Incremental Verification of Co-observability in Discrete-event Systems*

Huailiang Liu1, Ryan J. Leduc2, Robi Malik3 and S. L. Ricker4

Abstract—Existing strategies for verifying co-observability, one of the properties that must be satisfied for synthesizing solutions to decentralized supervisory control problems, require the construction of the complete system model. When the system is composed of many subsystems, these monolithic approaches may be impractical due to the state-space explosion problem. To address this issue, we introduce an incremental verification of co-observability approach. Selected subgroups of the system are evaluated individually, until verification of co-observability is complete. The new method is potentially much more efficient than the monolithic approaches, in particular for systems composed of many subsystems, allowing for some intractable state-space explosion problems to be manageable. Properties of this new strategy are presented, along with a corresponding algorithm and an example.

I. INTRODUCTION

One of the main challenges in the control of discrete-event systems (DES) is the combinatorial explosion of the state space. The state-space explosion problem becomes a bottleneck for the application of supervisory control of DES. To address the state-space explosion problem, an incremental method has been successfully used in verification of controllability without considering nonblocking [3]. Nonblocking verification has been addressed using the compositional approach [12], [6], in which the global system is constructed incrementally using abstraction in order to reduce the complexity of verification. A different type of abstraction appears in [7], [11] based on supervision equivalence to incrementally construct the monolithic supervisor for a system.

All the above literature using incremental methods assume that a supervisor has full observation, and none of them consider the setting of decentralized control [15], in which each supervisor can access only partial information and is allowed to disable only a subset of controllable events. The supervisors must coordinate the disabling and enabling of events in order to realize the legal or desired behavior.

Decentralized discrete-event control problems arise naturally through the investigation of a large variety of distributed systems such as complex automated manufacturing systems, communication networks, integrated sensor networks, networked control systems and automated guided vehicular system. These systems have many supervisors (also called controllers, agents or observers) that jointly control a given system which is inherently distributed.

Due to the distributed nature of the system, in decentralized control, supervisors at different sites typically can only observe some of the events and can only control some of the controllable events. In this situation, traditional monolithic control is not appropriate and it is possible to use decentralized control when the system satisfies a property called co-observability [15]. Existing work on decentralized supervisory control of DES focuses on problems where decentralized controllers each control and observe some events in a system and must together achieve some prescribed goal. The synthesis of decentralized supervisors requires that the specification satisfies the co-observability property. When the system is very large and composed of many subsystems, verifying co-observability using existing monolithic methods requires the construction of the complete system model which may be intractable in practice due to the state-space explosion problem.

To address this problem, we introduce a new approach called incremental verification of co-observability. Using this method, verifying co-observability is done incrementally by evaluating selected subgroups of the system individually, until the entire system has been shown to be co-observable. The incremental selection of suitable subgroups is guided by counter examples defined in Section IV. Compared to traditional monolithic methods, the new method is potentially more efficient for very large systems composed of many subsystems, rendering some intractable state-space explosion problems to be manageable.

This paper is organized as follows. Section II reviews supervisory control of DES. Section III provides details for the incremental verification of co-observability. Section IV presents an algorithm for incremental verification of co-observability guided by counter examples. Section V applies this algorithm to verify a classical communication protocol. Finally, Section VI provides conclusions and future work.

Formal proofs of technical results can be found in [10].

II. PRELIMINARIES

This section provides a brief review of the key concepts used in this paper. Readers unfamiliar with the notation and definitions may refer to [4].

Event sequences and languages are simple ways to describe DES behaviors. Let \( \Sigma \) be a finite set of distinct symbols (events), and \( \Sigma^* \) be the set of all finite sequences...
of events plus $\epsilon$, the empty string. A language $L$ over $\Sigma$ is any subset $L \subseteq \Sigma^*$. The concatenation of two strings $s, t \in \Sigma^*$, is written as $st$. Languages and alphabets can also be concatenated as: $L \cdot o := \{sa \in \Sigma^* \mid s \in L, a \in \Sigma\}$.

For strings $s, t \in \Sigma^*$, we say that $t$ is a prefix of $s$ (written $t \leq s$) if $s = tu$, for some $u \in \Sigma^*$. In this case, we also say that $t$ can be extended to $s$.

The prefix closure $\mathcal{T}$ of a language $L \subseteq \Sigma^*$ is defined as follows:

$$\mathcal{T} := \{t \in \Sigma^* \mid t \leq s \text{ for some } s \in L\}.$$ 

A language $L$ is said to be prefix-closed if $L = \mathcal{T}$.

Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $L_1 \subseteq \Sigma_1^*$, and $L_2 \subseteq \Sigma_2^*$. For $i \in \{1, 2\}$, $s \in \Sigma_i^*$, and $\sigma \in \Sigma$, to capture the notion of partial observation, we define the natural projection $P_i : \Sigma^* \to \Sigma_i^*$ according to:

$$P_i(\epsilon) := \epsilon,$$

$$P_i(\sigma) := \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_i; \\ \epsilon, & \text{otherwise}. \end{cases}$$

$$P_i(s\sigma) := P_i(s)P_i(\sigma)$$

For convenience in the above definition, we only define for $i \in \{1, 2\}$. In fact, projection works for any subset of $\Sigma^*$.

The inverse projection $P_i^{-1}$ is the mapping $P_{wr}(\Sigma_i^*) \to P_{wr}(\Sigma^*)$ defined on sets of strings (or languages), where $P_{wr}(\Sigma_i^*)$ and $P_{wr}(\Sigma^*)$ denote all subsets of $\Sigma_i^*$ and $\Sigma^*$ respectively. Given any $L_i \subseteq \Sigma_i^*$, the inverse projection $P_i^{-1}(L_i)$ is defined as: $P_i^{-1}(L_i) := \{s \in \Sigma^* \mid P_i(s) \in L_i\}$.

A DES automaton is represented as a tuple: $G := (Q, \Sigma, \delta, q_0)$, with finite state set $Q$, finite alphabet set $\Sigma$, partial transition function $\delta : Q \times \Sigma \to Q$ and initial state $q_0$. $\delta$ is defined at each state $q \in Q$ for some of the events $\sigma \in \Sigma$. We use $\delta(q, \sigma)!$ to represent that $\delta$ is defined for $\sigma \in \Sigma$ at state $q$. $\delta$ can be extended to $\Sigma^*$ by defining $\delta(q, \epsilon) := q$ and $\delta(q, \sigma) := \delta(\delta(q, \sigma), \sigma)$, provided that $q' = \delta(q, \sigma)!$ and $\delta(q', \sigma)!$.

The closed behavior for a DES $G$ is denoted by a regular language $L(G)$, and is defined to be: $L(G) := \{s \in \Sigma^* \mid \exists q \in Q, \delta(q, \sigma)! \}$! The reachable state subset of $G$, denoted as $Q_r$, is defined as: $Q_r := \{q \in Q \mid (\exists s \in \Sigma^*) \delta(q_0, s) = q\}$.

We say that $G$ is reachable if $Q_r = Q$.

Let $G_1 := (Q_1, \Sigma_1, \delta_1, q_{01})$ and $G_2 := (Q_2, \Sigma_2, \delta_2, q_{02})$ be two automata. Their synchronous product is a DES $G$ over event set $\Sigma = \Sigma_1 \cup \Sigma_2$, $G := G_1 | G_2 := (Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}))$. For $(q_1, q_2) \in Q_1 \times Q_2$, we define $\delta((q_1, q_2), \sigma)$ to be:

$$\begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)), & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \\ (\delta_1(q_1, \sigma), q_2), & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \\ (q_1, \delta_2(q_2, \sigma)), & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2. \end{cases}$$

The synchronous product of languages $L_1$ and $L_2$, denoted by $L_1 | L_2$, is defined to be: $L_1 | L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_1 \cap L_2)$. If both $L_1$ and $L_2$ are over the same event set $\Sigma$, then they have the following property: $L = L_1 | L_2 = P_1^{-1}(L_1) \cap P_2^{-1}(L_2) = L_1 \cap L_2$. In this paper, for convenience, we assume the synchronous product is defined over the same event set $\Sigma$. If we are given an automaton defined over a subset of $\Sigma$, we can simply self-loop the missing events at each state.

In supervisory control, the event set $\Sigma$ is partitioned into two disjoint sets: the controllable event set $\Sigma_c$ and the uncontrollable event set $\Sigma_{uc}$. Controllable events can be prevented from happening, or disabled, by a supervisor, while uncontrollable events cannot.

Let $K$ and $L = \mathcal{T}$ be languages over event set $\Sigma$, and $\Sigma_{uc} \subseteq \Sigma$ be the uncontrollable event set. $K$ is said to be controllable with respect to $L$ and $\Sigma_{uc}$ if, $\mathcal{K}_{\Sigma_{uc}} \cap L \subseteq \mathcal{K}$.

The property of co-observability was introduced in [15], and a verification algorithm was introduced in [14]. The following is the definition of co-observability adapted from [15], [1].

Definition 1: Let $K$, $L = \mathcal{T}$ be languages over event set $\Sigma$. Let $I = \{1, \ldots, n\}$ be an index set. Let $\Sigma_{i,1} \subseteq \Sigma$ and $\Sigma_{i,2} \subseteq \Sigma$ be sets of controllable and observable events, respectively, for $i \in I$, where $\Sigma_{c} = \bigcup_{i=1}^{n} \Sigma_{i,1}$ and $I_c(\sigma) := \{i \in I \mid \sigma \in \Sigma_{i,1}\}$. Let $P_i : \Sigma^* \to \Sigma_{i,2}$ be a natural projection. A language $K$ is said to be co-observable with respect to $L$, $\Sigma_{i,1}$, $\Sigma_{i,2}$, $i \in I$, if,

$$\forall t \in \mathcal{K} \cap L \cap I \frac{\forall \sigma}{\exists i \in I_c(\sigma)} P_i^{-1}(P_i(t)) \cap \mathcal{K} \cap L = \emptyset.$$ 

Notice that in definition 1, when there is only one controller (supervisor), $I = \{1\}$, the property is called observability [9]. To apply the definition to languages represented by plant automaton $G$ and specification (requirement) automaton $S$, we use $L = L(G)$ and $K = L(S)$.

Since in practice the specification $K$ is not necessarily a subset of $L$, we do not require that $K \subseteq L$ as in the original definition. Instead of checking all strings in $\mathcal{K}$, reasonably, we check all strings in $\mathcal{K} \cap L$. This makes co-observability easier to apply in an incremental algorithm.

To solve the control problem, decentralized controllers take local control decisions based on their partial observations. When the system leaves $\mathcal{K}$ (i.e., $t \sigma \in L \setminus \mathcal{K}$, where $t \sigma \in L$ and $t \sigma \notin \mathcal{K}$) there must be at least one controller (i.e., $\exists i \in I_c(\sigma)$) that has sufficient information from its own view of the system to take the correct control decision (i.e., disable $\sigma$). Note that, by default, a controller $i \in I$ will enable all events $\sigma \in \Sigma \setminus \Sigma_{c,i}$.

If an event $\sigma$ needs to be disabled (i.e., $t \sigma \in \mathcal{K}$, $t \sigma \in L \setminus \mathcal{K}$), then at least one of the controllers that control $\sigma$ must unambiguously know that $\sigma$ must be disabled (i.e., $P_i^{-1}(P_i(t)) \cap \mathcal{K} \cap L = \emptyset$). From this controller’s viewpoint, disabling $\sigma$ does not prevent any string in $\mathcal{K} \cap L$. For all other controllers that are uncertain about whether they should disable the event $\sigma$, they will enable the event $\sigma$, and the final fusion rule used here is the conjunction of all the decisions of controllers [15].

In the following, when there is no ambiguity, instead of saying that $K$ is co-observable with respect to $L$, $\Sigma_{i,1}$, $\Sigma_{c,i}$, $i \in I$, we will say that $K$ is co-observable w.r.t. $L$. 
III. INCREMENTAL VERIFICATION OF CO-OBSERVABILITY

In practice, the synchronous product of the plant \( L = L_1 \| \cdots \| L_m \) cannot affect whether Definition 2 is satisfied or not. As such, these \( t_i \) in \( C \) can be safely ignored.

Recall that if languages are over the same event set \( \Sigma \), then the above synchronous product of \( L \) and \( K \) have the following property: \( L = L_1 || \cdots || L_m = L_1 \cap \cdots \cap L_m \). \( K = K_1 || \cdots || K_r = K_1 \cap \cdots \cap K_r \). This is why we use the intersection of the languages in this section.

**Proposition 1:** Let \( K, L = L, M = L \) be languages over event set \( \Sigma \). If \( K \) is co-observable with respect to \( M \) and \( L \subseteq M \), then \( K \) is co-observable with respect to \( L \).

**Proposition 2:** Let \( K_1, K_2 \) and \( L \) be prefix-closed languages over event set \( \Sigma \). If both \( K_1 \) and \( K_2 \) are co-observable with respect to \( L \), then \( K = K_1 \cap K_2 \) is co-observable with respect to \( L \).

**Proposition 3:** Let \( K_1, K_2, M_1, \) and \( M_2 \) be prefix-closed languages over event set \( \Sigma \). If \( K_1 \) is co-observable with respect to \( M_1 \), and \( K_2 \) is co-observable with respect to \( M_2 \), then \( K = K_1 \cap K_2 \) is co-observable with respect to \( M = M_1 \cap M_2 \).

**Proposition 4:** Let \( K, L \) be prefix-closed languages over event set \( \Sigma \). If \( K \supseteq L \) then \( K \) is co-observable w.r.t. \( L \).

**Proposition 5:** Let \( K_1, K_2 \) and \( M \) be prefix-closed languages over event set \( \Sigma \). If \( K_1 \) is co-observable w.r.t. \( M \cap K_2 \), and \( K_2 \) is co-observable w.r.t. \( M \), then \( K = K_1 \cap K_2 \) is co-observable w.r.t. \( M \).

**Proposition 6:** Let \( K_1, K_2, M_1 \) and \( M_2 \) be prefix-closed languages over event set \( \Sigma \). If \( K_1 \) is co-observable w.r.t. \( M_1 \cap K_2 \), and \( K_2 \) is co-observable w.r.t. \( M_2 \), then \( K = K_1 \cap K_2 \) is co-observable w.r.t. \( M = M_1 \cap M_2 \).

**Proposition 7:** Let \( K_1, K_2 \) and \( M \) be prefix-closed languages over event set \( \Sigma \). If \( K_1 \) is not co-observable w.r.t. \( M \cap K_2 \), then \( K = K_1 \cap K_2 \) is not co-observable w.r.t. \( M \).

**Algorithm**

In this section, we give an algorithm on how to do incremental verification of co-observability guided by counter examples.

To show that the system fails to satisfy co-observability, we give the definition of a counter example for co-observability.

**Definition 2:** Let \( K, L = L \) be languages over event set \( \Sigma \). Let \( \Sigma_{c, i} \subseteq \Sigma \) and \( \Sigma_{o, i} \subseteq \Sigma \) be sets of controllable and observable events, respectively, for \( i \in I \), \( I_c (\sigma) := \{ i \in I \mid \sigma \in \Sigma_{c, i} \} \). Let \( P_i : \Sigma^* \rightarrow \Sigma_{o, i}^* \) be a natural projection. A co-observability counter example for the specification \( K \) and the plant \( L \) is a tuple \( C = (\sigma, t, t_1, ..., t_n) \) where

- \( \sigma \in \Sigma_c \);
- \( t \in K \cap L \) and \( t_\sigma \in L \setminus K \);
- \( \forall i \in I_c (\sigma) \) \( t_i \sigma \in K \cap L \);
- \( \forall i \in I_o (\sigma) \) \( P_i (t) = P_i (t_i) \).

Please note that for \( i \in I \setminus I_c (\sigma) \), the corresponding controllers cannot disable \( \sigma \), therefore these \( t_i \) in \( C \) cannot affect whether Definition 2 is satisfied or not. As such, these \( t_i \) in \( C \) can be safely ignored.
To show that a specification rejects a co-observability counter example, we give the following definition. The intention is that if we replace \( K \) in Definition 2 by \( K \cap K' \), then \( C \) will no longer be a valid counter example.

**Definition 3:** If \( C = (\sigma, t, t_1,..., t_n) \) is a co-observability counter example for the specification \( K = K \) and the plant \( L = L \), we say that specification \( K' = K' \subseteq \Sigma^* \) rejects \( C \), if:

- \( t \notin K' \), or
- \( (\exists i \in L_{\sigma}(\sigma)) \ t_i \sigma \notin K' \).

Analogously, we give the definition bellow for when a plant rejects a co-observability counter example. The intention here is that if we replace \( L \) in Definition 2 by \( L \cap L' \), then \( C \) will no longer be a valid counter example.

**Definition 4:** If \( C = (\sigma, t, t_1,..., t_n) \) is a co-observability counter example for the specification \( K = K \) and the plant \( L = L \), we say that plant \( L' = L' \subseteq \Sigma^* \) rejects \( C \), if:

- \( t \sigma \notin L' \), or
- \( (\exists i \in L_{\sigma}(\sigma)) \ t_i \sigma \notin L' \).

The plant is \( L = L_1|...|L_m \) and the specification is \( K = K_1|...|K_r \) and is co-observable w.r.t. \( L \) according to Propositions 2 and 3.

(2) If \( K' = K_1|...|K_i \) where \( \{i_1,...,i_a\} \subseteq \{1,...,r\} \) is co-observable w.r.t. \( L' = L_{j_1}|...|L_{j_b} \) where \( \{j_1,...,j_b\} \subseteq \{1,...,m\} \) then \( K' \) is co-observable w.r.t. \( L \) according to Proposition 1 and Corollary 1.

(3) If \( K_i \) where \( i \in \{1,...,r\} \), is co-observable w.r.t. a component of \( L \), then \( K_i \) is co-observable w.r.t. \( L \) according to Propositions 2 and 3.

(4) If there is a counter example which shows that \( K_i \) is not co-observable w.r.t. \( L \), then \( K_i \) can be treated as a plant to be synchronized with \( L \), according to Propositions 5 and 6.

(5) If there is a counter example which shows that \( K_i \) is not co-observable w.r.t. \( L \), then \( K_i \) can be treated as a plant to be synchronized with \( L \), according to Proposition 7.

To understand the abstract description of Algorithm 1, we need to define the relationship of the set of plants \( \mathcal{L} = \{L_1,...,L_m\} \) and the language \( L = L_1|...|L_m \) and the set of specifications \( \mathcal{K} = \{K_1,...,K_r\} \) and the language \( K = K_1|...|K_r \). The set \( \mathcal{L} \) is represented as the language \( L \) and similarly the set \( \mathcal{K} \) is represented as the language \( K \).

We can define the meaning of line 5. When we say that \( \mathcal{K}' = \{K_1,...,K_i\} \) is not co-observable w.r.t. \( \mathcal{L}' = \{L_{j_1},...,L_{j_b}\} \), we mean that is a sub-system \( K' = K_i|...|K_i \) where \( \{i_1,...,i_a\} \subseteq \{1,...,r\} \) is not co-observable w.r.t. a sub-system \( L' = L_{j_1}|...|L_{j_b} \) where \( \{j_1,...,j_b\} \subseteq \{1,...,m\} \).

In practice, the plant and specification are all represented as DES: \( L = L_1|...|L_m \) is represented as \( L(G) = L(G_1)|...|L(G_m) \), and the specification \( K = K_1|...|K_r \) is represented as \( L(S) = L(S_1)|...|L(S_m) \).

### Algorithm 1 Incremental Coobservability Verification

1. **input** plants \( \mathcal{L} = \{L_1,...,L_m\} \), specifications \( \mathcal{K} = \{K_1,...,K_r\} \);
2. **while** \( \mathcal{K} \neq \emptyset \) **do**
   3. Pick a \( K_i \in \mathcal{K} \);
   4. Let \( \mathcal{K}' = \{K_i\} \), \( \mathcal{L}' = \emptyset \);
   5. **while** \( \mathcal{K}' \) is not co-observable w.r.t. \( \mathcal{L}' \) **do**
      6. Let \( \mathcal{C} \) be a counter example showing that \( \mathcal{K}' \) is not co-observable w.r.t. \( \mathcal{L}' \);
      7. Find a component \( L_j \in \mathcal{L} \setminus \mathcal{L}' \) or \( K_h \in \mathcal{K} \setminus \mathcal{K}' \) which does not accept \( C \);
      8. **if** there is no such a component **then**
         9. **stop** \( \mathcal{K} = \{K_1,...,K_r\} \) is not co-observable w.r.t. \( \mathcal{L} = \{L_1,...,L_m\} \), counter example: \( C' \);
      10. **else** if the component found in line 7 is a plant **then**
          11. Let \( \mathcal{L}' = \mathcal{L}' \cup \{L_j\} \);
          12. **else**
              13. Let \( \mathcal{K}' = \mathcal{K}' \cup \{K_h\} \);
          14. **end if**
      15. **end while**
   16. Let \( \mathcal{K} = \mathcal{K} \setminus \mathcal{K}' \), \( \mathcal{L} = \mathcal{L} \cup \mathcal{L}' \);
17. **end while**
18. **stop** \( \mathcal{K} = \{K_1,...,K_r\} \) is co-observable w.r.t. \( \mathcal{L} = \{L_1,...,L_m\} \);

Notice that on line 4, we assign \( \mathcal{L}' = \emptyset \), which thus means that the corresponding language \( L' = \Sigma^* \). We say that \( \mathcal{L}' = \emptyset \) represents the language of the automaton for the empty set of plants. We represent \( L' = \Sigma^* \) as the language for the automaton \( G_{\Sigma^*} \), defined over \( \Sigma \), which is an automaton with only an initial state at which every event in \( \Sigma \) is self-looped. We thus have \( L(G_{\Sigma^*}) = \Sigma^* \). This means that for any automaton \( G \) defined over \( \Sigma \), \( G|G_{\Sigma^*} = G \), and thus \( L(G)|G_{\Sigma^*} = L(G) \). In other words, we initially verify whether \( K' \) is co-observable w.r.t. \( \Sigma^* \).

On line 2, if \( \mathcal{K} \) is initially empty then the specification \( K = \Sigma^* \) will be co-observable w.r.t. \( \mathcal{L} \), which is trivially true. This is because \( K = \Sigma^* \) is a superset of all languages over \( \Sigma \) and is co-observable w.r.t. every language over \( \Sigma \) according to Proposition 4.

If \( \mathcal{K} \) is not empty, then on line 3, one component \( K_i \in \mathcal{K} = \{K_1,...,K_r\} \), where \( i \in \{1,...,r\} \), will be picked to verify whether \( K_i \) is co-observable w.r.t. \( \mathcal{L} \). If each component \( K_i \) is co-observable w.r.t. \( \mathcal{L} \), then according to Propositions 2 or 3, \( K_i \) will be co-observable w.r.t. \( \mathcal{L} \). In fact, the following steps will use only one or some components of \( \mathcal{L} = \{L_1,...,L_m\} \), because if \( K_i \) is co-observable w.r.t. some components of \( \mathcal{L} \), then it will be co-observable w.r.t. \( \mathcal{L} \) according to Proposition 1 and Corollary 1.

On line 6, a counter example \( C = (\sigma,t,t_1,...,t_n) \) will be produced by the algorithm to show that \( K' \) is not co-observable w.r.t. \( L' \) according to Definition 2. If there are many counter examples, then usually the shortest one will be selected. Some other heuristics can also be used to select...
counter examples.

On line 7, a component $L_j$ in $L \setminus L'$ or $K_h$ in $K \setminus K'$ which does not accept $C$ is selected according to Definition 4 or Definition 3, respectively.

Lines 8 and 9 demonstrate that if there is no such a component, then we know that every other component accepts the counter example $C$ which shows that $K$ is not co-observable w.r.t. plant $L$ by Proposition 7.

Lines 10 and 11 incrementally add a plant component $L_j$ to $L'$ where $j \in \{1, \ldots, m\}$.

Lines 12 and 13 incrementally add a specification component $K_h$ to $K'$ where $h \in \{1, \ldots, r\}$.

If the subsystem consisting of specifications $K'$ and plants $L'$ is found to be co-observable, then line 16 removes the specifications $K'$ from $K$ and adds them to $L$, so they are treated as plants for the remainder of the algorithm, according to Proposition 5. The algorithm terminates when the set $K$ of specifications to be checked is empty in which case it asserts co-observability on line 18.

V. Incremental Verification of Co-Observability for the Sequence Transmission Protocol

In this section, we demonstrate the incremental verification of co-observability for the sequence transmission protocol which is a classical network protocol that occurs at the data link layer of the ISO OSI Reference Model [16].

A. The Sequence Transmission Problem

The sequence transmission problem is widely used in the literature of communication protocols [8], most often referred to by the name of its most famous solution: the Alternating Bit Protocol [2].

The sequence transmission problem can be stated in this way [8]: consider two agents, called the sender and the receiver. The sender will transmit in steps an arbitrarily long sequence of data messages to the receiver. The receiver must print out the sequence in the correct order and without duplicates.

This problem clearly has a trivial solution if we assume that messages sent by the sender can not be lost, corrupted, duplicated, or reordered. However, once we consider a faulty communication medium, the problem becomes far more complicated.

In the supervisory control framework, the sequence transmission problem is modeled in [5], [13]. The sequence transmission protocol modeled in this paper is adapted from [13]. Here, the physical requirements are: the sender and the receiver can only communicate via message exchanges, communication is asynchronous, all messages are transmitted over a half-duplex channel (i.e., bidirectional channel which may be used only in one direction at a time), the channel may lose messages, the sender may append one control bit 0 or 1 to data messages, and the receiver may transmit acknowledgement (ack) of one bit 0 or 1.

B. Protocol Model

The set $\Sigma$ of all possible events is given by

$$\Sigma = \{g, s_0, s_1, r_{a_0}, r_{a_1}, l, a_0, a_1, r_{s_0}, r_{s_1}, p\}$$

where

- $g :=$ get new data,
- $s_0 :=$ send data with control bit set to 0,
- $s_1 :=$ send data with control bit set to 1,
- $r_{s_0} :=$ receive data with control bit set to 0,
- $r_{s_1} :=$ receive data with control bit set to 1,
- $p :=$ print data received,
- $a_0 :=$ acknowledge data with control bit set to 0,
- $a_1 :=$ acknowledge data with control bit set to 1,
- $r_{a_0} :=$ receive ack with control bit set to 0,
- $r_{a_1} :=$ receive ack with control bit set to 1,
- $l :=$ contents in the channel are lost.

In this model, $\Sigma_o = \{g, s_0, s_1, a_0, a_1, p\}$.

Figure 1 is the behavior of the plant component SENDER. The behavior of SENDER is: SENDER gets new data and sends it; a loss causes it to re-send some data; receiving an acknowledgement can cause it to re-send the same data or some new data; receiving some acknowledgements can also cause it to get new data; then it repeats all the above actions.

Note that each automaton self-loops all events in $\Sigma$ that are not shown in its diagram.

![Fig. 1. The plant component SENDER](image1)

Figure 2 is the behavior of the plant component RECEIVER. Figure 3 illustrates the behavior of the plant component CHANNEL, which is a buffer with capacity one.

![Fig. 2. The plant component RECEIVER](image2)

Figure 4 and 5 show the specification requirement for SpecSNDR and SpecRcvr.

Figure 6 is the specification automaton SpecSEQ for the sequence transmission requirement. The behavior of SpecSEQ requires that the sequence printed out should equal the input sequence. SpecSEQ shows that $g$ and $p$ must alternate, which captures the legal requirement of the sequence transmission problem.

There exist two decentralized controllers in this protocol: the controller on the sender side, called controller one, and the controller on the receiver side, called controller two. Both decentralized controllers have limited controllable and observable event subsets.
The set of controllable events for controller one is, $\Sigma_{c,1} \triangleq \{g, s_0, s_1\}$. Namely, controller one can only control “get data” and “send data”, and cannot control “receive data”, “lose data” and all the events on the receiver side.

From the receiver side, controller two can only observe the following events, $\Sigma_{o,2} \triangleq \{a_0, a_1, r_{s_0}, r_{s_1}, p\}$. The set of controllable events for controller two is, $\Sigma_{c,2} \triangleq \{a_0, a_1, p\}$.

The behavior of the whole plant system is represented by the synchronous product of all the plant components $G_1 = \text{SENDER}, G_2 = \text{RECEIVER},$ and $G_3 = \text{CHANNEL}$. Therefore, the whole plant system is $G = G_1 || G_2 || G_3$. The language generated by plant system $G$ is $L(G) = L(G_1 || G_2 || G_3) = L(G_1) || L(G_2) || L(G_3)$.

The languages of the specifications are: $K_1 = L(\text{SpecSNDR}), K_2 = L(\text{SpecRCVR}), \text{and } K_3 = L(\text{SpecSEQ}).$ Therefore the global specification $K$ is represented by $K = K_1 || K_2 || K_3$.

### C. Verification of the Protocol

We need to verify whether the global specification $K$ is co-observable w.r.t. $L(G)$. According to Propositions 2 and 3, if each component requirement $K_1, K_2$ and $K_3$ is co-observable w.r.t. $L(G)$, then $K$ is co-observable w.r.t. $L(G)$.

According to Proposition 1, it is enough to only consider a subsystem $L(G')$ instead of the whole system $L(G)$.

1. Verification whether $K_1 = L(\text{SpecSNDR})$ is co-observable w.r.t. $L(G)$.

Step 1.1, we start from the empty subset of $G$, i.e., we let $G' = G_{\Sigma^*}$, and verify whether $K_1$ is co-observable w.r.t. $L(G_{\Sigma^*}) = \Sigma^*$. Fortunately, by examining the strings in $\overline{K_1}$, we find that $K_1$ is co-observable w.r.t. $\Sigma^*$.

Thus we conclude that $K_1$ is co-observable w.r.t. $L(G)$ by Proposition 1 and Corollary 1.

2. Verification whether $K_2 = L(\text{SpecRCVR})$ is co-observable w.r.t. $L(G)$.

Step 2.1, we start from the empty subset of $G$, i.e., we let $G' = G_{\Sigma^*}$, and verify whether $K_2$ is co-observable w.r.t. $L(G_{\Sigma^*}) = \Sigma^*$. This is true.

Thus we conclude that $K_2$ is co-observable w.r.t. $L(G)$ by Proposition 1 and Corollary 1.

3. Verification whether $K_3 = L(\text{SpecSEQ})$ is co-observable w.r.t. $L(G)$.

Step 3.1, we start from the empty subset of $G$, i.e., we let $G' = G_{\Sigma^*}$, and verify whether $K_3$ is co-observable w.r.t. $L(G_{\Sigma^*}) = \Sigma^*$.

It is easy to find that $K_3$ is not co-observable w.r.t. $\Sigma^*$. There is a very short counter example $t = \epsilon \in \overline{K_3} \cap L(G')$, $\sigma = p \in \Sigma_{c,2}, t\sigma = p \in L(G' \setminus \overline{K_3}), t' = g, P_2(t) = P_2(t') = \epsilon, t'\sigma = gp \in \overline{K_3} \cap L(G')$. Controller two cannot distinguish between $t\sigma = p$ and $t'\sigma = gp$, thus $\sigma = p$ cannot be disabled. Since $\sigma = p \notin \Sigma_{c,1}$, controller one cannot disable $p$ either.

Step 3.2, check whether this counter example is accepted by all the other components. It can be found that neither the plant subsystem RECEIVER nor the specification automaton SpecRCVR accept this counter example. We select SpecRCVR. Since SpecRCVR has already been verified to
be co-observable in Step 2.1, we can thus add it to plants according to Proposition 5. Then $G' = G_{\Sigma'}$ becomes $G' = G_{\Sigma'} \| \text{SpecRCVR} = \text{SpecRCVR}$.

Step 3.3, verify whether $K_3$ is co-observable w.r.t. $L(G') = L(\text{SpecRCVR}) = K_2$.

It is also easy to find that $K_3$ is not co-observable w.r.t. $L(G') = L(\text{SpecRCVR})$. Counter example: $t = g \in K_3 \cap L(G')$, $\sigma = g \in \Sigma_{c,1}$, $t\sigma = g \in L(G') \setminus K_3$, $t' = g_{s_0}r_{a_0}g \in L(G') \setminus K_3$, $t' = g_{s_0}r_{a_0}p_{a_0}r_{a_0}$, $P_1(t) = P_1(t') = g_{s_0}r_{a_0}t\sigma = g_{s_0}r_{a_0}p_{a_0}r_{a_0}g \in K_3 \cap L(G')$. For controller one, it cannot distinguish between $t\sigma = g_{s_0}r_{a_0}g$ and $t' = g_{s_0}r_{a_0}g$, thus $\sigma = g$ cannot be disabled. For controller two, $\sigma = g \notin \Sigma_{c,2}$, hence $\sigma = g$ cannot be disabled either.

Step 3.4, check whether this counter example is accepted by all the other components. It can be found that both the plant subsystem SENDER and the specification automaton SpecSNDR do not accept this counter example. We select SpecSNDR. Since SpecSNDR has already been verified to be co-observable in Step 1.1, we can thus add it to plants according to Proposition 5. Then $G' = \text{SpecRCVR}$ becomes $G' = G_{\Sigma'} \| \text{SpecRCVR}$.

Step 3.5, verify whether $K_3$ is co-observable w.r.t. $L(G') = L(\text{SpecRCVR})|\text{SpecSNDR} = K_2||K_1$.

It is also not hard to find that $K_3$ is not co-observable w.r.t. $L(G') = L(\text{SpecRCVR})|\text{SpecSNDR})$. Counter example: $t = g_{s_0}r_{a_0} \in K_3 \cap L(G')$, $\sigma = g \in \Sigma_{c,1}$, $t\sigma = g_{s_0}r_{a_0}g \in L(G') \setminus K_3$, $t' = g_{s_0}r_{a_0}p_{a_0}r_{a_0}$, $P_1(t) = P_1(t') = g_{s_0}r_{a_0}$, $t'\sigma = g_{s_0}r_{a_0}p_{a_0}r_{a_0}g \in K_3 \cap L(G')$. For controller one, it cannot distinguish between $t\sigma = g_{s_0}r_{a_0}g$ and $t'\sigma = g_{s_0}r_{a_0}p_{a_0}r_{a_0}g$, thus $\sigma = g$ cannot be disabled. For controller two, $\sigma = g \notin \Sigma_{c,2}$, hence $\sigma = g$ cannot be disabled either.

Step 3.6: check whether this counter example is accepted by all the other components. It can be found that only the plant subsystem $G_3 = \text{CHANNEL}$ does not accept this counter example. Thus $G' = G_{\Sigma'} \| \text{SpecRCVR})|\text{SpecSNDR}$ becomes $G' = G_{\Sigma'} \| \text{SpecRCVR})|\text{SpecSNDR}|\text{CHANNEL}$.

Step 3.7: verify whether $K_3$ is co-observable w.r.t. $L(G') = L(\text{SpecRCVR})|\text{SpecSNDR}|\text{CHANNEL} = K_2||K_1|\text{CHANNEL}$.

It can be found that $K_3$ is co-observable w.r.t. $L(G') = K_2||K_1|\text{CHANNEL}$.

Since $K_2$ and $K_1$ are both co-observable w.r.t. $L(G)$, we thus conclude that $K = K_3||K_2||K_3$ is co-observable w.r.t. $L(G)$ by Propositions 5 and 6.

In the above example, we verify each specification component individually. In addition, the most complex step involves only four out of six automata in the system. Further, the complete plant does not need to be composed together.

VI. CONCLUSION

In this paper, we introduce an approach called incremental verification of co-observability. We present results that provide the technical foundation of the method. We then present our algorithm and a classical communication example. This new approach allows decentralized control to be applied to larger systems, as it allows co-observability to be verified using only a portion of the system at a given time.

Future work will further demonstrate this approach using more examples, and develop heuristics to determine how best to select the next component of the system to verify in our incremental verification algorithm.

REFERENCES


