\ \varepsilon, & \text{otherwise.} \end{cases}$$

Labels are generated from a given finite set of *atoms*, denoted $\mathcal{A}$, using the least upper bound operation, whenever the operation is defined. For our purposes, atoms define elementary synchronizations between the plant and the decentralized supervisors. It will also be useful to compute the closure of $\mathcal{A}$ under least upper bound: $closure(\mathcal{A})$ is the least set containing the empty label, $(\varepsilon, \ldots, \varepsilon)$ and such that $\forall a \in \mathcal{A}$, $\forall \ell \in closure(\mathcal{A})$, $a \uparrow \ell \Rightarrow a \vee \ell \in closure(\mathcal{A})$. In section III, we define a set of atoms for the decentralized control and communication problem.

III. A COMMUNICATION PROTOCOL FOR DECENTRALIZED CONTROL

The overall system architecture is shown in figure 2. Our primary interest is the construction of the communication protocol ϕ_{ij}, all the places where supervisor i sends information to supervisor j. The final output from the procedures presented in this paper consists of *all* the sequences in the plant language that could be followed by a communication event, thereby allowing the decentralized supervisors to correctly solve the control problem. Note that we make no assumptions as to the format of a communication event. For example, a communication event could be a recently observed event, state estimates, or a new symbol entirely. The positioning of the communication event simply reserves a place where useful information may be sent and received.

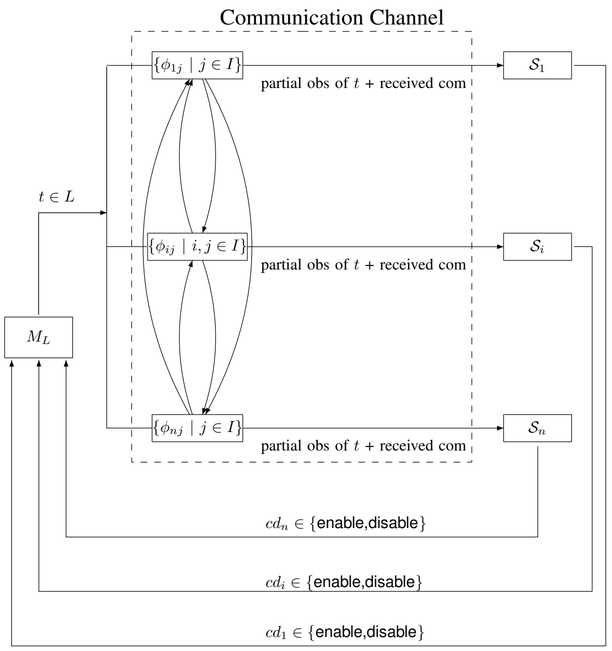
Communication Channel
$\{\phi_{1j} \mid j \in I\}$
$\{\phi_{ij} \mid i, j \in I\}$
$\{\phi_{nj} \mid j \in I\}$
partial obs of t + received com
partial obs of t + received com
partial obs of t + received com
$\mathcal{S}_1$
$\mathcal{S}_i$
$\mathcal{S}_n$
$t \in L$
M_L
$cd_n \in$ {enable,disable}
$cd_i \in$ {enable,disable}
$cd_1 \in$ {enable,disable}

Fig. 2. Decentralized architecture for communication and control, where decentralized supervisors $\mathcal{S}_i$ (for $i \in I$) make control decisions to either enable or disable σ after observing sequence t and receiving communication from other supervisors.

For the remainder of the paper, we use the notation $u\sigma$ to refer a representative illegal sequence, $v\sigma$ as a legal sequence, and assume that for all $i \in I_\sigma$, $P_i(u) = P_i(v)$.

When a system satisfies observability, then there must be at least one observable event that allows a centralized supervisor to distinguish $u\sigma$ from $v\sigma$ (i.e., $P(u) \neq P(v)$). When communication between decentralized supervisors occurs at such a place, the new information allows a formerly-confused supervisor to make the correct control decision about σ. We use the observability of K to identify all prefixes of u or v where they do differ with respect to a centralized observation. We can identify places where u and v differ by computing the longest common observable prefix, denoted here by $P(u) \sqcap P(v)$. In particular, we can define this set of prefixes or *communication sequences* along $u\sigma$:

$$u' \preceq u \text{ such that } P(u') \setminus (P(u') \sqcap P(v)) \neq \varepsilon.$$

Similarly, we can describe a similar set of communication sequences along the legal sequence $v\sigma$:

$$v' \preceq v \text{ such that } P(v') \setminus (P(v') \sqcap P(u)) \neq \varepsilon.$$

For example, suppose that $u = acaccaacccaaccaaccb$ and $v = acaccaacaaccaab$ and we use the controllable and observable event sets from the example in figure 1. If we choose $u' = acaccaaccc$ and want to know if u' is a communication sequence, we calculate $P(acaccaaccc) \setminus (P(acaccaaccc) \sqcap P(acaccaacaaccaab)) = acaccaaccc \setminus acaccaac = cc \neq \varepsilon$. The complete set of communication sequences along u is

$$\{acaccaacccaaccaacc, acaccaacccaaccaac,$$
$$acaccaacccaacc, acaccaacccaac,$$
$$acaccaaccc, acaccaacc\}.$$

We also calculate the set of communication sequences along v: $\{acaccaacaac, acaccaacaacc\}$.

We describe the components that are required to construct our communication protocol.

1) A finite structure U_σ, such that $L(U_\sigma)$ is a language on the alphabet of labels, that encodes all sequences $u\sigma$ and $v\sigma$ that violate co-observability.
2) A structure that, given two sequences s and t as input, will output their longest common observable prefix, $P(s) \sqcap P(t)$.
3) A finite structure that, given a sequence s as input, will output prefixes s' of s, i.e., $s' \preceq s$, such that s' ends with an event observable to the communicating supervisor and *not* observable to the supervisor making the control decision about σ.

A. A finite automaton to detect violations of co-observability

The automaton described here encodes all legal and illegal sequences that end in event σ and that violate (1), i.e., co-observability. This is not to be confused with the non-deterministic automaton described in [11], which generates the complete sublanguage of L that is not co-observable.

We begin with our regular languages L and K. It can be assumed, w.l.o.g. that M_L and M_K have identical transition relations and state structures. Hence M_L and M_K may differ only in their final state sets: $M_L = (Q, q_0, \Sigma, \delta, F_L)$, $M_K = (Q, q_0, \Sigma, \delta, F_K)$. We define a third language (and its corresponding automaton representation) that accepts only illegal sequences: $\mathcal{B} = L \setminus K$ and $M_\mathcal{B} = (Q, q_0, \Sigma, \delta, F_\mathcal{B})$, where $F_\mathcal{B} = F_L \setminus F_K$.

Our goal is to isolate sequences that violate co-observability wrt event $\sigma \in \Sigma_c$. So we must further refine K and $\mathcal{B}$. Let $\mathcal{B}_\sigma = \{s \mid s\sigma \in \mathcal{B}\}$ be the language of sequences

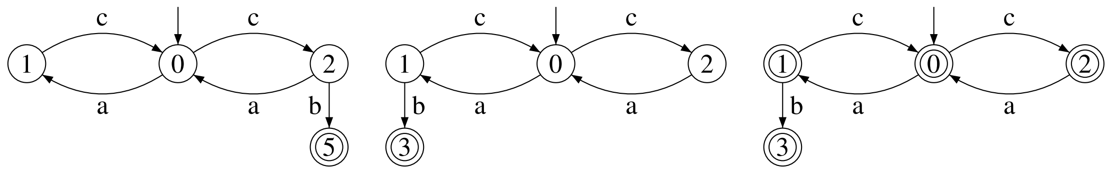

Fig. 3. The automata $M_{\mathcal{B}_\sigma}$, $M_{\mathcal{G}_\sigma}$, and $M_{\mathcal{H}_\sigma}$ for M_L and M_K in figure 1. Final states are denoted by double circles. Selfloops of ε at each state are omitted for readability.

that are illegal and end with σ. The corresponding automaton is $M_{\mathcal{B}_\sigma} = (Q, q_0, \Sigma, \delta, F_{\mathcal{B}_\sigma})$, where $F_{\mathcal{B}_\sigma} = \{q \mid \delta(q, \sigma) \in F_{\mathcal{B}}\}$. Similarly, we define a language of legal sequences that end with σ, along with its automaton representation: $\mathcal{G}_\sigma = \{s \mid s\sigma \in K\}$ and $M_{\mathcal{G}_\sigma} = (Q, q_0, \Sigma, \delta, F_{\mathcal{G}_\sigma})$, where $F_{\mathcal{G}_\sigma} = \{q \mid \delta(q, \sigma) \in F_K\}$. Finally, we want the prefix closure of $\mathcal{G}_\sigma$, which we denote by $\mathcal{H}_\sigma$. Its automaton representation is $M_{\mathcal{H}_\sigma} = (Q, q_0, \Sigma, \delta, F_{\mathcal{H}_\sigma})$, where $F_{\mathcal{H}_\sigma}$ is the least set containing $F_{\mathcal{G}_\sigma}$ such that $\forall q, \forall \alpha, \delta(q, \alpha) \in F_{\mathcal{G}_\sigma} \Rightarrow q \in F_{\mathcal{G}_\sigma}$.

To facilitate the synchronization required for the partially-observed events in K, we will augment $M_{\mathcal{B}_\sigma}$, $M_{\mathcal{G}_\sigma}$ and $M_{\mathcal{H}_\sigma}$ by adding the empty string ε to Σ and adding ε self-loops of at each state (e.g., for all $q \in Q$). The three automata, defined for the plant in figure 1, are shown in figure 3: $M_{\mathcal{B}_\sigma}$ (illegal sequences that end in σ), $M_{\mathcal{G}_\sigma}$ (legal sequences that end in σ), and $M_{\mathcal{H}_\sigma}$ (prefixes of all legal sequences that end in σ).

We build an automaton, denoted U_σ, to identify pairs of sequences that violate co-observability, by composing $M_{\mathcal{B}_\sigma}$, $|I_\sigma|$ copies of the $M_{\mathcal{G}_\sigma}$ automaton (one for each $i \in I_\sigma$) and $(n - |I_\sigma|)$ copies of the $M_{\mathcal{H}_\sigma}$ automaton (one for each $j \notin I_\sigma$). The state set of U_σ is simply the Cartesian product of the states of the component automata. The transition labels, though, are defined by synchronization labels, as discussed in section II-B. The composition can be defined in two steps: (1) the Cartesian product of the transition relations; and (2) the restriction of the resulting transition relation to those transitions with labels in $closure(\mathcal{A})$.

The set of atoms, $\mathcal{A}$, which we use to calculate the transition labels for U_σ, is defined as the union of the following two label sets.

The first atom represents transition labels for unobservable events in Σ for each supervisor: $a^{uo}_{i,e} : I_0 \to (\Sigma \setminus \Sigma_{i,o}) \cup \{\varepsilon\}$ where for $i \in I$, $j \in I_0$ and $e \in \Sigma \setminus \Sigma_{i,o}$

$$a^{uo}_{i,e}(j) = \begin{cases} \varepsilon, & \text{when } j \neq i; \\ e, & \text{when } j = i. \end{cases}$$

For example, event c is unobservable to supervisor 1 and we have $a^{uo}_{1,c} = (\varepsilon, c, \varepsilon)$, that is: $a^{uo}_{1,c}(0) = \varepsilon, a^{uo}_{1,c}(1) = c$, and $a^{uo}_{1,c}(2) = \varepsilon$.

The second atom represents transition labels for observable events in the Σ: $a^o_e : I_0 \to \Sigma_o \cup \{\varepsilon\}$ where

$$a^o_e(i) = \begin{cases} e, & \text{if } i = 0; \\ e, & \text{if } i \neq 0 \text{ and } e \in \Sigma_{i,o}; \\ \varepsilon, & \text{otherwise.} \end{cases}$$

For example, event c is observable by supervisor 2 and we have $a^o_c = (c, \varepsilon, c)$, that is: $a^o_c(0) = c, a^o_c(1) = \varepsilon$, and $a^o_c(2) = c$.

Therefore, $\mathcal{A} = \bigcup_{i \in I}(\{a^{uo}_{i,e}\}_{e \in \Sigma \setminus \Sigma_{i,o}}) \cup \{a^o_e\}_{e \in \Sigma}$.

The set of atoms for U_σ constructed from the three automata in fig. 3: $\mathcal{A} = \{(\varepsilon, c, \varepsilon), (\varepsilon, \varepsilon, a), (a, a, \varepsilon), (b, b, b), (c, \varepsilon, c)\}$.

The alphabet of U_σ is the closure of $\mathcal{A}$. For notational convenience, we will denote $closure(\mathcal{A})$ by $\mathbb{E}$. For the set of atoms noted above: $\mathbb{E} = \{(\varepsilon, \varepsilon, \varepsilon), (\varepsilon, c, \varepsilon), (\varepsilon, \varepsilon, a), (a, a, \varepsilon), (b, b, b), (c, \varepsilon, c), (c, c, c), (\varepsilon, c, a), (a, a, a)\}$.

Now we can construct automaton U_σ as follows:

$$U_\sigma = (Q^{U_\sigma}, (q_0)^{n+1}, \mathbb{E}, \delta^{U_\sigma}, F_{U_\sigma}),$$

where $Q^{U_\sigma} \subseteq Q_{\mathcal{B}_\sigma} \times (Q_{\mathcal{G}_\sigma})^{|I_\sigma|} \times (Q_{\mathcal{H}_\sigma})^{n-|I_\sigma|}$; $F_{U_\sigma} \subseteq F_{\mathcal{B}_\sigma} \times (F_{\mathcal{G}_\sigma})^{|I_\sigma|} \times (F_{\mathcal{H}_\sigma})^{n-|I_\sigma|}$; and the transition function is defined according to: $\delta^{U_\sigma}(q, \ell) = q'$ iff $\ell \in \mathbb{E}$ and $\forall i \in I_0$, $\delta(q(i), \ell(i)) = q'(i)$.

B. A non-deterministic finite transducer to find particular prefixes of a sequence

Given a sequence s, we would like to be able to find all of its prefixes that end with an event observable to communicating supervisor j, but not observable to supervisor i, the recipient of the communication. A non-deterministic automaton, called a finite transducer, X, to find prefixes s' of a sequence s such that $s' = s''\gamma$, where $\gamma \in \Sigma_{j,o} \setminus \Sigma_{i,o}$, is shown in figure 4. A transducer is an automaton M with the addition of an output alphabet Γ and an updated transition function. Here we define Γ to be $\Sigma^\varepsilon := \Sigma \cup \{\varepsilon\}$.

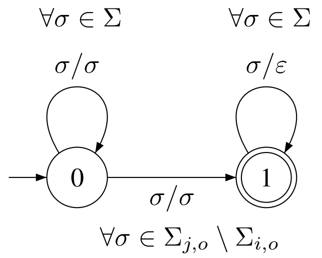
$\forall\sigma \in \Sigma$
$\forall\sigma \in \Sigma$
σ/σ
σ/ε
0
1
σ/σ
$\forall\sigma \in \Sigma_{j,o} \setminus \Sigma_{i,o}$

Fig. 4. A nondeterministic finite transducer to generate specific prefixes of a sequence.

C. An infinite transducer to find the longest common prefix of two sequences

We must first calculate $P(u') \sqcap P(v)$. We can do this by constructing an infinite deterministic transducer.

We will form a transducer T with an input alphabet of $\Sigma^\varepsilon \times \Sigma^\varepsilon$, an output alphabet of Σ^ε, a state set $Q = \{\{Left, Right\} \times \Sigma^+\} \cup \{\top, \bot\}$, where the initial state is $\top$ and the sink state is $\bot$. The set of transition rules is defined as follows: for $e, f \in \Sigma, s \in \Sigma^*$

1. $\top \xrightarrow{(e,\varepsilon)/\varepsilon} (Left, e)$
2. $\top \xrightarrow{(\varepsilon,e)/\varepsilon} (Right, e)$
3. $\top \xrightarrow{(e,e)/e} \top$
4. $\top \xrightarrow{(e,f)/\varepsilon} \bot,\ (e \neq f)$
5. $\bot \xrightarrow{(-,-)/\varepsilon} \bot$, where '-' represents any letter in Σ^ε
6. $(Left, f) \xrightarrow{(\varepsilon,f)/f} \top$
7. $(Right, e) \xrightarrow{(e,\varepsilon)/e} \top$
8. $(Left, s) \xrightarrow{(e,\varepsilon)/\varepsilon} (Left, se)$
9. $(Left, fs) \xrightarrow{(e,f)/f} (Left, se)$
10. $(Left, fs) \xrightarrow{(\varepsilon,f)/f} (Left, s),\ (s \neq \varepsilon)$
11. $(Left, fs) \xrightarrow{(\varepsilon,g)/\varepsilon} \bot,\ (f \neq g)$
12. $(Left, fs) \xrightarrow{(e,g)/\varepsilon} \bot$
13. $(Right, s) \xrightarrow{(\varepsilon,f)/\varepsilon} (Right, sf),\ (s \neq \varepsilon)$
14. $(Right, es) \xrightarrow{(e,f)/e} (Right, sf)$
15. $(Right, es) \xrightarrow{(e,\varepsilon)/e} (Right, s),\ (s \neq \varepsilon)$
16. $(Right, es) \xrightarrow{(g,\varepsilon)/\varepsilon} \bot,\ (g \neq e)$
17. $(Right, es) \xrightarrow{(g,f)/\varepsilon} \bot$
18. $q \xrightarrow{(\varepsilon,\varepsilon)/\varepsilon} q,\ q \in Q$

Suppose that we have input sequences $u = accb$ and $v = a\varepsilon\varepsilon b$, as taken from the sequence (a, a, ε) (c, ε, c) $(\varepsilon, \varepsilon, a)$ (c, ε, c) $(\varepsilon, \varepsilon, a)$ $(\varepsilon, \varepsilon, a)$ (b, b, b) in $L(U_\sigma)$, where we ignore the contributions of the sequence observed by supervisor 2 and, for the sake of simplicity, omit $(\varepsilon, \varepsilon)$ pairs from the illegal sequence u and the sequence observed by supervisor 1. Then we have the following path that defines the largest common prefix, a, of these two sequences (by applying, in order, rules 3,1,9,12): $\top \xrightarrow{(a,a)/a} \top \xrightarrow{(c,\varepsilon)/\varepsilon} (Left, c) \xrightarrow{(c,\varepsilon)/\varepsilon} (Left, cc) \xrightarrow{(b,b)/\varepsilon} \bot$

D. An infinite mind-the-gap automaton

Figure 5 provides a graphical interpretation of the interaction of the three components we define for building a communication protocol. The language of U_σ defines sequences of labels, where each label has $n+1$ components. What we are actually interested in are pairs of sequences whose canonical projections, P_i, are equal. By construction, for a given path in U_σ, the sequence of 0^{th} elements of the labels defines u. Similarly, we can isolate v by examining the sequence of the i^{th} elements, where $i \in \{1..|I_\sigma|\}$. The difficulty with the output from U_σ is that the input to transducer T is not synchronized. That is, when T is processing pairs of letters, T has to perform the matching of the two sequences. It is possible that T will require unbounded memory to store all the observations seen prior to the matching of two events. For example, suppose that we expand M_L (and M_K) in figure 1 so that we introduce a new initial state $0'$ and add two new transitions $\delta(0', c) = 0'$ and $\delta(0', b) = 0$. The resulting effect is an infinite number of states. This is clearly not desirable for the effective synthesis of a communication protocol, and we must somehow restrict

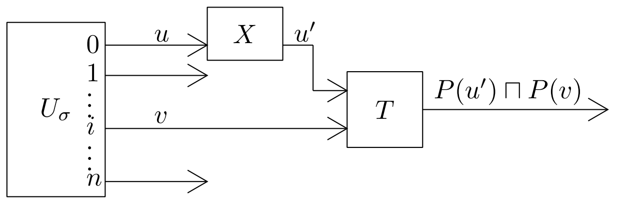

Fig. 5. Building a communication protocol wrt u. The output from T, when subtracted from u', the output of X, is a possible communication sequence (along u) that would allow sequences u and v to be distinguished. A similar diagram, with v as input to X and u and v' as input to T can be constructed for a communication protocol wrt v.

U_σ so that it generates a relation that will reach only finitely many states of T.

IV. A FINITE MIND-THE-GAP AUTOMATON

It will be useful to prove certain properties of U_σ so that we can perform the appropriate transformation on the input to T. In particular, we are interested in the step property of step transition systems [13] and their relationship to Mazurkiewicz trace languages [7] and, more precisely, generalized trace languages [6].

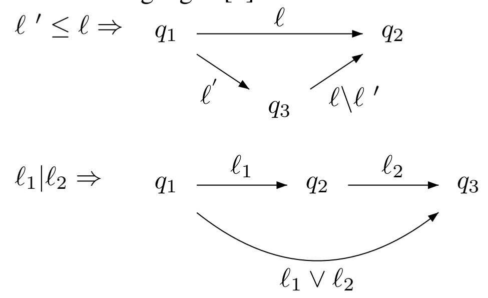

Fig. 6. The step property of deterministic transition systems.

Definition 1: (Adapted from [13]) A deterministic transition system (Q, q_0, δ) satisfies the *step property* if it ensures that transitions with non-atomic labels can be decomposed into arbitrary sequences of decompositions of the label. Formally, $\forall q_1, q_2 \in Q, \forall \ell, \ell' \in \mathbb{E}$,

$$\delta(q_1, \ell) = q_2 \text{ and } \ell' \leq \ell \Rightarrow \exists q_3$$
$$\text{such that } \delta(q_1, \ell') = q_3 \text{ and } \delta(q_3, \ell \backslash \ell') = q_2.$$

Independent transitions originating from the same state may be merged: $\forall q_1, q_2, q_3 \in Q, \forall \ell_1, \ell_2 \in \mathbb{E}$,

$$\delta(q_1, \ell_1) = q_2 \text{ and } \delta(q_2, \ell_2) = q_3 \text{ and}$$
$$\ell_1 | \ell_2 \Rightarrow \delta(q_1, \ell_1 \vee \ell_2) = q_3. \ \Box$$

Figure 6 illustrates the step property.

The language of a transition system with the step property is a generalized trace language [6], generalizing Mazurkiewicz trace languages [7], and defined as follows.

Let $\sim$ be the least congruence on $\mathbb{E}^*$ such that: $\forall a, b \in \mathbb{E}^*$ $a|b \Rightarrow ab \sim ba$ and $a \leq b \Rightarrow b \sim a \cdot (b \backslash a)$. Let $\simeq$ be the least congruence on $\mathbb{E}^*$ such that $\forall a, b \in \mathbb{E}$, $a \mid b \Rightarrow ab \simeq ba$.

Definition 2: (Adapted from [7], [6]) $\mathcal{L} \subseteq \mathbb{E}^*$ is a trace language iff it is closed under $\simeq$. $\mathcal{L} \subseteq \mathbb{E}^*$ is a generalized trace language iff it is closed under $\sim$: $\forall u \in \mathcal{L}, \forall v \in \mathbb{E}^*$ $u \sim v \Rightarrow v \in \mathcal{L}$. $\Box$

It is straightforward to check that U_σ has the step property and its language is a generalized trace language, with relation $|$ as the independence relation.

It is the case that the language of a step transition system is a Mazurkiewicz trace language, whenever labels are decomposed into arbitrary sequences of atoms, for which we can compute a normal form called Cartier-Foata Normal Form (CFNF) [3].

Definition 3: A word $u = \ell_1 \dots \ell_n \in \mathcal{A}$ is in CFNF iff $\forall i = 1 \dots n-1, \forall \ell' \in \mathcal{A},\ \ell' \not\leq \ell_{i+1}$ or not$(\ell_i \mid \ell')$. □

Intuitively, words in CFNF correspond to greedy behaviors of the system, where no label belonging to step ℓ_{i+1} can be moved into step ℓ_i.

When the language of U_σ is in CFNF, the effect is to have performed a pruning of transitions in U_σ so that any sequence of the pruned U_σ will be in CFNF. Note that this pruning depends only on the elements in the synchronized alphabet, and not on the state of the plant. We will show why this leads to a bound on the memory for T.

Since $L(U_\sigma)$ is a generalized trace language, we can restrict U_σ so that its language is in CFNF. This effectively breaks cycles in U_σ that could cause divergence of the size of reachable states of T and leaves only states that require a bounded memory. This implies that only finitely many states of T remain reachable when U_σ is restricted, as defined above.

To compute the CFNF of U_σ we need to compute one further automaton, N, based only on labels: $N = (Q^N, q_0^N, \mathbb{E}, \delta^N)$, where $Q^N = 2^{I_0}$; $q_0^N = I_0$; and the transition function $\delta \subseteq Q^N \times \mathbb{E} \times Q^N$ is defined as follows. A transition $\delta^N(q, \ell) = q'$ iff $\forall \ell' \in \mathcal{A},\ \ell' \leq \ell \Rightarrow (||\ell'|| \cap q \neq \emptyset)$ and $q' = ||\ell||$.

The product of U^σ and N is the restriction of U_σ, denoted by $\widehat{U}_\sigma$.

Theorem 1: $\widehat{U}_\sigma \times T$ has finitely many reachable states.

Proof: Recall that in U_σ observable events are synchronized. Therefore, states of the form (q, μ, u), where $\mu \in \{Left, Right\}$ and u contains observable events for the communicating supervisor, are not reachable in $\widehat{U}_\sigma \times T$.

Assume that there exists an infinite path λ in $\widehat{U}_\sigma \times T$ causing divergence of state size. $\widehat{U}_\sigma$ is finite state, therefore there exists a state q in $\widehat{U}_\sigma$ that is visited infinitely often by λ. Thus λ can be decomposed as $\lambda = \delta\gamma\lambda'$ such that

$$(q_0, \top) \overset{\delta}{\rightarrow^*} (q, \mu, u_0) \overset{\gamma}{\rightarrow^*} (q, \mu, u_1)$$

where $|u_0| < |u_1|$ and $\mu \in \{Left, Right\}$ and cycle γ can be iterated to form an ultimately periodic path $\delta\gamma^\omega$ in $\widehat{U}_\sigma \times T$ that also causes divergence of state sizes. This means that

$$(q_0, \top) \overset{\delta}{\rightarrow^*} (q, \mu, u_0) \overset{\gamma}{\rightarrow^*} (q, \mu, u_1) \overset{\gamma}{\rightarrow^*} (q, \mu, u_2) \dots$$

where $|u_0| < |u_1| < \dots < |u_n|$, $\mu \in \{Left, Right\}$ and $\forall i \geq 0$, u_i is unobservable.

By symmetry and wlog, assume that $\mu = Left$. Define a projection π_i that returns the i^{th} component of γ. The inequality $|u_0| < |u_1|$ implies that

$$|\pi_0(\gamma)| > |\pi_1(\gamma)|. \qquad (3)$$

Wlog assume that $|u_0| > |\pi_0(\gamma)|$. This implies that γ may not contain any observable events wrt the communicating supervisor. Thus, $\pi_1(\gamma)$ and $\pi_0(\gamma)$ are unobservable wrt the communicating supervisor. Inequality (3) implies that the last vector of γ commutes with one atom of the first vector of γ. Thus $\gamma\cdot\gamma$ is not in CFNF. This contradicts the assumption that every path in $\widehat{U}_\sigma$ is in CFNF. Hence no ultimately periodic path of $\widehat{U}_\sigma$ causes state size divergence. Therefore, there exists a bound on the size of states reachable in $\widehat{U}_\sigma \times T$ and $\widehat{U}_\sigma \times T$ has finitely many reachable states. ■

V. Future Work

It is clear that the communication sublanguage generated by communicating early or late is regular. While we can now calculate more than just the "first" and "last" possible places to communicate in the plant language, we should not expect that, in general, when considering intermediate possibilities for communication that we can find a procedure that will always find a regular communication sublanguage of the plant. That is, it is possible to find non-regular sublanguages of a regular language (e.g., $a^n b^n \subset \{a, b\}^*$). We are currently investigating strategies for constructing regular communication languages, that include more than just the "first" and "last" place to communicate. Additionally, in an effort to address computational complexity issues, we are pursuing more efficient implementations of the mathematical formulas presented here for the construction of our communication protocol.

References

[1] Arnold A.: Finite transition systems. Prentice-Hall, 1994

[2] Barrett, G. and Lafortune, S.: Decentralized Supervisory Control with Communicating Controllers. IEEE Trans. on Automat. Contr. **45** (9) (2000) 1620–1638

[3] Cartier, P. and Foata, D.: Problèmes combinatoires de permutations et réarrangements. Springer-Verlag, 1969 (Lecture Notes in Math., 85).

[4] Cassandras, C. and Lafortune, S. Introduction to Discrete Event Systems Kluwer (1999)

[5] Cieslak, R., Desclaux, C. and Fawaz, A.S. and Varaiya, P.: Supervisory control of discrete-event processes with partial observations. IEEE Trans. Automat. Contr. **33**(3) (1988) 249–260

[6] Hoogers, P.W., Kleijn, H.C.M. and Thragaragan, P.S.: A Trace Semantics for Petri Nets, Information and Computation **117** (1995) 98–114

[7] Mazurkiewicz, A. Concurrent program schemes and their interpretations. Aarhus University Publication (DAIMI PB-78, 1977)

[8] Ramadge, P.J. and Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Contr. Optim. **25**(1) (1987) 206–230

[9] Ramadge, P.J. and Wonham, W.M.: The control of discrete-event systems. Proc. IEEE **77**(1) (1989) 81–98

[10] Ricker, S.L. and Rudie, K.: Incorporating communication and knowledge into decentralized discrete-event systems. Proc. IEEE Conf. Decision Contr. (1999) 1326–1332

[11] Rudie, K. and Willems, J.C.: The Computational Complexity of Decentralized Discrete-Event Control Problems. IEEE Trans. Automat. Contr. **40**(7) (1995) 1313–1319

[12] Rudie, K. and Wonham, W.M.: Think Globally, Act Locally: Decentralized Supervisory Control. IEEE Trans. Automat. Contr. **37**(11) (1992) 1692–1708

[13] Stark, E.W.: Concurrent Transition Systems. Theoretical Computer Science **64**(3) (1989) 221–269

[14] Wong, K.C. and van Schuppen, J.H.: Decentralized Supervisory Control of Discrete-Event Systems with Communication Proc. Int. Workshop on Discrete Event Systems (1996) 284–289