Department of Computing and Software

Faculty of Engineering — McMaster University

Verifying Co-observability in Discrete-event Systems using an Incremental Approach

by

Huailiang Liu, Ryan J. Leduc, Robi Malik, and S. L. Ricker

CAS Report Series **CAS-13-06-RL**

Department of Computing and Software **November 2013**

Information Technology Building
McMaster University
1280 Main Street West Hamilton, Ontario, Canada L8S 4K1

Verifying Co-observability in Discrete-event Systems using an Incremental Approach

Huailiang Liu[1], Ryan J. Leduc[1], Robi Malik[2], and S. L. Ricker[3]

[1] Department of Computing and Software, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada

[2] Department of Computer Science University of Waikato, Private Bag 3105, Hamilton, New Zealand

[3] Department of Mathematics and Computer Science Mount Allison University, Sackville, NB E4L 1E6, Canada

Technical Report CAS-13-06-RL
Department of Computing and Software
McMaster University

November 22, 2013

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.

Keywords: discrete-event systems, supervisory control, decentralized control, incremental verification, co-observability

Contents

1 Introduction

One of the main challenges in the control of discrete-event systems (DES) is the combinatorial explosion of the product 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] 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.

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 a property called co-observability [15]. Nevertheless, 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. Properties of incremental verification of co-observability, the corresponding algorithm, and a verification example for a classical communication protocol are presented. Compared to traditional monolithic methods, the new method is potentially much more efficient for very large systems composed of many subsystems, rendering some intractable state-space explosion problems to be manageable. To the best of our knowledge, no existing work explores incremental verification of co-observability.

This paper is organized as follows. Section 2 reviews some related concepts of supervisory control in DES. Section 3 provides the details for the incremental verification of co-observability. Section 4 provides an algorithm for incremental verification of co-observability guided by counter examples. Section 5 applies this algorithm to verify a classical communication protocol. Finally, Section 6 provides conclusions and future work.

2 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 Σ be a finite set of distinct symbols (*events*), and Σ^* be the set of all finite sequences of events plus ϵ, the *empty string*. A language L over Σ 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: $L\sigma := \{s\sigma \in \Sigma^* | s \in L,\ \sigma \in \Sigma\}$.

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* $\overline{L}$ of a language $L \subseteq \Sigma^*$ is defined as follows:

$\overline{L} := \{t \in \Sigma^* | t \leq s \ for\ some\ s \in L\}$.

A language L is said to be *prefix-closed* if $L = \overline{L}$. In this paper, we are not concerned with nonblocking, thus we assume that all languages are prefix-closed.

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

$P_i(\epsilon) := \epsilon$

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

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

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

The *inverse projection* P_i^{-1} is the mapping $Pwr(\Sigma_i^*) \to Pwr(\Sigma^*)$ defined on sets of strings (or languages), where $Pwr(\Sigma_i^*)$ and $Pwr(\Sigma^*)$ denote all subsets of Σ_i^* and Σ^* respectively. Given any $L_i \in \Sigma_i^*$, 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 Σ, partial transition function $\delta :\ Q \times \Sigma \to Q$ and initial state q_0.

We will always assume that a DES has a finite state and event set, and is deterministic. *The closed behavior* for a DES G is denoted by a regular language $L(G)$, and is defined to be: $L(G) := \{s \in \Sigma^* | \delta(q_0,\ s)!\}$, where $\delta(q_0,\ s)!$ means that δ is defined for $s \in \Sigma^*$ at state q_0. The *reachable state subset* of G, denoted as Q_r, is defined as: $Q_r := \{q \in Q | (\exists s \in \Sigma^*)\delta(q_0,\ s) = q\}$. We say that G is *reachable* if $Q_r = Q$.

Let $G_1 = (Q_1,\ \Sigma_1,\ \delta_1,\ q_{0,1})$ and $G_2 = (Q_2,\ \Sigma_2,\ \delta_2,\ q_{0,2})$ be two automata. Their *synchronous product* is a DES G over event set $\Sigma = \Sigma_1 \cup \Sigma_2$, $G := G_1||G_2 = (Q_1 \times Q_2,\ \Sigma,\ \delta,\ (q_{0,1},\ q_{0,2}))$. For $(q_1, q_2) \in Q_1 \times Q_2$, we define:

$$\delta((q_1, q_2),\ \sigma) := \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)),\ if\ \sigma \in (\Sigma_1 \cap \Sigma_2),\ \delta_1(q_1, \sigma)!\ and\ \delta_2(q_2, \sigma)!; \\ (\delta_1(q_1, \sigma), q_2),\ if\ \sigma \in \Sigma_1 \backslash \Sigma_2\ and\ \delta_1(q_1, \sigma)!; \\ (q_1, \delta_2(q_2, \sigma)),\ if\ \sigma \in \Sigma_2 \backslash \Sigma_1\ and\ \delta_2(q_2, \sigma)!. \end{cases}$$

The *synchronous product of languages* L_1 and L_2, denoted by $L_1||L_2$, is defined to be: $L_1||L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$. If both L_1 and L_2 are over the same event set Σ, then their languages 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 Σ. If we are given an automaton defined over a subset of Σ, we can simply self-loop the missing events at each state.

In supervisory control, the event set Σ is partitioned into two disjoint sets: *the controllable event set* Σ_c and *the uncontrollable event set* Σ_{uc}. Controllable events can be prevented from happening, or *disabled*, by a supervisor S, while uncontrollable events cannot.

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

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 2.1. *Let K, $L = \overline{L}$ be languages over event set Σ. Let $I = \{1, ..., n\}$ be an index set. Let $\Sigma_{c,i} \subseteq \Sigma$ and $\Sigma_{o,i} \subseteq \Sigma$ be sets of controllable and observable events, respectively, for $i \in I$, where $\Sigma_c = \cup_{i=1}^{n} \Sigma_{c,i}$ and $I_c(\sigma) := \{i \in I \,|\sigma \in \Sigma_{c,i}\}$. Let $P_i : \Sigma^* \to \Sigma_{o,i}^*$ be a natural projection. A language K is said to be co-observable with respect to L, $\Sigma_{o,i}$, $\Sigma_{c,i}$, $i \in I$, if,*
$(\forall t \in \overline{K} \cap L)\ (\forall \sigma \in \Sigma_c)\ t\sigma \in L \backslash \overline{K} \Rightarrow (\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset.$

Notice that in definition 2.1, when there is only one controller, $I = \{1\}$, the property is called *observability* [10]. To apply the definition to languages represented by plant G and specification 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 $\overline{K}$, reasonably, we check all strings in $\overline{K} \cap L$. 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 $\overline{K}$ (i.e., $t\sigma \in L \backslash \overline{K}$, where $t\sigma \in L$ and $t\sigma \notin \overline{K}$) 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 σ). Note that, by default, a controller $i \in I$ will enable all events $\sigma \in \Sigma \backslash \Sigma_{c,i}$.

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

We can synthesize decentralized controllers that ensure the supervised system generates exactly the behavior in the specification K, if K is controllable and satisfies co-observability [15].

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

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

Definition 2.2. *Let K, $L = \overline{L}$ be languages over event set Σ. Let $\Sigma_{c,i} \subseteq \Sigma$ and $\Sigma_{o,i} \subseteq \Sigma$ be sets of controllable and observable events, respectively, for $i \in I$, $I_c(\sigma) :=\{i \in I \,|\sigma \in \Sigma_{c,i}\}$. Let $P_i : \Sigma^* \to \Sigma_{o,i}^*$ 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*
- *$\sigma \in \Sigma_c$;*
- *$t \in \overline{K} \cap L$ and $t\sigma \in L \backslash \overline{K}$;*
- *$(\forall i \in I_c(\sigma))\ t_i\sigma \in \overline{K} \cap L$;*
- *$(\forall i \in I_c(\sigma))\ P_i(t) = P_i(t_i)$.*

Please note that for $i \in I \backslash I_c(\sigma)$, the corresponding controllers cannot disable σ, therefore these t_i in C cannot affect whether Definition 2.2 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 $\overline{K}$ in Definition 2.2 by $\overline{K} \cap \overline{K'}$, then C will no longer be a valid counter example.

Definition 2.3. *If $C = (\sigma,\ t,\ t_1, ..., t_n)$ is a co-observability counter example for the specification $K = \overline{K}$ and the plant $L = \overline{L}$, we say that specification $K' = \overline{K'} \subseteq \Sigma^*$ rejects C, if:*
- $t \notin \overline{K'}$, or
- $(\exists i \in I_c(\sigma))\ t_i\sigma \notin \overline{K'}$.

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 2.2 by $L \cap L'$, then C will no longer be a valid counter example.

Definition 2.4. *If $C = (\sigma,\ t,\ t_1, ..., t_n)$ is a co-observability counter example for the specification $K = \overline{K}$ and the plant $L = \overline{L}$, we say that plant $L' = \overline{L'} \subseteq \Sigma^*$ rejects C, if:*
- $t\sigma \notin L'$, or
- $(\exists i \in I_c(\sigma))\ t_i\sigma \notin L'$.

3 Incremental Verification of Co-observability

In practice, the synchronous product of the plant $L = L_1||...||L_m$ and the specification $K = K_1||...||K_r$ may be very large, and it is difficult to verify co-observability due to the state-space explosion problem. Therefore, we introduce an incremental verification method for co-observability.

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

Proposition 3.1. *Let K, $L = \overline{L}$, $M = \overline{M}$, be languages over event set Σ. If K is co-observable with respect to M and $L \subseteq M$, then K is co-observable with respect to L.*

Proof. Assume K is co-observable w.r.t. M and $L \subseteq M$.
Must show K is co-observable w.r.t. L.
Let $t \in \overline{K} \cap L$.
Let $\sigma \in \Sigma_c$.
Assume $t\sigma \in L \backslash \overline{K}$.
Sufficient to show $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.
As $L \subseteq M$ and $t \in \overline{K} \cap L$, we have: $t \in \overline{K} \cap M$.
As $t\sigma \in L \backslash \overline{K}$, we have: $t\sigma \in L$ and $t\sigma \notin \overline{K}$.
$\Rightarrow t\sigma \in M$ and $t\sigma \notin \overline{K}$, as $L \subseteq M$.
$\Rightarrow t\sigma \in M \backslash \overline{K}$.
Then $t \in \overline{K} \cap M$ and $t\sigma \in M \backslash \overline{K}$.
As K is co-observable w.r.t. M, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap M = \emptyset$.
As $L \subseteq M$, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.
As $t \in \overline{K} \cap L$ and $\sigma \in \Sigma_c$ were chosen arbitrarily, we conclude that K is co-observable w.r.t. L.

$\square$

Proposition 3.1 is a fundamental proposition which can be paraphrased as follows: if a specification language is co-observable w.r.t. a language M, then it must be co-observable w.r.t. all the prefix-closed sublanguages of M. We also know that if L_1 and L_2 are prefix-closed, so is $L = L_1 \cap L_2$.

Corollary 3.1. *Let K, $L_1 = \overline{L_1}$, and $L_2 = \overline{L_2}$ be languages over event set Σ. If K is co-observable with respect to L_1 then K is co-observable with respect to $L = L_1 \cap L_2$.*

Proof. As K is co-observable w.r.t. L_1 and $L = L_1 \cap L_2 \subseteq L_1$, by Proposition 3.1, we have: K is co-observable w.r.t. L.

□

If we want to verify whether K is co-observable w.r.t. $L = L_1 \cap ... \cap L_m$, it is sufficient to show that there exists a subset of indexes $\{j_1, ..., j_k\} \subseteq \{1, ..., m\}$ such that K is co-observable w.r.t. $L' = L_{j_1} \cap ... \cap L_{j_k}$. This follows from Corollary 3.1.

Proposition 3.2. *Let K_1, K_2 and L be prefix-closed languages over event set Σ. If both K_1 and K_2 are, respectively, co-observable with respect to L, then $K = K_1 \cap K_2$ is co-observable with respect to L.*

Proof. Assume K_1, K_2 and L are prefix-closed languages over event set Σ.
Assume K_1 is co-observable w.r.t. L.
Assume K_2 is co-observable w.r.t. L.
Must show K is co-observable w.r.t. L.
Let $t \in \overline{K} \cap L$.
Let $\sigma \in \Sigma_c$.
Assume $t\sigma \in L \backslash \overline{K}$.
Sufficient to show $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.
As $K = K_1 \cap K_2$, K_1 and K_2 are prefix-closed, we have: $\overline{K} = \overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$.
As $t \in \overline{K} \cap L$, we have: $t \in \overline{K_1} \cap \overline{K_2} \cap L$.
$\Rightarrow t \in \overline{K_1} \cap L$ and $t \in \overline{K_2} \cap L$.
As $t\sigma \in L \backslash \overline{K}$, we have: $t\sigma \in L$ and $t\sigma \notin \overline{K}$.
As $t\sigma \notin \overline{K}$, and prefix-closed languages $\overline{K} = \overline{K_1} \cap \overline{K_2}$, we have: $t\sigma \notin \overline{K_1}$ or $t\sigma \notin \overline{K_2}$.
$\Rightarrow t\sigma \notin \overline{K_1}$ *and* $t\sigma \in L$ or $t\sigma \notin \overline{K_2}$ *and* $t\sigma \in L$.
Case 1) $t\sigma \notin \overline{K_1}$ *and* $t\sigma \in L$.
Then $t \in \overline{K_1} \cap L$, $t\sigma \notin \overline{K_1}$ *and* $t\sigma \in L$.
$\Rightarrow t \in \overline{K_1} \cap L$, $t\sigma \in L \backslash \overline{K_1}$.
As K_1 is co-observable w.r.t. L, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1} \cap L = \emptyset$.
As $\overline{K} \subseteq \overline{K_1}$, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.
Case 2) $t\sigma \notin \overline{K_2}$ *and* $t\sigma \in L$.
Then $t \in \overline{K_2} \cap L$, $t\sigma \notin \overline{K_2}$ *and* $t\sigma \in L$.
$\Rightarrow t \in \overline{K_2} \cap L$, $t\sigma \in L \backslash \overline{K_2}$.
As K_2 is co-observable w.r.t. L, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_2} \cap L = \emptyset$.
As $\overline{K} \subseteq \overline{K_2}$, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap L = \emptyset$.
As $s \in \overline{K}$ and $\sigma \in \Sigma_c$ are chosen arbitrarily, we can conclude by case 1) and 2) that $K = K_1 \cap K_2$ is co-observable w.r.t. L.

□

Proposition 3.2 can be extended to an arbitrary number of specification languages.

In the incremental verification of co-observability, given a specification language $K = K_1 \cap ... \cap K_r$ and a language L, if we want to verify whether K is co-observable w.r.t. L, it is enough to simply show that for each $j \in \{1, ..., r\}$, K_j is co-observable w.r.t. L. Combining this with Proposition 3.1, we see that we can use a component L' instead of the global system L for the verification.

Proposition 3.3. *Let K_1, K_2, M_1, and M_2 be prefix-closed languages over event set Σ. If K_1 is co-observable w.r.t. 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 $M = M_1 \cap M_2$.*

Proof. Assume K_1, K_2, M_1, and M_2 are prefix-closed.

Assume K_1 is co-observable w.r.t. M_1.

Assume K_2 is co-observable w.r.t. M_2.

Must show K is co-observable w.r.t. M.

As $M = M_1 \cap M_2$, we have: $M \subseteq M_1$ *and* $M \subseteq M_2$.

As K_1 is co-observable w.r.t. the prefix-closed language M_1, and $M \subseteq M_1$, we have: K_1 is co-observable w.r.t. M, by Proposition 3.1.

As K_2 is co-observable w.r.t. the prefix-closed language M_2, and $M \subseteq M_2$, we have: K_2 is co-observable w.r.t. M, by Proposition 3.1.

As K_1 and K_2 are both co-observable w.r.t. M, respectively, and as K_1, K_2, and M are prefix-closed, we have: $K = K_1 \cap K_2$ is co-observable w.r.t. M by Proposition 3.2.

□

Proposition 3.3 is useful for verifying co-observability in large decentralized systems. As described above, to verify whether a specification language $K = K_1 \cap K_2$ is co-observable w.r.t. a language $M = M_1 \cap M_2$, we can simply show that K_1 is co-observable w.r.t. M_1, and K_2 is co-observable w.r.t. M_2. The results of Proposition 3.3 can be extended to an arbitrary number of specification and plant component languages.

Proposition 3.4. *Let K, L be prefix-closed languages over event set Σ. If $K \supseteq L$ then K is co-observable w.r.t. L.*

Proof. Assume $K \supseteq L$.

Then every string in L is also in $\overline{K}$.

$\Rightarrow L \backslash \overline{K} = \emptyset$.

Therefore the pre-condition of co-observability, $t\sigma \in L \backslash \overline{K}$, is always $false$.

As there does not exist $t\sigma \in L \backslash \overline{K}$ satisfying the precondition of definition 2.1, the result that K is co-observable w.r.t. L is trivially $true$.

We thus conclude that K is co-observable w.r.t. L.

□

Proposition 3.4 indicates that any language is co-observable w.r.t. all its sub-languages.

Proposition 3.5. *Let K_1, K_2 and M be prefix-closed languages over event set Σ. 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.*

Proof. Assume K_1 is co-observable w.r.t. $M \cap K_2$.

Assume K_2 is co-observable w.r.t. M.

Assume K_1, K_2 and M are prefix-closed.

Must show K is co-observable w.r.t. M.

Let $t \in \overline{K} \cap M$.

Let $\sigma \in \Sigma_c$.

Assume $t\sigma \in M \backslash \overline{K}$.

Sufficient to show $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap M = \emptyset$.

As $t \in \overline{K} \cap M$ and $\overline{K} = \overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$ for prefix-closed languages, we have: $t \in \overline{K_1} \cap M$ and $t \in \overline{K_2} \cap M$.

As $t\sigma \in M \backslash \overline{K}$, we have: $t\sigma \in M$ and $t\sigma \notin \overline{K}$.

$\Rightarrow t\sigma \in M$ and ($t\sigma \notin \overline{K_1}$ or $t\sigma \notin \overline{K_2}$) for prefix-closed languages.

$\Rightarrow$ ($t\sigma \in M$ and $t\sigma \notin \overline{K_1}$) or ($t\sigma \in M$ and $t\sigma \notin \overline{K_2}$).

Case 1) $t\sigma \in M$ and $t\sigma \notin \overline{K_2}$.

Then $t\sigma \in M \backslash \overline{K_2}$.

$\Rightarrow t \in \overline{K_2} \cap M$ and $t\sigma \in M \backslash \overline{K_2}$.

As K_2 is co-observable w.r.t. M, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_2} \cap M = \emptyset$.

As $\overline{K} = \overline{K_1 \cap K_2} \subseteq \overline{K_2}$, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap M = \emptyset$.

Case 2) $t\sigma \in M$ and $t\sigma \notin \overline{K_1}$.

Under this case, there are still two cases: $t\sigma \notin \overline{K_2}$ or $t\sigma \in \overline{K_2}$.

As $t\sigma \notin \overline{K_2}$ is the same as case 1), we only consider $t\sigma \in \overline{K_2}$.

$\Rightarrow t\sigma \in M$ and $t\sigma \notin \overline{K_1}$ and $t\sigma \in \overline{K_2}$.

We will use the assumption that K_1 is co-observable w.r.t. $M \cap K_2$ to prove the result.

We will thus look on K_1 as a specification, and $M \cap K_2$ as a plant.

As $t\sigma \in M$ and $t\sigma \notin \overline{K_1}$ and $t\sigma \in \overline{K_2}$, we have: $t\sigma \in M \cap \overline{K_2}$ and $t\sigma \notin \overline{K_1}$.

$\Rightarrow t\sigma \in (M \cap \overline{K_2}) \backslash \overline{K_1}$.

As $t \in \overline{K_1} \cap M$ and $t \in \overline{K_2} \cap M$, we have: $t \in \overline{K_1} \cap (M \cap \overline{K_2})$.

$\Rightarrow t \in \overline{K_1} \cap (M \cap \overline{K_2})$ and $t\sigma \in (M \cap \overline{K_2}) \backslash \overline{K_1}$.

As K_1 is co-observable w.r.t. $M \cap \overline{K_2}$, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1} \cap (M \cap \overline{K_2}) = \emptyset$.

$\Rightarrow (\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap (\overline{K_1} \cap \overline{K_2} \cap M) = \emptyset$.

As $\overline{K} = \overline{K_1} \cap \overline{K_2}$ for prefix-closed languages, we have: $(\exists i \in I_c(\sigma))\ P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap M = \emptyset$.

As $t \in \overline{K}$ and $\sigma \in \Sigma_c$ are chosen arbitrarily, we conclude by cases 1) and 2), that K is co-observable w.r.t. M.

□

Proposition 3.5 is used to show co-observability when K_2 is co-observable w.r.t. M and K_1 is not co-observable w.r.t. M. However, we still have that $K = K_1 \cap K_2$ is co-observable w.r.t. M if K_1 is co-observable w.r.t. the extended system $M \cap K_2$, according to Proposition 3.5. Essentially, Proposition 3.5 allows us to treat specification K_2 as a plant component.

Proposition 3.6. *Let K_1, K_2, M_1 and M_2 be prefix-closed languages over event set Σ. 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. $M = M_1 \cap M_2$.*

Proof. Assume K_1 is co-observable w.r.t. $M_1 \cap K_2$.

Assume K_2 is co-observable w.r.t. M_2.

Assume K_1, K_2, M_1 and M_2 are prefix-closed.

Must show K is co-observable w.r.t. M.

As $M = M_1 \cap M_2 \subseteq M_1$, we have: $M \cap K_2 \subseteq M_1 \cap K_2$.

As K_1 is co-observable w.r.t. $M_1 \cap K_2$, we have: K_1 is co-observable w.r.t. $M \cap K_2$, by Proposition 3.1.

As K_2 is co-observable w.r.t. M_2, and $M = M_1 \cap M_2 \subseteq M_2$, we have: K_2 is co-observable w.r.t. M, by Proposition 3.1.

$\Rightarrow K_1$ is co-observable w.r.t. $M \cap K_2$, and K_2 is co-observable w.r.t. M.

We thus have that $K = K_1 \cap K_2$ is co-observable w.r.t. M, for prefix-closed languages, by Proposition 3.5.

$\square$

Compared to Proposition 3.5, Proposition 3.6 provides us with a more general way to incrementally verify co-observability, especially for systems composed of a large number of subsystems.

Here, the plant language $M = M_1 \cap M_2$ has only two components. In fact, the system can have an arbitrary number of components, which is also true for the number of components of the specification languages.

Proposition 3.7. *Let K_1, K_2 and M be prefix-closed languages over event set Σ. 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.*

Proof. Assume K_1, K_2, and M are prefix-closed.

Assume K_1 is not co-observable w.r.t. $M \cap K_2$.

Must show K is not co-observable w.r.t. M.

Sufficient to show $(\exists t \in \overline{K} \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \backslash \overline{K}$ and $(\forall i \in I_c(\sigma))$ $P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap M \neq \emptyset$.

We will use the condition that K_1 is not co-observable w.r.t. $M \cap K_2$ to prove the result.

We thus look on K_1 as a specification, and $M \cap K_2$ as a plant.

As K_1 is not co-observable w.r.t. $M \cap K_2$, it follows that: $(\exists t \in \overline{K_1} \cap (M \cap K_2))$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in (M \cap K_2)\backslash \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1} \cap (M \cap K_2) \neq \emptyset$.

$\Rightarrow (\exists t \in \overline{K_1} \cap (M \cap K_2))$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \cap K_2$, $t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $P_i^{-1}[P_i(t)]\sigma \cap \overline{K_1} \cap (M \cap K_2) \neq \emptyset$.

$\Rightarrow (\exists t \in \overline{K_1} \cap (M \cap K_2))$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \cap K_2$, $t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap (M \cap K_2))$ $P_i(t) = P_i(t_i)$, $t_i\sigma \in \overline{K_1} \cap (M \cap K_2)$.

$\Rightarrow (\exists t \in \overline{K_1} \cap K_2 \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \cap K_2$, $t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap K_2 \cap M)$ $P_i(t) = P_i(t_i)$, $t_i\sigma \in \overline{K_1} \cap K_2 \cap M$.

$\Rightarrow (\exists t \in \overline{K_1} \cap \overline{K_2} \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \cap K_2$, $t\sigma \notin \overline{K_1}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap \overline{K_2} \cap M)$ $P_i(t) = P_i(t_i)$, $t_i\sigma \in \overline{K_1} \cap \overline{K_2} \cap M$, as K_2 is prefix-closed.

$\Rightarrow (\exists t \in \overline{K_1} \cap \overline{K_2} \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \cap K_2$, $t\sigma \notin \overline{K_1} \cap \overline{K_2}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K_1} \cap \overline{K_2} \cap M)$ $P_i(t) = P_i(t_i)$, $t_i\sigma \in \overline{K_1} \cap \overline{K_2} \cap M$.

$\Rightarrow (\exists t \in \overline{K} \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \cap K_2$, $t\sigma \notin \overline{K}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K} \cap M)$ $P_i(t) = P_i(t_i)$, $t_i\sigma \in \overline{K} \cap M$, as $\overline{K} = \overline{K_1 \cap K_2} = \overline{K_1} \cap \overline{K_2}$, since K_1 and K_2 are prefix-closed.

$\Rightarrow (\exists t \in \overline{K} \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M$, $t\sigma \notin \overline{K}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K} \cap M)$, $P_i(t) = P_i(t_i)$, $t_i\sigma \in \overline{K} \cap M$, as $M \cap K_2 \subseteq M$.

$\Rightarrow (\exists t \in \overline{K} \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \backslash \overline{K}$ and $(\forall i \in I_c(\sigma))$ $(\exists t_i \in \overline{K} \cap M)$ $P_i(t) = P_i(t_i)$, $t_i\sigma \in \overline{K} \cap M$.

$\Rightarrow (\exists t \in \overline{K} \cap M)$ $(\exists \sigma \in \Sigma_c)$ $t\sigma \in M \backslash \overline{K}$ and $(\forall i \in I_c(\sigma))$ $P_i^{-1}[P_i(t)]\sigma \cap \overline{K} \cap M \neq \emptyset$.

We thus conclude that K is not co-observable w.r.t. M.

$\square$

Proposition 3.7 is used for incremental verification to determine the failure of co-observability. If we can show that K_1 is not co-observable w.r.t. $M \cap K_2$, then we can conclude that $K = K_1 \cap K_2$ is not co-observable w.r.t. M.

In the incremental verification of co-observability, we can use each proposition provided in this paper independently, and we also can combine any of the above propositions to verify co-observability in a very flexible way.

Algorithm 1 Incremental Coobservability Verification

1: **input** plants $\mathcal{L} = \{L_1, \ldots, L_m\}$,
    specifications $\mathcal{K} = \{K_1, \ldots, K_r\}$;
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 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** "$\mathcal{K} = \{K_1, \ldots, K_r\}$ is *not* co-observable w.r.t. $\mathcal{L} = \{L_1, \ldots, L_m\}$, 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 $\mathcal{K} = \mathcal{K} \setminus \mathcal{K}'$, $\mathcal{L} = \mathcal{L} \cup \mathcal{K}'$;
17: **end while**
18: **stop** "$\mathcal{K} = \{K_1, \ldots, K_r\}$ is co-observable w.r.t. $\mathcal{L} = \{L_1, \ldots, L_m\}$";

4 Algorithm

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

The plant is $L = L_1||...||L_m$, and the specification is $K = K_1||...||K_r$, all over the same event set Σ. We want to verify whether K is co-observable w.r.t. L.

Algorithm 1 describes incremental verification of co-observability guided by counter examples. The general idea is:

(1) If each K_i, where $i \in \{1, ..., r\}$, is co-observable w.r.t. a component of L, then K is co-observable w.r.t. L according to Propositions 3.2 and 3.3.

(2) If $K' = K_{i_1}||...||K_{i_a}$ 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 3.1 and Corollary 3.1. This is a compensation used when some K_i in (1) is not co-observable w.r.t. L.

(3) If K_i, where $i \in \{1, ..., r\}$, is co-observable w.r.t. a component of L, then K_i can be treated as a plant to be synchronized with L, according to Propositions 3.5 and 3.6.

(4) If there is a counter example which shows that K_i is not co-observable w.r.t. $L||K_1||...||K_{i-1}||K_{i+1}||...||K_r$, then we can conclude that K is not co-observable w.r.t. L, according to Proposition 3.7.

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 $\mathcal{K}' = \{K_{i_1}, ..., K_{i_a}\}$ is not co-observable w.r.t. $\mathcal{L}' = \{L_{j_1}, ..., L_{j_b}\}$, what we mean is that a component $K' = K_{i_1}||...||K_{i_a}$ where $\{i_1, ..., i_a\} \subseteq \{1, ..., r\}$ is co-observable w.r.t. a component 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 $L(S) = L(S_1)||...||L(S_m)$.

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

On line 2, if $\mathcal{K}$ is initially empty then the specification $K = \Sigma^*$ will be co-observable w.r.t L, which is trivially true. This is because $K = \Sigma^*$ is a superset of all languages over Σ and is co-observable w.r.t. every language over Σ according to Proposition 3.4.

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. L. If each component of K_i, where $i \in \{1, ..., r\}$, is co-observable w.r.t. L, then according to Propositions 3.2 or 3.3, K will be co-observable w.r.t. 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. L according to Proposition 3.1 and Corollary 3.1.

On line 6, a counter example C =$(\sigma, t$,$t_1, ..., t_n)$ will be picked to show that K' is not co-observable w.r.t. L' according to Definition 2.2. If there are many counter examples, then 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 $\mathcal{L}\backslash\mathcal{L}'$ or K_h in $\mathcal{K}\backslash\mathcal{K}'$ which does not accept C is selected according to Definition 2.3 or Definiton 2.4, 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. Thus we can give the counter example C which shows that that K is not co-observable w.r.t. plant L by Proposition 3.7.

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

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

If the subsystem consisting of specifications $\mathcal{K}'$ and plants $\mathcal{L}'$ is found to be co-observable, then line 16 removes the specifications $\mathcal{K}'$ from $\mathcal{K}$ and adds them to $\mathcal{L}$, so they are treated as plants for the remainder of the algorithm, according to Proposition 3.5. The algorithm terminates when the set $\mathcal{K}$ of specifications to be checked is empty in which case it asserts co-observability on line 18.

5 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 [8, 16].

5.1 The Sequence Transmission Problem

The sequence transmission problem is widely used in the literature of communication protocols [9], most often referred to by the name of its most famous solution: the Alternating Bit Protocol [2]. The Alternating Bit Protocol is one type of stop-and-wait protocols which are discussed in detail in [16].

The sequence transmission problem can be stated in this way [9]: 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.

One typical solution to the sequence transmission problem is the stop-and-wait protocol [16], where the sender adds a sequence number as control bits to each data frame sent, and the receiver returns a request number which serves as an acknowledgment. Since the input sequence can be arbitrarily long, it is possible that the sequence number and the request number can become arbitrarily large. Lynch [11] conjectured that at least two control bits are required for any adequate scheme of this sort, and that only one control bit will never do. Less than a year later, Bartlett *et al* [2] proved Lynch wrong by producing their well-known solution using only one control bit, now called the Alternating Bit Protocol.

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 acknowledgements of one bit 0 or 1.

5.2 Protocol Model

The set Σ 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 acknowledgement with control bit set to* 0,

$r_{a_1} :=$ *receive acknowledgement with control bit set to* 1,

$l :=$ *contents of channel are lost.*

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

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 Σ that are not shown in its diagram.

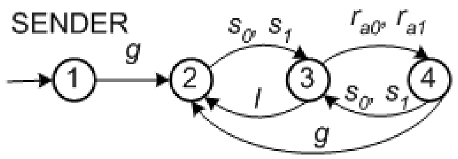

Figure 1: The plant component SENDER

Figure 2 is the behavior of the plant component RECEIVER. The behavior of RECEIVER is: RECEIVER receives new data; prints it and acknowledges it; then it may continually receive some data, and acknowledge it or print it followed by an acknowledgement; after that the above actions are repeated.

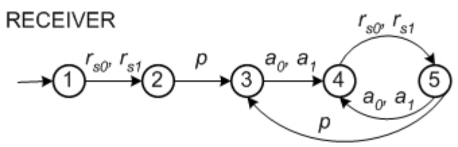

Figure 2: The plant component RECEIVER

Figure 3 illustrates the behavior of the plant component CHANNEL, which is a buffer with capacity one. The behavior of CHANNEL is: CHANNEL is either empty in state one, or contains data s_0, s_1, a_0, or a_1 respectively in state two, three, four, or five. When CHANNEL is full, it can only return to the empty state if the message in it is either received or the content is lost.

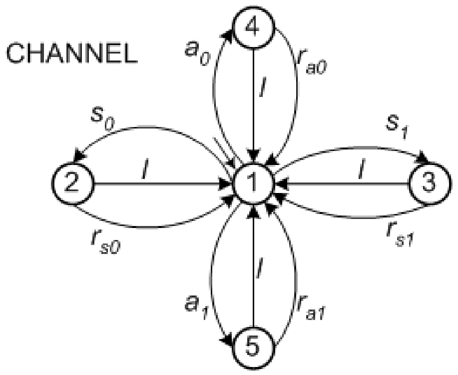

Figure 3: The plant component CHANNEL

Figure 4 is the specification requirement SpecSNDR. The behavior of SpecSNDR is: SpecSNDR gets new data and sends it with the control bit set to 0; detecting a loss or receiving the acknowledgement with the control bit set to 1 causes it to re-send the same data. Receiving the acknowledgement with the control bit set to 0 causes it to get new data, and to send it with the control bit set to 1, followed by similar behavior of sending with a control bit set to 0.

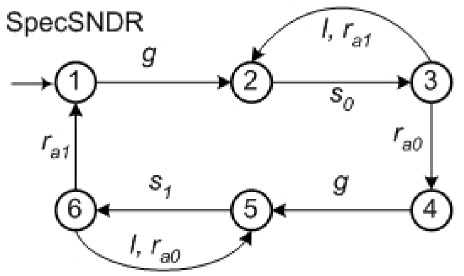

Figure 4: The Specification of SENDER

Figure 5 is the the specification requirement SpecRCVR. The behavior of SpecRCVR is: SpecRCVR receives new data with control bit set to 0, prints it and acknowledges it with control bit set to 0; then it may continually receive the same data and acknowledge it in the same way, until it receives data with control bit set to 1. It then follows similar behavior as when receiving data with control bit set to 0.

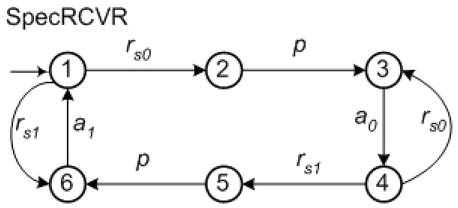

Figure 5: The Specification of RECEIVER

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. The language of SpecSEQ shows that g and p must alternate, which captures the legal requirement of the sequence transmission problem.

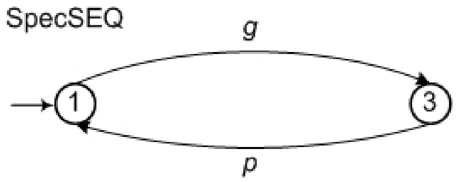

Figure 6: The Specification of Sequence

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.

From the sender side, controller one can only observe the following events:

$\Sigma_{o,1}$:=$\{g,\ s_0,\ s_1,\ r_{a_0},\ r_{a_1},\ l\}$.

Therefore, the events on the receiver side are unobservable for controller one. We assume that there is a long enough timeout mechanism, which allows controller one to recognize that a data frame has been lost. In practice, this can be done by setting a time-to-live limit in the frame. After the time limit expires, if the required data is not received, then it disappears from the channel and will never reach the other side.

The set of controllable events for controller one is, $\Sigma_{c,1} := \{g,\ s_0,\ s_1\}$. 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, $\Sigma_{o,2} := \{a_0,\ a_1,\ r_{s_0},\ r_{s_1},\ p\}$. The set of controllable events for controller two is, $\Sigma_{c,2} := \{a_0,\ a_1,\ p\}$.

The behavior of the whole plant system is represented by the synchronous product of all the plant components G_1=SENDER, G_2=RECEIVER, and G_3=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(SpecSNDR), K_2=L(SpecRCVR), and K_3=L(SpecSEQ). Therefore the global specification K is represented by $K := K_1||K_2||K_3$.

5.3 Verification for the Protocol

We need to verify whether the global specification K is co-observable w.r.t. $L(G)$. According to Propositions 3.2 and 3.3, 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 3.1, it is enough to only consider a subsystem $L(G')$ instead of the whole system $L(G)$.

1. Verification for whether K_1=L(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. Σ^*.

Thus we conclude that K_1 is co-observable w.r.t. $L(G)$ by Proposition 3.1 and Corollary 3.1.

2. Verification for whether K_2=L(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 3.1 and Corollary 3.1.

3. Verification for whether K_3=L(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. Σ^*. There is a very short counter example $t = \epsilon \in \overline{K_3} \cap L(G')$, $\sigma = p \in \Sigma_{c,2}$, $t\sigma = p \in L(G')\backslash\overline{K_3}$, $t' = g$, $P_2(t) = P_2(t') = \epsilon$, $t'\sigma = gp \in \overline{K_3} \cap L(G')$. 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 string $t\sigma = p$ 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 3.5. Then $G' = G_{\Sigma^*}$ becomes $G' = G_{\Sigma^*}||$SpecRCVR=SpecRCVR.

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

It is also easy to find that K_3 is not co-observable w.r.t. $L(G')$=L(SpecRCVR). Counter example: $t = g \in \overline{K_3} \cap L(G')$, $\sigma = g \in \Sigma_{c,1}$, $t\sigma = gg \in L(G')\backslash\overline{K_3}$, $t' = gr_{s_0}p$, $P_1(t) = P_1(t') = g$, $t'\sigma = gr_{s_0}pg \in \overline{K_3} \cap L(G')$. For controller one, it cannot distinguish between $t\sigma = gg$ and $t'\sigma = gr_{s_0}pg$, 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 string $t\sigma = gg$ 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 3.5. Then G'=SpecRCVR becomes G'=SpecRCVR||SpecSNDR.

Step 3.5, verify whether K_3 is co-observable w.r.t. $L(G')$=L(SpecRCVR||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(SpecRCVR|| SpecSNDR). Counter example: $t = gs_0r_{a_0} \in \overline{K_3} \cap L(G')$, $\sigma = g \in \Sigma_{c,1}$, $t\sigma = gs_0r_{a_0}g \in L(G')\backslash\overline{K_3}$, $t' = gs_0r_{s_0}pa_0r_{a_0}$, $P_1(t) = P_1(t') = gs_0r_{a_0}$, $t'\sigma = gs_0r_{s_0}pa_0r_{a_0}g \in \overline{K_3} \cap L(G')$. For controller one, it cannot distinguish between $t\sigma = gs_0r_{a_0}g$ and $t'\sigma = gs_0r_{s_0}pa_0r_{a_0}g$, 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 string $t\sigma$=$gs_0r_{a_0}g$ is accepted by all the other components. It can be found that only the plant subsystem CHANNEL does not accept this counter example. Thus G'=SpecRCVR||SpecSNDR becomes G'=SpecRCVR||SpecSNDR|| CHANNEL.

Step 3.7: verify whether K_3 is co-observable w.r.t. $L(G')$=L(SpecRCVR||SpecSNDR|| CHANNEL)=$K_2||K_1||L(G_3)$.

It can be found that K_3 is co-observable w.r.t. $L(G')$=$K_2||K_1||L(G_3)$.

Since K_2 and K_1 are both co-observable w.r.t. $L(G)$, we thus conclude that $K = K_1||K_2||K_3$ is co-observable w.r.t. $L(G)$ by Propositions 3.5 and 3.6.

In the above example, we verify each specification component individually, and we never have to use more than a single plant component at a time. In addition, only one out of three plant components needed to be considered. Further, the complete plant does not need to be composed together.

6 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

[1] G. Barrett and S. Lafortune. Decentralized supervisory control with communicating controllers. *Automatic Control, IEEE Transactions on*, 45(9):1620–1638, 2000.

[2] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson. A note on reliable full-duplex transmission over half-duplex links. *Communications of the ACM*, 12(5):260–261, 1969.

[3] B. A. Brandin, R. Malik, and P. Malik. Incremental verification and synthesis of discrete-event systems guided by counter examples. *Control Systems Technology, IEEE Transactions on*, 12(3):387–401, 2004.

[4] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems (second edition)*. Springer, 2008.

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

[6] H. Flordal and R. Malik. Compositional verification in supervisory control. *SIAM Journal on Control and Optimization*, 48(3):1914–1938, 2009.

[7] H. Flordal, R. Malik, M. Fabian, and K. Åkesson. Compositional synthesis of maximally permissive supervisors using supervision equivalence. *Discrete Event Dynamic Systems*, 17(4):475–504, 2007.

[8] B. T. Hailpern. *Verifying concurrent processes using temporal logic*, volume 129. Springer, 1982.

[9] J. Y. Halpern and L. D. Zuck. A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols. *Journal of the ACM (JACM)*, 39(3):449–478, 1992.

[10] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Info. Sci.*, 44:173–198, 1988.

[11] W. C. Lynch. Reliable full-duplex file transmission over half duplex telephone lines. *Communications of the ACM*, 11(6):407–410, 1968.

[12] P. N. Pena, J. E. Cury, and S. Lafortune. Testing modularity of local supervisors: An approach based on abstractions. In *Discrete Event Systems, 2006 8th International Workshop on*, pages 107–112. IEEE, 2006.

[13] K. Rudie. *Decentralized Control of Discrete-event Systems.* PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont., 1992.

[14] K. Rudie and J. C. Willems. The computational complexity of decentralized discrete-event control problems. *Automatic Control, IEEE Transactions on*, 40(7):1313–1319, 1995.

[15] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *Automatic Control, IEEE Transactions on*, 37(11):1692–1708, 1992.

[16] A. S. Tanenbaum. *Computer Networks (fourth edition).* Englewood Cliffs, NJ: Prentice Hall, 2003.