SCISPACE
formerly Typeset

Open access • Journal Article • DOI:10.1109/9.400469

The computational complexity of decentralized discrete-event control problems

— Source link

Karen Rudie, Jan C. Willems

Institutions: Queen's University, University of Minnesota

Published on: 01 Jul 1995 - IEEE Transactions on Automatic Control (IEEE-INST ELECTRICAL ELECTRONICS ENGINEERS INC)

Topics: Time complexity, Computational complexity theory, Supervisory control and Decidability

Related papers:

- Think globally, act locally: decentralized supervisory control
- Supervisory control of a class of discrete event processes
- Introduction to Discrete Event Systems
- A General Architecture for Decentralized Supervisory Control of Discrete-Event Systems
- Supervisory control of discrete-event processes with partial observations

Share this paper:

View more about this paper here: https://typeset.io/papers/the-computational-complexity-of-decentralized-discrete-event-4izhkoie1t

The Computational Complexity of Decentralized Discrete-Event Control Problems

Karen Rudie*
Institute for Mathematics
and its Applications
University of Minnesota
514 Vincent Hall
Minneapolis, MN 55455
USA
email: rudie@ima.umn.edu

Jan C. Willems
Mathematics Institute
University of Groningen
P.O. Box 800
9700 AV Groningen
The Netherlands
email: J.C.Willems@math.rug.nl

February 11, 1993

Abstract

Computational complexity results are obtained for *decentralized* discrete-event system problems. These results generalize the earlier work of Tsitsiklis, who showed that for *centralized* supervisory control problems (under partial observation), solution existence is decidable in polynomial time for a special type of problem but becomes computationally intractable for the general class. As in the case of centralized control, there is no polynomial-time algorithm for producing supervisor solutions.

Keywords: discrete-event systems, computational complexity, decentralized supervisory control

1 Introduction

Supervisory control theory has been used in the past decade to model control problems that arise from discrete-event processes. Problems associated with centralized (as opposed to distributed) discrete-event systems have been more extensively explored [RW82], [WR87], [LW88], [CM89], [Kro87], [Laf88] and an application within semiconductor manufacturing [Bal91], [HSF91] provides a compelling argument for considering this class of problems as useful in future engineering practise. More recently, decentralized control has been investigated and possible applications include flexible manufacturing systems [LW90] and communication systems [CDFV88], [RW90], [RW92a]. At this

*On one-year leave from the Department of Electrical Engineering, Queen's University, Kingston, Ontario, Canada K7L 3N6.

point, examples of decentralized discrete-event systems control have primarily served a pedagogical and mathematical purpose and have been highly simplified versions of physically realistic applications. However, we believe that sufficient intuitive motivation has been given to justify further exploring the question "Given a discrete-event system, do there exist decentralized controllers that satisfy a certain objective?"

In this paper, we explore the computational complexity of a class of decentralized discrete-event problems. We generalize the results of Tsitsiklis [Tsi89]. These results indicate that testing for solvability within a restricted class of control problems can be done in polynomial time. However, even when a solution is proven to exist, there does not exist a polynomial-time algorithm for producing it. Moreover, if we move to the more general class of problems, we cannot even test for solvability in polynomial time, let alone produce supervisor solutions. The good news is that some interesting problems are captured by the restricted decentralized discrete-event control formulation. In particular, communication protocol verification is one such example.

We note that it is not, *a priori*, obvious that Tsitsiklis' results hold for the decentralized case. Just because the centralized version of a problem can be solved efficiently does not imply that its decentralized version is also solvable efficiently. For example, some NP-complete[1] multiprocessor scheduling problems become trivially solvable in polynomial-time when restricted to a single processor [GJ79].

2 Preliminaries

2.1 Supervisory Control Theory

We present (from [RW90] and [RW92b]) a problem formulation that describes a class of discrete-event systems subject to decentralized control. For more details on the formalities of supervisory control theory, the reader is referred to [RW82], [Ram83], [RW87], [WR87], [LW88], [Won88], [WR88], [LW90].

Consider a discrete-event process that can be characterized by an automaton

$$G = (Q, \Sigma, \delta, q_0, Q_m)$$

where Σ is a finite alphabet of event labels (and represents the set of all possible events that can occur within the system), Q is a set of states, $q_0 \in Q$ is the initial state, $Q_m \subseteq Q$ is the set of terminal (often called *marker*) states and $\delta : \Sigma \times Q \longrightarrow Q$, the transition function, is a partial function defined at each state in Q for a subset of Σ. When Q is finite, G can be represented by a directed graph whose nodes are the states in Q and whose edges are transitions defined by δ and

[1]The class of problems solvable in polynomial time is called P and the class solvable in nondeterministic polynomial time (i.e., by a Turing machine that is permitted to makes "guesses" before certain moves) is called NP. It is is known that P $\subseteq$ NP and it is widely conjectured that the inclusion is proper, i.e, that P $\neq$ NP. Therefore, problems in NP are considered to be computationally intractable. An NP-complete problem is a problem which is in NP and to which all other problems in NP can be reduced via a polynomial-time transformation. That is, NP-complete problems are considered the hardest problems in NP. If the conjecture that P $\neq$ NP is true, then NP-complete problems are not solvable in polynomial time.

labeled by elements from Σ. The automaton G describes the behaviour of a discrete-event process if we interpret transitions as event occurrences that take the process from state to state.

Sequences of concatenated symbols from Σ are interpreted as sequences of events, called *strings*. Let Σ^* denote the set of all finite strings over Σ including the null string ε. Then the transition function δ can be extended to $\Sigma^* \times Q \longrightarrow Q$ by defining $\delta(\varepsilon, q) := q$ and for $s \in \Sigma^*, \sigma \in \Sigma$, $\delta(s\sigma, q) := \delta(\sigma, \delta(s, q))$. That is, we now think of δ as indicating to which state (or states) a *sequence* of events will lead. A subset of Σ^* is called a *language*. The behaviour of the uncontrolled process G, which we also call a *plant*, is given by two languages. The *closed behaviour* of G, written $L(G)$, is the language defined as

$$L(G) := \{s \mid s \in \Sigma^* \; and \; \delta(s, q_0) \; is \; defined\}$$

and is interpreted to mean the set of all possible event sequences which the plant could generate. The *marked behaviour* of G, written $L_m(G)$, is the language defined as

$$L_m(G) := \{s \mid s \in \Sigma^* \; and \; \delta(s, q_0) \in Q_m\}$$

and is intended to distinguish some subset of possible plant behaviour as representing completed tasks.

To impose supervision on the plant, we identify some of its events as *controllable* and the rest as *uncontrollable*, thereby partitioning Σ into the disjoint sets Σ_c, the set of controllable events, and Σ_{uc}, the set of uncontrollable events. Controllable events are those which an external agent may enable (permit to occur) or disable (prevent from occurring) while uncontrollable events are those which cannot be prevented from occurring and are therefore considered to be permanently enabled. The event set Σ is also partitioned into disjoint sets Σ_o and Σ_{uo} of *observable* and *unobservable* events, respectively. Observable events are those which an external agent may observe during the course of tracking the plant. A *supervisor* (sometimes called a *controller*) is then an agent which observes subsequences of the sequences of events generated by G and enables or disables any of the controllable events at any point in time throughout its observation. By performing such a manipulation of controllable events, the supervisor ensures that only a subset of $L(G)$ is permitted to occur. Formally, a *supervisor* $\mathcal{S}$ is a pair (T, ψ) where T is an automaton which recognizes a language over the same event set as the plant G, i.e.,

$$T = (X, \Sigma, \xi, x_0, X_m)$$

where X is the set of states, ξ is the transition function, x_0 is the initial state and X_m are the marker states of the supervisor. The mapping $\psi : \Sigma \times X \rightarrow \{enable, disable\}$, called a *feedback map*, satisfies

$$\psi(\sigma, x) = enable, \text{ if } \sigma \in \Sigma_{uc}, \; x \in X$$

and

$$\psi(\sigma, x) \in \{enable, disable\}, \text{ if } \sigma \in \Sigma_c, \; x \in X.$$

The automaton T is constrained so that

$$\sigma \in \Sigma_{uo}, x \in X \Longrightarrow \xi(\sigma, x) = x.$$

The automaton T tracks the behaviour of G. It changes state according to the observable events generated by G and, in turn, at each state x of T, the control rule $\psi(\sigma, x)$ dictates whether σ is to be enabled or disabled at the corresponding state of G.

The sequences of events generated while the plant G is under the control of $\mathcal{S} = (T, \psi)$ characterizes the behaviour of the *closed-loop system* and is represented by an automaton $\mathcal{S}/G$ whose closed behaviour, denoted by $L(\mathcal{S}/G)$, permits a string to be generated if the string is in both G and T and if each event in the string is enabled by ψ. The marked behaviour of the closed-loop system is denoted by $L_m(\mathcal{S}/G)$ and consists of those strings in $L(\mathcal{S}/G)$ that are marked by both G and S. Formally, the automaton $\mathcal{S}/G$ is given by

$$\mathcal{S}/G := (Q \times X, \Sigma, (\delta \times \xi)^{\psi}, (q_0, x_0), Q_m \times X_m)$$

where $(\delta \times \xi)^{\psi} : \Sigma \times Q \times X \longrightarrow Q \times X$ is defined by

$$(\delta \times \xi)^{\psi}(\sigma, q, x) := \begin{cases} (\delta(\sigma, q), \xi(\sigma, x)) & \text{if both } \delta(\sigma, q),\, \xi(\sigma, x) \text{ are defined} \\ & \text{and } \psi(\sigma, x) = enable \\ undefined & \text{otherwise.} \end{cases}$$

Often it is important to find supervisors that guarantee that the closed-loop system is *nonblocking*, i.e., that every string generated by the closed-loop system can be completed to a marked string in the system. This requirement is expressed as follows: a supervisor $\mathcal{S}$ is *proper for G* if

$$\overline{L_m(\mathcal{S}/G)} = L(\mathcal{S}/G)$$

where $\overline{K}$ denotes the prefix-closure of a language K.

Typically, control problems require finding for a given plant a supervisor (or set of supervisors) such that the closed-loop system satisfies some prescribed desirable behaviour. Representative centralized supervisory control problems can be found in [RW82], where it is assumed that all events are observable, and [LW88], where it is assumed that some events may not be observable. When controllers act on a given plant, we say that the closed-loop behaviour is *synthesized* by the controllers. Then, control problems involve examining under what conditions prescribed behaviours can be synthesized.

Now, we consider the situation where the physical requirements of a problem dictate that decentralized control be used. When a supervisor may act on any controllable event in the entire event set, we say that the supervisor is *global*; in contrast, a supervisor which can only control some subset of controllable events is said to be *local*. A decentralized solution prescribes the actions that two or more local supervisors may take. In this paper, we consider the case of two local supervisors.

The decentralized control problem presented below requires the following definitions. For supervisors $\mathcal{S}_1 = (T_1, \phi)$ and $\mathcal{S}_2 = (T_2, \psi)$ acting on G with $T_1 = (X, \Sigma, \xi, x_0, X_m)$ and $T_2 = (Y, \Sigma, \eta, y_0, Y_m)$, the *conjunction* of $\mathcal{S}_1$ and $\mathcal{S}_2$ is the supervisor

$$\mathcal{S}_1 \wedge \mathcal{S}_2 := (T_1 \times T_2, \phi * \psi)$$

defined by

$$T_1 \times T_2 := (X \times Y, \Sigma, \xi \times \eta, (x_0, y_0), X_m \times Y_m)$$

with $\sigma \in \Sigma,\ x \in X,\ y \in Y \Longrightarrow$

$$(\xi \times \eta)(\sigma, x, y) := \begin{cases} (\xi(\sigma, x), \eta(\sigma, y)) & \text{if both } \xi(\sigma, x) \text{ and } \eta(\sigma, y) \text{ are defined} \\ undefined & \text{otherwise} \end{cases}$$

$$(\phi * \psi)(\sigma, x, y) := \begin{cases} disable & \text{if either } \phi(\sigma, x) = disable \text{ or } \psi(\sigma, y) = disable \\ enable & \text{otherwise.} \end{cases}$$

That is, $T_1 \times T_2$ recognizes the intersection of the languages recognized by T_1 and T_2 and $\phi * \psi$ disables an event if and only if either ϕ or ψ disables it. Thus, $\mathcal{S}_1 \wedge \mathcal{S}_2$ models the actions of $\mathcal{S}_1$ and $\mathcal{S}_2$ operating in parallel. It can be shown [WR88] that $L(\mathcal{S}_1 \wedge \mathcal{S}_2/G) = L(\mathcal{S}_1/G) \cap L(\mathcal{S}_2/G)$ and $L_m(\mathcal{S}_1 \wedge \mathcal{S}_2/G) = L_m(\mathcal{S}_1/G) \cap L_m(\mathcal{S}_2/G)$.

Given a local supervisor $\mathcal{S}$ that controls some subset $\Sigma_{loc,c}$ of Σ_c while observing some subset $\Sigma_{loc,o}$ of Σ, $\tilde{\mathcal{S}}$ denotes the supervisor which takes the same control action as $\mathcal{S}$ on $\Sigma_{loc,c}$, enables all events in $\Sigma \setminus \Sigma_{loc,c}$, makes the same transitions as $\mathcal{S}$ on $\Sigma_{loc,o}$ and stays at the same state for events in $\Sigma \setminus \Sigma_{loc,o}$. The supervisor $\tilde{\mathcal{S}}$ is called the *global extension* of $\mathcal{S}$ (since $\tilde{\mathcal{S}}$ acts on all of Σ while $\mathcal{S}$ acts only on a subset of Σ).

We now introduce our main decentralized control problem formulation:

Decentralized Control Problem *Given a plant G over an alphabet Σ, a language $E \subseteq L_m(G)$, a language $A \subseteq E$, and sets $\Sigma_{1,c}, \Sigma_{2,c}, \Sigma_{1,o}, \Sigma_{2,o} \subseteq \Sigma$, construct local supervisors $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\tilde{\mathcal{S}}_1 \wedge \tilde{\mathcal{S}}_2$ is a proper supervisor for G and such that*

$$A \subseteq L(\tilde{\mathcal{S}}_1 \wedge \tilde{\mathcal{S}}_2/G) \subseteq E.$$

Here, for $i = 1, 2$, supervisor $\mathcal{S}_i$ can observe only events in $\Sigma_{i,o}$ and control only events in $\Sigma_{i,c}$ and where $\tilde{\mathcal{S}}_i$ is the global extension of $\mathcal{S}_i$. The set of uncontrollable events, Σ_{uc} is understood to be $\Sigma \setminus (\Sigma_{1,c} \cup \Sigma_{2,c})$.

The language E embodies the system designer's notion of *legal* or desirable behaviour while A specifies the behaviour common to any acceptable solution, i.e., the *minimally adequate* behaviour. That is, any solution must exhibit at least the behaviour described by A and no more than that described by E.

The above problem can be solved by first considering the special case where the range of desirable behaviour is narrowed to a single language, i.e., where $A = E$. This case was first solved in [CDFV88], provided A and E are prefix-closed. The solution is conveniently described using the notions of controllability and co-observability defined in [RW82] and [RW92b], respectively. A language $K \subseteq L(G)$ is *controllable w.r.t. G* if

$$\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K}$$

where for any languages L and M, the notation LM stands for $\{st \mid s \in L \wedge t \in M\}$. If we interpret $L(G)$ as physically possible behaviour and K as legal behaviour, an informal description of controllability is that K is controllable if for any sequence of events s that starts out as a legal sequence ($s \in \overline{K}$), the occurrence of an uncontrollable event ($\sigma \in \Sigma_{uc}$) which is physically possible ($s\sigma \in L(G)$) does not lead the sequence out of the legal range ($s\sigma \in \overline{K}$).

Given any event set Σ, we may associate with it a mapping, called the *canonical projection*, which we interpret as a supervisor's view of the strings in Σ^*. The projection $P : \Sigma^* \longrightarrow \Sigma_o^*$ is defined as follows: $P(\varepsilon) := \varepsilon$ and for $s \in \Sigma^*, \sigma \in \Sigma$, $P(s\sigma) := P(s)P(\sigma)$, i.e., P erases all unobservable events. If we have a string s generated by the plant, then $P(s)$ is the sequence of events that an external agent observes.

Now we recall the notion of co-observability. Given a plant G over alphabet Σ, sets $\Sigma_{1,c}, \Sigma_{2,c}, \Sigma_{1,o}, \Sigma_{2,o} \subseteq \Sigma$, projections $P_1 : \Sigma^* \longrightarrow \Sigma_{1,o}^*$, $P_2 : \Sigma^* \longrightarrow \Sigma_{2,o}^*$, a language $K \subseteq L_m(G)$ is *co-observable w.r.t.* G, P_1, P_2 if

$$s, s', s'' \in \Sigma^*,\ P_1(s) = P_1(s'),\ P_2(s) = P_2(s'') \Longrightarrow$$

$$\begin{array}{lll} & (\forall \sigma \in \Sigma_{1,c} \cap \Sigma_{2,c})\ s \in \overline{K} \wedge s\sigma \in L(G) \wedge s'\sigma, s''\sigma \in \overline{K} \Longrightarrow s\sigma \in \overline{K} & \textbf{conjunct 1} \\ \wedge & (\forall \sigma \in \Sigma_{1,c} \setminus \Sigma_{2,c})\ s \in \overline{K} \wedge s\sigma \in L(G) \wedge s'\sigma \in \overline{K} \Longrightarrow s\sigma \in \overline{K} & \textbf{conjunct 2} \\ \wedge & (\forall \sigma \in \Sigma_{2,c} \setminus \Sigma_{1,c})\ s \in \overline{K} \wedge s\sigma \in L(G) \wedge s''\sigma \in \overline{K} \Longrightarrow s\sigma \in \overline{K} & \textbf{conjunct 3} \\ \wedge & s \in \overline{K} \cap L_m(G) \wedge s', s'' \in K \Longrightarrow s \in K. & \textbf{conjunct 4} \end{array}$$

Intuitively, a supervisor knows what action to take if it knows what sequence of events actually occurred. However, a string which, for each supervisor, looks like (i.e., has the same projection as) another string may be potentially ambiguous in determining control action. On this basis, if we assume that some external agent, such as a supervisor, determines which strings are allowed to be in $\overline{K}$ and which in K, an informal description of co-observability is as follows. A language K is co-observable if (1) after the occurrence of an ambiguous string, s, in $\overline{K}$, the decision to enable or disable a controllable event σ is forced by the action that a supervisor which can control σ would take on other strings which look like s (encompassed by conjuncts (1)–(3) in the definition of co-observability), and (2) the decision to mark or not mark a potentially confusing string is determined by at least one of the supervisors (covered by conjunct (4)). Note that if a language K is prefix-closed, then conjunct (4) always holds. The reader is referred to [RW92b] for more details on co-observability.

The solution to our decentralized control problem for the special case where $A = E$ is as follows. There exist supervisors $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $L_m(\hat{\mathcal{S}_1} \wedge \hat{\mathcal{S}_2}/G) = E$ and $\hat{\mathcal{S}_1} \wedge \hat{\mathcal{S}_2}$ guarantees nonblocking if and only if E is controllable and co-observable w.r.t. the plant G [CDFV88], [RW92b]. In [RW92b] it is shown that if G is finite-state, E is a prefix-closed, regular language and $E \neq \emptyset$, then there is a computable procedure for determining if E is controllable and co-observable w.r.t. G.

The solution to the decentralized control problem when $A \neq E$ requires computing the infimal prefix-closed, controllable and co-observable language containing A and checking if that language is in E. A procedure for checking this condition and for constructing finite-state supervisors, when the condition holds, was given in [RW92b].

2.2 Computational Complexity

In this paper, we are interested in asymptotic complexity, which is a way of measuring worst-case behaviour. The asymptotic computing time of an algorithm indicates how the time needed to perform the algorithm increases as a function of the inputs. Throughout the paper, the notation $O(\cdot)$ is used to describe the asymptotic complexity of an algorithm in time. Using the definition in

[HS82], for some input parameter n, to say that a function $f(n)$ requires $O(g(n))$ time means that for some N and some c, $|f(n)| < c|g(n)|$ for all $n \geq N$, where $f(n)$ represents the time required by an algorithm as a function of the input n. The parameter n is a way of characterizing the size of the inputs; e.g., it could be the number of inputs or the sum of some inputs. This notation, pronounced "big-oh of $g(n)$", is a quantitative way of giving an upper bound on how fast an algorithm increases as a function of its input parameters. Big-oh notation allows general trends to be observed by factoring out the value of the multiple of the bounding function and by disregarding what happens to $f(n)$ for small values of n.

If an algorithm takes $O(g(n))$ time where $g(n)$ is a polynomial, we say that the algorithm is "polynomial time". Using the definition in [GJ79], an algorithm that cannot be bounded by a polynomial is said to be "exponential time".[2] Problems that are not solvable in polynomial time are considered to be computationally highly inefficient—in essence, infeasible.

3 Computing Solutions to Decentralized Control Problems

Recall from Section 2 that, given some fixed plant, to synthesize a desirable language, that language must be controllable and co-observable with respect to the plant. So, to solve the special case of the Decentralized Control Problem given in Section 2, we must be able to check whether a language is controllable and co-observable. It was shown in [WR88] that controllability of E w.r.t. G can be decided in polynomial time with respect to the number of states of G and the automaton representation of E. However, the computing time taken by the procedure given in [RW92b] to check for controllability together with co-observability is exponential in the number of states (of G and E). It was shown in [RW92a] that the special case of the Decentralized Control Problem (with $A = E$) can be used to check partial correctness (i.e., to verify safety properties) of communication protocols. Given the widespread interest in protocol verification in the communications protocols community, it is worth asking if there is a more efficient algorithm for checking whether a language is co-observable than the one we have in [RW92b]. In this section we demonstrate that, in fact, a polynomial-time algorithm can be found. This extends the results given in [Tsi89], where it was shown that *observability*, the centralized counterpart to co-observability, can be decided in polynomial time.

The utility of the more general formulation of the Decentralized Control Problem (where $A \neq E$) has been less apparent. It serves as a natural model for formulating some communication problems and, in a limited way, for synthesizing protocols [RW90]. However, as will be discussed further on, there is no polynomial-time algorithm for solving all problems within the general class given by the Decentralized Control Problem.

[2]In addition to functions of 2^n, this includes non-polynomial time complexity functions, such as $n^{\log n}$, which are not otherwise regarded as exponential functions.

3.1 Deciding Co-observability

We are going to show that, given finite-state automata, G and E, co-observability of the language $L(E)$ w.r.t. G can be decided by examining an automaton M constructed from G and E.[3] In this section, we also assume that the legal language is prefix-closed, so blocking is not an issue here. Informally, the paths in M will keep track of strings that violate the co-observability of $L(E)$ w.r.t. G.

Given $G = (Q^G, \Sigma, \delta^G, q_0^G, Q^G)$ and $E = (Q^E, \Sigma, \delta^E, q_0^E, Q^E)$, we construct an automaton $M(G, E)$ (or M for short) as follows. Assuming that G and E are finite-state, there exists some element not in $Q^G \cup Q^E$; let us denote by d (for "dump") one such element. Then,

$$M := (Q^M, \Sigma, \delta^M, q_0^M, Q_m^M)$$

where

$$\begin{aligned} Q^M &:= Q^E \times Q^E \times Q^E \times Q^G \cup \{d\} \\ q_0^M &:= (q_0^E, q_0^E, q_0^E, q_0^G) \\ Q_m^M &:= \{d\} \end{aligned}$$

and δ^M is defined below. In the definition of δ^M, we refer to the following set of conditions:

$$\left.\begin{array}{ll} \delta^E(\sigma, q_3) & \text{is not defined} \\ \delta^G(\sigma, q_4) & \text{is defined} \\ \delta^E(\sigma, q_1) & \text{is defined if } \sigma \in \Sigma_{1,c} \\ \delta^E(\sigma, q_2) & \text{is defined if } \sigma \in \Sigma_{2,c}. \end{array}\right\} (*)$$

Now, δ^M is given by listing all the transitions it defines. We make a slight abuse of notation and label transitions by events from Σ together with a number from 0 to 6; the numbers serve to distinguish different transitions that have the same event label. We call the number in the pair labeling a transition its *transition type*. We may alternately identify a transition in M by a pair consisting of an event plus its transition type, or simply by the event label itself; similarly, we identify paths in M by sequences of pairs $(\sigma_1, i_1) \cdots (\sigma_n, i_n)$, or simply by sequences of events $\sigma_1 \cdots \sigma_n$, depending on whether we need to specify the transition type of each event in the path. The partial transition function δ^M is defined as follows:

For $\sigma \notin \Sigma_{1,o}, \sigma \notin \Sigma_{2,o}$,

$$\begin{array}{l} (q_1, q_2, q_3, q_4) \overset{(\sigma,1)}{\longmapsto} (\delta^E(\sigma, q_1), q_2, q_3, q_4) \\ (q_1, q_2, q_3, q_4) \overset{(\sigma,2)}{\longmapsto} (q_1, \delta^E(\sigma, q_2), q_3, q_4) \\ (q_1, q_2, q_3, q_4) \overset{(\sigma,3)}{\longmapsto} (q_1, q_2, \delta^E(\sigma, q_3), \delta^G(\sigma, q_4)) \\ (q_1, q_2, q_3, q_4) \overset{(\sigma,4)}{\longmapsto} (\delta^E(\sigma, q_1), \delta^E(\sigma, q_2), \delta^E(\sigma, q_3), \delta^G(\sigma, q_4)) \\ (q_1, q_2, q_3, q_4) \overset{(\sigma,0)}{\longmapsto} d \ \text{ if } \ (*) \end{array}$$

[3]In this section, we use E to denote an automaton and $L(E)$ to denote the language generated by it. Our algorithm takes finite-state automata, not languages, as inputs so we consider only the case where the legal language is generated by a finite-state machine.

For $\sigma \notin \Sigma_{1,o}, \sigma \in \Sigma_{2,o}$,

$$(q_1, q_2, q_3, q_4) \overset{(\sigma,1)}{\longmapsto} (\delta^E(\sigma, q_1), q_2, q_3, q_4)$$
$$(q_1, q_2, q_3, q_4) \overset{(\sigma,5)}{\longmapsto} (q_1, \delta^E(\sigma, q_2), \delta^E(\sigma, q_3), \delta^G(\sigma, q_4))$$
$$(q_1, q_2, q_3, q_4) \overset{(\sigma,4)}{\longmapsto} (\delta^E(\sigma, q_1), \delta^E(\sigma, q_2), \delta^E(\sigma, q_3), \delta^G(\sigma, q_4))$$
$$(q_1, q_2, q_3, q_4) \overset{(\sigma,0)}{\longmapsto} d \ \text{ if } \ (*)$$

For $\sigma \in \Sigma_{1,o}, \sigma \notin \Sigma_{2,o}$,

$$(q_1, q_2, q_3, q_4) \overset{(\sigma,2)}{\longmapsto} (q_1, \delta^E(\sigma, q_2), q_3, q_4)$$
$$(q_1, q_2, q_3, q_4) \overset{(\sigma,6)}{\longmapsto} (\delta^E(\sigma, q_1), q_2, \delta^E(\sigma, q_3), \delta^G(\sigma, q_4))$$
$$(q_1, q_2, q_3, q_4) \overset{(\sigma,4)}{\longmapsto} (\delta^E(\sigma, q_1), \delta^E(\sigma, q_2), \delta^E(\sigma, q_3), \delta^G(\sigma, q_4))$$
$$(q_1, q_2, q_3, q_4) \overset{(\sigma,0)}{\longmapsto} d \ \text{ if } \ (*)$$

For $\sigma \in \Sigma_{1,o}, \sigma \in \Sigma_{2,o}$,

$$(q_1, q_2, q_3, q_4) \overset{(\sigma,4)}{\longmapsto} (\delta^E(\sigma, q_1), \delta^E(\sigma, q_2), \delta^E(\sigma, q_3), \delta^G(\sigma, q_4))$$
$$(q_1, q_2, q_3, q_4) \overset{(\sigma,0)}{\longmapsto} d \ \text{ if } \ (*)$$

For $\sigma \in \Sigma$, $\delta^M(\sigma, d)$ is undefined.

Note that the automaton M thus defined is a nondeterministic finite automaton since a single event may lead to several states.

We are going to show that in the large automaton M, each 4-tuple labeling a state (q_1, q_2, q_3, q_4) informally keeps track of strings as follows: for some $s, s', s'' \in \Sigma^*$, $\sigma \in \Sigma$, the sequence $s'\sigma$ leads to q_1, the sequence $s''\sigma$ leads to q_2, the sequence s leads to q_3 and $s\sigma$ leads to q_4. The states q_1, q_2, q_3, and q_4 (respectively) are then used to determine if $s'\sigma \in L(E)$, $s''\sigma \in L(E)$, $s \in L(E)$ and $s\sigma \in L(G)$ (resp.). In this way, we can track through M to see if co-observability fails.

Our proof is in the same spirit as the proof in [Tsi89] where the corresponding centralized result is given. However, there, given G and E, a game was devised such that there exists a winning strategy to the game if and only if $L(E)$ is not co-observable w.r.t G. In our proof, the automaton M is constructed such that M recognizes a nonempty language if and only if $L(E)$ is not co-observable w.r.t G.

Proposition 3.1 *Given automata E and G, the language $L(E)$ is not co-observable w.r.t. G iff $M(G, E)$ recognizes a nonempty language, i.e., iff there is a path in M from the initial state to the dump state d.*

Proof ($\Longleftarrow$) Assume that there is a path in M from the initial state to the dump state. Since the initial state is not the dump state, there is a sequence $s^M\sigma$ for some $s^M \in \Sigma^*$, $\sigma \in \Sigma$ such that $s^M\sigma$ starts at initial state of M and ends at the dump state. Without loss of generality, assume that s^M does not end at the dump state. (We can make this assumption since if s^M ends at the dump state then we can pick the largest prefix of s^M that doesn't end at the dump and there exists at least one such prefix since the initial state is not equal to the dump state.) Suppose that s^M ends at state (q_1, q_2, q_3, q_4) of M.

We produce three strings s, s', s'' constructed from s^M and show that these strings are a counterexample to co-observability. The strings s, s', s'' are formed by projecting out certain events from s^M. The appropriate projections are given by the following operators $F_1, F_2, F_3 : L(M) \to \Sigma^*$, defined as follows:

For $\sigma \in \Sigma$ such that (σ, i) is some transition defined by δ^M,

$$F_1(\sigma) \;:=\; \begin{cases} \sigma & \text{if } i = 1,4,6 \\ \varepsilon & \text{otherwise} \end{cases}$$

$$F_2(\sigma) \;:=\; \begin{cases} \sigma & \text{if } i = 2,4,5 \\ \varepsilon & \text{otherwise} \end{cases}$$

$$F_3(\sigma) \;:=\; \begin{cases} \sigma & \text{if } i = 3,4,5,6 \\ \varepsilon & \text{otherwise.} \end{cases}$$

For $i = 1,2,3$, $F_i(\varepsilon) := \varepsilon$ and for $s \in \Sigma^*$, $\sigma \in \Sigma$, $F_i(s\sigma) := F_i(s)F_i(\sigma)$.

The projection operator F_1 records only those transitions in a sequence in $L(M)$ in which there is also a transition in the first argument of the current 4-tuple state. That is, for a sequence in M that leads to state (q_1, q_3, q_3, q_4), the event σ in the pair labeling the next transition is not erased by F_1 if that transition leads to a 4-tuple state whose first argument is $\delta^E(\sigma, q_1)$. Similarly, F_2 records only those transitions in which there is a transition in the second argument of the 4-tuple state. The operator F_3 records transitions in the third and fourth arguments.

Now we define sequences s, s', s'':

$$\begin{aligned} s &:= F_3(s^M) \\ s' &:= F_1(s^M) \\ s'' &:= F_2(s^M). \end{aligned}$$

The fact that s, s', s'' lead to a counterexample to co-observability will follow almost immediately from the following claims.

The 4-tuples of M are constructed to keep track of sequences s, s', s'' such that $P_1(s) = P_1(s')$ and $P_2(s) = P_2(s'')$. Think of the evolution of the first argument in the sequences of the 4-tuple states as describing the string s', the second argument as describing the string s'' and the third and fourth arguments as describing the string s (with the third being used to track the movement of s through E and the fourth to track its movement through G). Then the filter F_1 picks out those events in s^M in which a transition (possibly a self-loop) occurs because of an event occurrence in

s'; F_2 picks out those events in s^M in which a transition occurs because of an event in s''; and F_3 picks out those events in s^M in which a transition occurs because of an event in s.

The first claim allows us to track through M and keep a record of where s leads to in E and where s leads to in G. Recall, we are trying to show that $s\sigma \in L(G)$ and that $s\sigma \notin L(E)$ (to get co-observability to fail).

Claim 1 *For $s^M \in L(M) \setminus L_m(M)$, $F_3(s^M) \in L(E)$ and if s^M leads to state (q_1, q_2, q_3, q_4) in M then $F_3(s^M)$ leads to state q_3 in E and $F_3(s^M)$ leads to state q_4 in G.*

Proof We proceed by induction on length of s^M. We use the notation $|t|$ to denote the length of a string t. We first consider only where the sequence $F_3(s^M)$ leads to in E.

Basis Suppose that $|s^M| = 0$, i.e., $s^M = \varepsilon$. Then s^M leads to state $(q_0^E, q_0^E, q_0^E, q_0^G)$ and $F_3(s^M) = F_3(\varepsilon) = \varepsilon$, which leads to state q_0^E in E.

Inductive Hypothesis Suppose that for all s^M such that $|s^M| \leq n$, $F_3(s^M) \in L(E)$ and if s^M leads to state (q_1, q_2, q_3, q_4) in M then $F_3(s^M)$ leads to state q_3 in E. Now, take $s^M = \overline{s}\,\overline{\sigma}$ where $|\overline{s}| = n$. Suppose that $\overline{s}$ leads to state (q_1, q_2, q_3, q_4). By the inductive hypothesis, $F_3(\overline{s}) \in L(E)$ and $F_3(\overline{s})$ leads to state q_3 in E.

> **Case 1: $\boldsymbol{F_3(\overline{\sigma}) = \varepsilon}$.** Then $F_3(\overline{s}\,\overline{\sigma}) = F_3(\overline{s})$ and so $F_3(\overline{s}\,\overline{\sigma}) \in L(E)$. Moreover, if $F_3(\overline{\sigma}) = \varepsilon$ then $\overline{\sigma}$ results from a transition of type 1 or 2 and for both these types of transitions , the third argument in the 4-tuple state, (q_1, q_2, q_3, q_4), of M doesn't change so $\overline{s}\,\overline{\sigma}$ leads to (x_1, x_2, q_3, x_4) for some x_1, x_2, x_4. Since $F_3(\overline{s})$ leads to q_3 in E we also have $F_3(\overline{s}\,\overline{\sigma})$ leads to q_3 in E.
>
> **Case 2: $\boldsymbol{F_3(\overline{\sigma}) = \overline{\sigma}}$.** Then $\overline{\sigma}$ results from a transition of type 3, 4, 5, or 6. In all those cases, $\delta^E(\sigma, q_3)$ is defined. Since, by the inductive hypothesis, $F_3(\overline{s}) \in L(E)$ and $F_3(\overline{s})$ ends at q_3 in E, then $\delta^E(\sigma, q_3)$ being defined implies that $F_3(\overline{s})\overline{\sigma} \in L(E)$ and that $F_3(\overline{s})\overline{\sigma}$ ends at state $\delta^E(\sigma, q_3)$ (which is the third argument of the 4-tuple state, (q_1, q_2, q_3, q_4), of M to which s^M leads, regardless of whether transition $\overline{\sigma}$ results from a transition of type 3, 4, 5, or 6.

The argument regarding where $F_3(s^M)$ leads to in G follows easily because the fourth arguments in the 4-tuple states of M change when and only when the third arguments do. $\blacksquare_{\text{claim 1}}$

The next claim allows us to track through M and keep a record of where s' leads to in E.

Claim 2 *For $s^M \in L(M) \setminus L_m(M)$, $F_1(s^M) \in L(E)$ and if s^M leads to state (q_1, q_2, q_3, q_4) in M then $F_1(s^M)$ leads to state q_1 in E.*

Proof This is analogous to Claim 1 except that for Case 1 we argue that if $F_1(\overline{\sigma}) = \varepsilon$ then $\overline{\sigma}$ results from transition 2, 3 or 5 and in all those transitions , the first argument in the 4-tuple doesn't change. A similar argument to that used in Case 2 of Claim 1 holds when $F_1(\overline{\sigma}) \neq \varepsilon$. $\blacksquare_{\text{claim 2}}$

The next claim allows us to track through M and keep a record of where s'' leads to in E.

Claim 3 *For $s^M \in L_m(M) \setminus (M)$, $F_2(s^M) \in L(E)$ and if s^M leads to state (q_1, q_2, q_3, q_4) in M then $F_2(s^M)$ leads to state q_2 in E.*

Proof The proof is analogous to Claim 1 except that for Case 1 we argue that if $F_2(\overline{\sigma}) = \varepsilon$ then $\overline{\sigma}$ results from transition 1, 3 or 6 and in all those transitions , the second argument in the 4-tuple doesn't change. A similar argument to that used in Case 2 of Claim 1 holds when $F_1(\overline{\sigma}) \neq \varepsilon$.$\blacksquare_{\text{claim 3}}$

The following claim will allow us to show that our strings s and s' as defined above satisfy $P_1(s) = P_1(s')$.

Claim 4 *For $s^M \in L(M) \setminus L_m(M)$, $P_1(F_3(s^M)) = P_1(F_1(s^M))$.*

Proof We proceed by induction on length of s^M.

Basis If $|s^M| = 0$ then $s^M = \varepsilon$ and $F_3(\varepsilon) = F_1(\varepsilon) = \varepsilon$, by definition.

Inductive Hypothesis Suppose that for all s^M such that $|s^M| \leq n$, $P_1(F_1(s^M)) = P_1(F_3(s^M))$. Now, take $s^M = \overline{s}\,\overline{\sigma}$ where $|\overline{s}| = n$. Then,

$$\begin{aligned} P_1(F_1(\overline{s}\,\overline{\sigma})) &= P_1(F_1(\overline{s})F_1(\overline{\sigma})) \\ &= P_1(F_1(\overline{s}))P_1(F_1(\overline{\sigma})) \\ &= P_1(F_3(\overline{s}))P_1(F_1(\overline{\sigma})) \quad \text{(by the inductive hypothesis).} \end{aligned}$$

Case 1: $\boldsymbol{\overline{\sigma} \in \Sigma_{1,o} \cap \Sigma_{2,o}}$. Since $\overline{\sigma}$ doesn't lead to the dump state (because $\overline{s}\,\overline{\sigma} \notin L_m(M)$), it must be the result of a transition of type 4. By definition of F_1 and F_3, for a transition of type 4, $F_1(\overline{\sigma}) = F_3(\overline{\sigma}) = \overline{\sigma}$. So we have

$$\begin{aligned} P_1(F_1(\overline{s}\,\overline{\sigma})) &= P_1(F_3(\overline{s})P_1(F_3(\overline{\sigma})) \\ &= P_1(F_3(\overline{s}\,\overline{\sigma})). \end{aligned}$$

Case 2: $\boldsymbol{\overline{\sigma} \in \Sigma_{1,o} \setminus \Sigma_{2,o}}$. So $\overline{\sigma}$ must be the result of either a transition of type 2, 4 or 6. If $\overline{\sigma}$ is the result of a transition of type 2, then $F_1(\overline{\sigma}) = F_3(\overline{\sigma}) = \varepsilon$, by definition of F_1 and F_3. If $\overline{\sigma}$ is a result of either a transition of type 4 or 6, then $F_1(\overline{\sigma}) = F_3(\overline{\sigma}) = \overline{\sigma}$. In each of the three cases, $F_1(\overline{\sigma}) = F_3(\overline{\sigma})$.

Case 3: $\boldsymbol{\overline{\sigma} \in \Sigma_{2,o} \setminus \Sigma_{1,o}}$. Then since $F_1(\overline{\sigma})$ and $F_3(\overline{\sigma}) \in \{\varepsilon, \overline{\sigma}\}$ and since $\overline{\sigma} \notin \Sigma_{1,o}$, we have

$$P_1(F_1(\overline{\sigma})) = P_1(F_3(\overline{\sigma}) = \varepsilon.$$

Case 4: $\overline{\sigma} \notin \Sigma_{1,o} \cup \Sigma_{2,o}$. Same argument as for Case 3.

$\blacksquare_{\text{claim 4}}$

The following claim will allow us to show that our strings s and s'' as defined above satisfy $P_2(s) = P_2(s'')$.

Claim 5 *For $s^M \in L(M) \setminus L_m(M)$, $P_2(F_3(s^M)) = P_2(F_2(s^M))$.*

Proof The proof is similar to the proof of Claim 4 if we observe that for the case when $\overline{\sigma} \in \Sigma_{2,o} \setminus \Sigma_{1,o}$, $\overline{\sigma}$ results from either a transition of type 1, 4 or 5 and in all those cases $F_2(\overline{\sigma}) = F_3(\overline{\sigma})$. $\blacksquare_{\text{claim 5}}$

By Claims 4 and 5, $P_1(s) = P_1(s')$ and $P_2(s) = P_2(s'')$. By Claims 2 and 3, $s, s', s'' \in L(E)$, and s ends at state q_3 in E, s' ends at state q_1 in E, s'' ends at state q_2 in E and s ends at state q_4 in G. Since σ leads from (q_1, q_2, q_3, q_4) to the dump state we have that $\delta^E(\sigma, q_3)$ is not defined, $\delta^G(\sigma, q_1)$ is defined, and if $\sigma \in \Sigma_{1,c}$ then $\delta^E(\sigma, q_1)$ is defined, and if $\sigma \in \Sigma_{2,c}$ then $\delta^E(\sigma, q_2)$ is defined. So we have $s\sigma \notin L(E)$, $s\sigma \in L(G)$, and if $\sigma \in \Sigma_{1,c}$, then $s'\sigma \in L(E)$ and if $\sigma \in \Sigma_{2,c}$ then $s''\sigma \in L(E)$; therefore, co-observability fails.

$(\Longrightarrow)$ Assume that E *not* co-observable w.r.t. G. First we require a claim that shows that M was indeed constructed so that the 4-tuple states through which paths in M pass keep track of strings s, s', s'' such that $P_1(s) = P_1(s')$ and $P_2(s) = P_2(s'')$.

Claim 6 *Given $s, s', s'' \in L(E)$ such that $P_1(s) = P_1(s')$ and $P_2(s) = P_2(s'')$; then there exists a sequence $s^M \in L(M) \setminus L_m(M)$ such that $F_3(s^M) = s$, $F_1(s^M) = s'$ and $F_2(s^M) = s''$.*

Proof We proceed by induction on length of s.

Basis Suppose that $|s| = 0$, i.e., $s = \varepsilon$. Then $s' \in (\Sigma \setminus \Sigma_{1,o})^*$, since $P_1(s) = P_1(s')$, and $s'' \in (\Sigma \setminus \Sigma_{2,o})^*$, since $P_2(s) = P_2(s'')$. So we can assume that s' and s'' have the following form:

$$\begin{aligned} s' &= s'(0)s'(1)\ldots s'(m) \quad \text{where for } i = 1, \ldots, m,\ s'(i) \in \Sigma \setminus \Sigma_{1,o} \\ s'' &= s''(0)s''(1)\ldots s''(n) \quad \text{where for } i = 1, \ldots, n,\ s''(i) \in \Sigma \setminus \Sigma_{2,o} \end{aligned}$$

for some natural numbers m, n. Then we can define a path s^M in M according to:

$$(q_0^E, q_0^E, q_0^E, q_0^G) \underbrace{\xrightarrow{(s'(0),1)} \ldots \xrightarrow{(s'(m),1)}}_{m+1 \text{ times}} \underbrace{\xrightarrow{(s''(0),2)} \ldots \xrightarrow{(s''(n),2)}}_{n+1 \text{ times}} (q_1, q_2, q_3, q_4)$$

for some $(q_1, q_2, q_3, q_4) \neq d$, i.e., $s^M = s's''$. This sequence is a legitimate path in M since $s' \in L(E)$ means $\delta^E(s'(m), \ldots \delta^E(s'(1), \delta^E(s'(0), q_0^E)) \ldots)$ is defined and so m transitions of type 1 are allowed in M; then, since transitions of type 1 don't change the value of the second argument in the 4-tuple state of M, after a string of transitions of type 1 we can do n transitions of type 2 since $s'' \in L(E)$. By definition, F_1 erases transitions of type 2 and F_2 erases transitions of type 1 so $F_1(s's'') = s'$

and $F_2(s's'') = s''$, as required. The operator F_3 erases both transitions of type 1 and of type 2 so $F_3(s^M) = \varepsilon = s$.

Inductive Hypothesis Suppose that the claim holds for all s such that $|s| \leq n$. Consider $t = s\sigma$ where $|s| = n$, $\sigma \in \Sigma$ such that $t, t', t'' \in L(E)$, $P_1(t) = P_1(t')$ and $P_2(t) = P_2(t'')$.

Case 1: $\boldsymbol{\sigma \in \Sigma_{1,o} \cap \Sigma_{2,o}}$. Then $(P_1(s\sigma) = P_1(t')) \implies (t' = s'\sigma v')$ for some $s' \in \Sigma^*, v' \in (\Sigma \setminus \Sigma_{1,o})^*$, and $P_1(s) = P_1(s')$. Also, $(P_2(s\sigma) = P_2(t'')) \implies (t'' = s''\sigma v'')$ for some $s'' \in \Sigma^*, v'' \in (\Sigma \setminus \Sigma_{2,o})^*$, and $P_2(s) = P_2(s'')$.

Since the inductive hypothesis holds for s, s', s'', there exists an $s^M \in L(M) \setminus L_m(M)$ such that

$$\begin{aligned} F_3(s^M) &= s \\ F_1(s^M) &= s' \\ F_2(s^M) &= s''. \end{aligned}$$

Assume that s^M ends at state (q_1, q_2, q_3, q_4) in M. Since $t, t', t'' \in L(E)$ we have $s\sigma, s'\sigma, s''\sigma \in L(E)$ (since $L(E)$ is prefix-closed). Also, by Claims 1, 2 and 3, s ends at state q_3 in E and at state q_4 in G, s' ends at state q_1 in E, and s'' ends at state q_2 in E. So, $\delta^E(\sigma, q_1)$, $\delta^E(\sigma, q_2)$, $\delta^E(\sigma, q_3)$, and $\delta^G(\sigma, q_4)$ are all defined. Therefore, the following is also a path in M:

$$(q_0^E, q_0^E, q_0^E, q_0^G) \xrightarrow{s^M} (q_1, q_2, q_3, q_4) \xrightarrow{(\sigma,4)} (\delta^E(\sigma, q_1), \delta^E(\sigma, q_2), \delta^E(\sigma, q_3), \delta^G(\sigma, q_4))$$

Now, we can add the following transitions and still stay on a path in M (that does not lead to the dump state):

$$\underbrace{\xrightarrow{(v'(0),1)} \cdots \xrightarrow{(v'(k),1)}}_{k+1 \text{ times}} \underbrace{\xrightarrow{(v''(0),2)} \cdots \xrightarrow{(v''(\ell),2)}}_{\ell+1 \text{ times}}$$

$$\text{where} \quad \begin{aligned} v'(0) \dots v'(k) = v' \quad & v'(i) \in \Sigma \setminus \Sigma_{1,o} \\ v''(0) \dots v''(\ell) = v'' \quad & v''(i) \in \Sigma \setminus \Sigma_{2,o}. \end{aligned}$$

The same reasoning used in the basis case to show that the $s's''$ defined there was a legitimate path in M can be used here to show that $s^M\sigma v'v''$ as defined above is indeed a legitimate path in M.

All that remains is to show that

$$\begin{aligned} F_3(s^M\sigma v'v'') &= t \\ F_1(s^M\sigma v'v'') &= t' \\ F_2(s^M\sigma v'v'') &= t''. \end{aligned}$$

We have

$$F_3(s^M\sigma v'v'') \quad = \quad F_3(s^M)F_3(\sigma)F_3(v'v'')$$

$$
\begin{aligned}
&= sF_3(\sigma)F_3(v'v'') \quad \text{(by the inductive hypothesis)}\\
&= s\sigma F_3(v'v'') \quad \text{(since } \sigma \text{ comes from a transition of type 4)}\\
&= s\sigma\\
&\qquad \text{(since } v'v'' \text{ result from transitions of types 1 and 2}\\
&\qquad \text{and } F_3 \text{ maps to } \varepsilon \text{ for all those)}\\
&= t.
\end{aligned}
$$

Now, $F_1(s^M\sigma) = s'\sigma$ since $F_1(\sigma) = \sigma$ for σ for a transition of type 4. Also,

$$
\begin{aligned}
F_1(v') &= v' \quad \text{(since } v' \text{ composed of transitions of type 1)}\\
F_1(v'') &= \varepsilon \quad \text{(since } v'' \text{ composed of transitions of type 2).}
\end{aligned}
$$

Therefore, $F_1(s^M\sigma v'v'') = s'\sigma v' = t'$. A similar argument can be used to show that $F_2(s^M\sigma v'v'') = s''\sigma v'' = t$.

Case 2: $\boldsymbol{\sigma \in \Sigma_{i,o} \setminus \Sigma_{2,o}}$. Then $P_1(s\sigma) = P_1(t') \Longrightarrow t' = s'\sigma v'$ for some $v' \in (\Sigma \setminus \Sigma_{1,o})^*$ and $P_1(t') = P_1(s')$. Also,

$$
\begin{aligned}
P_2(s) &= P_2(s\sigma) \quad \text{(since } \sigma \notin \Sigma_{2,o})\\
&= P_2(t'').
\end{aligned}
$$

So, we have s, s', t'' that satisfy the inductive hypothesis. Therefore, there exists an $s^M \in L(M) \setminus L_m(M)$ such that

$$
\begin{aligned}
F_3(s^M) &= s\\
F_1(s^M) &= s'\\
F_2(s^M) &= t''.
\end{aligned}
$$

Assume that s^M ends at state (q_1, q_2, q_3, q_4). Then, using reasoning analogous to that for Case 1, we get that $\delta^E(\sigma, q_1), \delta^E(\sigma, q_3)$ and $\delta^G(\sigma, q_4)$ are defined. Therefore, the following is a path in M:

$$
(q_0^E, q_0^E, q_0^E, q_0^G) \xrightarrow{s^M} (q_1, q_2, q_3, q_4) \xrightarrow{(s,6)} (\delta^E(\sigma, q_1), q_2, \delta^E(\sigma, q_3), \delta^G(\sigma, q_4)).
$$

Now, we can add the following transitions and still get a path in M (that does not lead to the dump state):

$$
\underbrace{\xrightarrow{(v'(0),1)} \ldots \xrightarrow{(v'(k),1)}}_{k+1 \text{ times}}
$$

where $v'(0)\ldots v'(k) = v', v'(i) \in \Sigma \setminus \Sigma_{1,o}$. We can do this since, by Claim 2, s' leads to state q_1 in E; so, $t' = s'\sigma v' \in L(E)$ means that $s'\sigma$ leads to state $\delta^E(\sigma, q_1)$ and, therefore, $\delta(v', \delta^E(\sigma, q_1))$ is defined.

Now, we also have that

$$
F_3(s^M\sigma v') = sF_3(\sigma)F_3(v') \quad \text{(by the inductive hypothesis)}
$$

$$\begin{aligned}
&= s\sigma F_3(v') \quad \text{(since } \sigma \text{ comes from a transition of type 6)} \\
&= s\sigma \\
&\qquad \text{(since } v' \text{ results from transitions of type 1} \\
&\qquad \text{and } F_3 \text{ maps to } \varepsilon \text{ for all those)} \\
&= t
\end{aligned}$$

$$\begin{aligned}
F_1(s^M\sigma v') &= s'F_1(\sigma)F_1(v') \quad \text{(by the inductive hypothesis)} \\
&= s'\sigma F_1(v') \quad \text{(since } F_1(\sigma)=\sigma \text{ for } \sigma \text{ a transition of type 6)} \\
&= s'\sigma v' \\
&\qquad \text{(since } v' \text{ composed of transitions of type 1} \\
&\qquad \text{and by definition of } F_1\text{)} \\
&= t' \\
F_2(s^M\sigma v') &= t''F_2(\sigma)F_2(v') \quad \text{(by the inductive hypothesis)} \\
&= t'' \\
&\qquad \text{(since } \sigma \text{ \& } v' \text{ come from transitions of types 6 and 1} \\
&\qquad \text{and } F_2 \text{ projects those onto } \varepsilon\text{).}
\end{aligned}$$

Case 3: $\boldsymbol{\sigma \in \Sigma_{2,o} \setminus \Sigma_{1,o}}$. This case is analogous to Case 2; we just use $(\sigma, 5)$ first plus a string of transitions of type 2.

Case 4: $\boldsymbol{\sigma \notin \Sigma_{1,o}, \sigma \notin \Sigma_{2,o}}$. Then we get

$$P_1(s\sigma) = P_1(t') \Longrightarrow P_1(s) = P_1(t')$$
$$P_2(s\sigma) = P_2(t'') \Longrightarrow P_2(s) = P_2(t'').$$

Therefore, s, t', t'' satisfy the inductive hypothesis. So, there exists an $s^M \in L(M) \setminus L_m(M)$ such that

$$F_3(s^M) = s$$
$$F_1(s^M) = t'$$
$$F_2(s^M) = t''.$$

Assume that s^M ends at state (q_1, q_2, q_3, q_4). Since $s\sigma \in L(E)$ and s ends at state q_3 in E and q_4 in G (by Claim 1), then $\delta^E(\sigma, q_3)$ and $\delta^G(\sigma, q_4)$ are defined. Therefore, we add the following transition to s^M to get a new path in M (that does not end at the dump state):

$$(q_1, q_2, q_3, q_4) \xrightarrow{(\sigma,3)} (q_1, q_2, \delta^E(\sigma, q_3), \delta^G(\sigma, q_1)).$$

We also have that

$$\begin{aligned}
F_3(s^M\sigma) &= F_3(s^M)F_3(\sigma) \\
&= sF_3(\sigma) \quad \text{(by the inductive hypothesis)}
\end{aligned}$$

$$\begin{aligned} &= s\sigma \quad \text{(since } \sigma \text{ results from a transition of type 3 and } F_3(\sigma) = \sigma) \\ &= t \end{aligned}$$

$$\begin{aligned} F_1(s^M\sigma) &= F_1(s^M)F_1(\sigma) \\ &= t'F_1(\sigma) \quad \text{(by the inductive hypothesis)} \\ &= t' \quad \text{(since } F_1(\sigma) = \varepsilon \text{ for } \sigma \text{ a transition of type 3).} \end{aligned}$$

Similarly, $F_2(s^M\sigma) = t''$.

$\blacksquare_{\text{claim 6}}$

We return to the proof of Proposition 3.1.

Since $L(E)$ is *not* co-observable w.r.t. G, one of the first three conjuncts of co-observability fails. (The fourth conjunct holds since $L(E)$ is prefix-closed.) Therefore there exist $s, s', s'' \in L(E)$ such that $P_1(s) = P_1(s')$, $P_2(s) = P_2(s'')$ and either (i) $\exists\sigma \in \Sigma_{1,c} \cap \Sigma_{2,c})$ $s'\sigma \in L(E), s''\sigma \in L(E), s\sigma \in L(G)$, but $s\sigma \notin L(E)$ or (ii) $(\exists\sigma \in \Sigma_{1,c} \setminus \Sigma_{2,c})$ $s'\sigma \in L(E), s\sigma \in L(G)$, but $s\sigma \notin L(E)$, or (iii) $(\exists\sigma \in \Sigma_{2,c} \setminus \Sigma_{1,c})$ $s''\sigma \in L(E), s\sigma \in L(G)$, but $s\sigma \notin L(E)$.

By Claim 6, there exists $s^M \in L(M) \setminus L_m(M)$ such that

$$\begin{aligned} F_3(s^M) &= s \\ F_1(s^M) &= s' \\ F_2(s^M) &= s''. \end{aligned}$$

Suppose that s^M leads to state (q_1, q_2, q_3, q_4) in M. By Claims 1, 2 and 3, s leads to state q_3 in E and q_4 in G, s' leads to state q_1 in E, and s'' leads to state q_2 in E.

Suppose that (i) is the case. Then $\delta^E(\sigma, q_1)$ is defined, $\delta^E(\sigma, q_2)$ is defined, $\delta^G(\sigma, q_4)$ is defined, but $\delta^E(\sigma, q_3)$ is not defined. Since $\sigma \in \Sigma_{1,c} \cap \Sigma_{2,c}$ we have that σ leads from (q_1, q_2, q_3, q_4) to the dump state. Therefore $s^M\sigma$ is a path in M from the initial state to the dump state. Analogous reasoning works for (ii) and (iii). $\blacksquare$

Having shown that the above construction is a test for co-observability, we now turn to our main concern: the computational complexity of our algorithms.

Proposition 3.2 *Given finite-state automata E and G, the construction of $M(G, E)$ is polynomial (time) in $max(|Q^G|, |Q^E|)$.*

Proof We assume that $|\Sigma|$ is small relative to the state sets or that it stays relatively constant even with increases in the number of states and, therefore, we disregard the size of Σ in our computations.

Let $n = max(|Q^G|, |Q^E|)$. Computing the state set of M takes asymptotic computing time of at most $O(n^4)$ (since the state set is the Cartesian product of four state sets). At each state of M we must compute the transition function δ^M.

We can check if an element is in a set of size m in time $O(m \log m)$ (if necessary, by first sorting the set, which can be done in time $O(m \log m)$ and then searching through the ordered set, which can be done in time $O(\log m)$ [HS82]). Therefore, we can check if a given event σ is in $\Sigma_{1,o}$ (resp. $\Sigma_{2,o}$, $\Sigma_{1,c}$, $\Sigma_{2,c}$) in time $O(|\Sigma_{1,o}| \log |\Sigma_{1,o}|)$ (resp. $O(|\Sigma_{2,o}| \log |\Sigma_{1,o}|)$, $O(|\Sigma_{1,c}| \log |\Sigma_{1,c}|)$, $O(|\Sigma_{2,c}| \log |\Sigma_{2,c}|)$).

We assume that E and G are stored so that given any $\sigma \in \Sigma$, $q^E \in Q^E$ (resp. $q^G \in Q^G$), $\delta^E(\sigma, q^E)$ (resp. $\delta^G(\sigma, q^G)$) can be computed in polynomial time w.r.t. the number of states in E (resp. G). This is the case, for example, when E and G are each stored as adjacency matrices (whose space requirements are $O(m^2)$, where m is the number of states in the automaton).

To summarize, at each state of M, for each σ in Σ, we check if σ is in $\Sigma_{1,o}$ and if σ is in $\Sigma_{2,o}$. Depending on the outcome, we compute the set of next states in M. To compute next states all we need to do is compute $\delta^E(\cdot, \cdot)$ and $\delta^G(\cdot, \cdot)$ and, in some cases, check if $\sigma \in \Sigma_{1,c}$ or if $\sigma \in \Sigma_{2,c}$. As explained above, all these computations can be done in polynomial time. ∎

Our main result follows almost immediately. That is, we can show that our construction in no way leads to an exponential explosion in computing time and, therefore, co-observability can be decided in polynomial time.

Theorem 3.1 *Given finite-state automata E and G, it can be decided in polynomial time whether or not $L(E)$ is co-observable w.r.t. G.*

Proof Given G and E, we construct M as above. By Proposition 3.1, deciding co-observability is equivalent to checking if the language recognized by M is nonempty. This can be done in time polynomial w.r.t. the state space of M, which is $|Q^E| \cdot |Q^E| \cdot |Q^E| \cdot |Q^G|$. Since M has only one terminal state, M recognizes a nonempty language if and only if there is a path from the initial state in M to the dump state. Such a search on M can be done in $O(n^2)$ time where $n = |Q^E| \cdot |Q^E| \cdot |Q^E| \cdot |Q^G|$ [PS82]. By Proposition 3.2, the construction itself can be done in polynomial time w.r.t. the number of states. ∎

Unfortunately, just as in the centralized control problems considered by Tsitsiklis [Tsi89], the above result is the only positive complexity result associated with the decentralized supervisory control problems under consideration. So, while we can determine in polynomial time whether or not the special case (i.e., when the endpoints of the desired range of behaviour are equal) of the Decentralized Control Problem given in Section 2 is solvable, even if the answer is "yes", it can be shown that there is no polynomial-time algorithm to construct a supervisor solution. This follows *a fortiori* from the centralized control examples given in [Tsi89].

3.2 Solving the More General Problem

When the range of desirable behaviour specified in the Decentralized Control Problem is not narrowed to a single language, i.e., when A is not necessarily equal to E, checking for solution existence becomes qualitatively harder. This is formalized in the next theorem, which follows almost immediately from the centralized analog given in [Tsi89].

Tsitsiklis shows in [Tsi89] that, given finite-state automata G, A and E, there is no polynomial-time algorithm for deciding whether there exists a single supervisor $\mathcal{S}$ such that $L(A) \subseteq L(\mathcal{S}/G) \subseteq$

$L(E)$.[4] His problem formulation does not address the issue of blocking, and it is not immediately apparent that just because it takes a long time to decide if there is a supervisor that generates closed-loop behaviour within a given range that the decision couldn't be made quicker by requiring that the supervisor also be nonblocking. However, it's almost trivial to show that checking for blocking does not speed things up. This result is due to Professor Feng Lin and arose in discussions.

Consider the following two problems:

Problem 1 *Given finite-state machines, G, A, and E, does there exist a supervisor $\mathcal{S}$ with partial observation such that $L(A) \subseteq L(\mathcal{S}/G) \subseteq L(E)$?*

Problem 2 *Given finite-state machines, G, A, and E, does there exist a supervisor $\mathcal{S}$ with partial observation such that $L(A) \subseteq L(\mathcal{S}/G) \subseteq L(E)$ and $\mathcal{S}/G$ is nonblocking?*

Now we show that given an algorithm to Problem 2 we could find an algorithm for Problem 1, which would mean that if Problem 2 were solvable in polynomial time, so would be Problem 1 (provided our reduction is also constructed in polynomial time).

Proposition 3.3 *Given an algorithm for solving Problem 2, there exists an algorithm for solving Problem 1.*

Proof Given a finite-state automaton G, let G' have the same transition structure as G but with all its states marked. We claim that G, A, E is a positive instance for Problem 1 if and only if G', A, E is a positive instance for Problem 2.

($\Longleftarrow$) Suppose that G', A, E is a positive instance of Problem 2. Then there exists a supervisor $\mathcal{S}$ such that $L(A) \subseteq L(\mathcal{S}/G') \subseteq L(E)$ and $\mathcal{S}/G'$ is nonblocking. Dropping the nonblocking property, we have a supervisor $\mathcal{S}$ such that $L(A) \subseteq L(\mathcal{S}/G') \subseteq L(E)$. This implies that $L(A) \subseteq L(\mathcal{S}/G) \subseteq L(E)$ since closed behaviour is not affected by which states are marker states.

($\Longrightarrow$) Suppose that G, A, E is a positive instance of Problem 1. Then there exists a supervisor $\mathcal{S}$ such that $L(A) \subseteq L(\mathcal{S}/G) \subseteq L(E)$. Again, since marker states do not affect closed behaviour, we have that $L(A) \subseteq L(\mathcal{S}/G') \subseteq L(E)$. Let $\mathcal{S}'$ be the supervisor $\mathcal{S}$ but with all states of its transition structure marked. Then $L(A) \subseteq L(\mathcal{S}'/G') \subseteq L(E)$, since the marking of states in the supervisor does not affect closed behaviour. Moreover, $\mathcal{S}'/G'$ is trivially nonblocking since all its states are marked. ∎

Theorem 3.2 *Given finite-state automata G, A and E, (unless $P = NP$) there is no polynomial-time algorithm for deciding whether there exist $\mathcal{S}_1$ and $\mathcal{S}_2$ such that $\mathcal{S}_1 \wedge \mathcal{S}_2$ is a proper supervisor for G and $L(A) \subseteq L(\tilde{\mathcal{S}}_1 \wedge \tilde{\mathcal{S}}_2/G) \subseteq L(E)$.*

[4] The theorem statement contains the proviso that P, the class of decision problems solvable by polynomial-time algorithms, is not equal to NP, the class of decision problems solvable by nondeterministic polynomial-time algorithms—a widely accepted conjecture.

Proof The result follows immediately from the fact that centralized control is just a special case of decentralized control and from Proposition 3.3, together with the observation that the only computations in the reduction require marking some states of certain automata, which can be done in polynomial time (by searching through the list of states). ∎

4 Concluding Remarks

We have generalized the results on computational complexity of supervisory control problems given in the seminal paper by Tsitsiklis [Tsi89]. We have shown that the question of whether there exist decentralized controllers that ensure that closed-loop behaviour precisely equals some prescribed desired behaviour is decidable in polynomial time (w.r.t. the size of the state spaces of the processes involved). However, there is no polynomial-time algorithm that, for any given plant, can *produce* supervisors (when they exist) that ensure that the closed-loop system behaves as desired. Moreover, once the class of problems is broadened to include those where controllers are sought to guarantee that behaviour lie within a prescribed *range*, then solvability is no longer decidable in polynomial time.

In previous work, [RW92a], it was shown that protocol verification problems can be viewed as decentralized discrete-event problems. In particular, the data transmission problem associated with the well-known Alternating Bit Protocol was analyzed using our methodology, i.e., using the property of "co-observability". It was demonstrated that co-observability could be used to detect protocol failures. At the time when [RW92a] was written, the existing algorithm for deciding co-observability was exponential time. Now, with the results presented here, we know that checking protocol correctness for that type of communication problem can be done in polynomial time.

Insofar as computational complexity formalizes what intuition, experiments or simulation have suggested is "hard" to solve, there may be some connection between the computational infeasibility of *synthesizing* supervisors that guarantee desired behaviour in decentralized discrete-event systems and the relative lack of success in communication protocol synthesis as compared with protocol verification.

References

[Bal91] S. Balemi. Discrete-event systems control of a rapid thermal multiprocessor. Report No. 91-12, Automatic Control Laboratory, Swiss Federal Institute of Technology, ETH-Zentrum, Zürich, Switzerland, 1991.

[CDFV88] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Transactions on Automatic Control*, 33(3):249–260, March 1988.

[CM89] H. Cho and S. I. Marcus. Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations. *Mathematical Systems Theory*, 22:177–211, 1989.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[HS82] E. Horowitz and S. Sahni. *Fundamentals of Data Structures.* Computer Science Press, Rockville, MD, 1982.

[HSF91] G. Hoffmann, C. Schaper, and G. Franklin. Discrete event controller for a rapid thermal multiprocessor. In *Proceedings of the American Control Conference*, volume 3, pages 2936–2938, Boston, MA, June 1991.

[Kro87] B. H. Krogh. Controlled Petri nets and maximally permissive feedback logic. In *Proceedings of the 25th Annual Allerton Conference on Communication, Control and Computing*, pages 317–326, University of Illinois, Urbana, 1987.

[Laf88] S. Lafortune. Modeling and analysis of transaction execution in database systems. *IEEE Transactions on Automatic Control*, 33(5):439–447, May 1988.

[LW88] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44:173–198, 1988. Also appears as Systems Control Group Report #8701, Department of Electrical Engineering, University of Toronto, 1987.

[LW90] F. Lin and W. M. Wonham. Decentralized control and coordination of discrete-event systems with partial observation. *IEEE Transactions on Automatic Control*, 35(12):1330–1337, December 1990. Also appears as Systems Control Group Report #8909, Department of Electrical Engineering, University of Toronto, 1989, and in shorter form as "Decentralized control and coordination of discrete-event systems", in *Proceedings of the 27th IEEE Conference on Decision and Control*, December 1988, pages 1125-1130.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[Ram83] P. J. Ramadge. *Control and Supervision of Discrete Event Processes.* PhD thesis, Department of Electrical Engineering, University of Toronto, 1983.

[RW82] P. J. Ramadge and W. M. Wonham. Supervision of discrete event processes. In *Proceedings of the 21st IEEE Conference on Decision and Control*, volume 3, pages 1228–1229, December 1982.

[RW87] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987. Also appears as Systems Control Group Report #8311, Department of Electrical Engineering, University of Toronto, 1983.

[RW90] K. Rudie and W. M. Wonham. Supervisory control of communicating processes. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Protocol Specification, Testing and Verification X*, pages 243–257. Elsevier Science (North-Holland), 1990. Expanded version appears as Systems Control Group Report #8907, Department of Electrical Engineering, University of Toronto, 1989.

[RW92a] K. Rudie and W. M. Wonham. Protocol verification using discrete-event systems. In *Proceedings of the 31st IEEE Conference on Decision and Control*, pages 3770–3777, Tucson, Arizona, December 1992.

[RW92b] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.

[Tsi89] J. N. Tsitsiklis. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals, and Systems*, 2:95–107, 1989.

[Won88] W. M. Wonham. A control theory for discrete-event systems. In M. J. Denham and A. J. Laub, editors, *Advanced Computing Concepts and Techniques in Control Engineering*, volume F47 of *NATO ASI Series*, pages 129–169. Springer-Verlag, Berlin, 1988. Also appears as Systems Control Group Report #8714, Department of Electrical Engineering, University of Toronto, 1988.

[WR87] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987. Also appears as Systems Control Group Report #8312, Department of Electrical Engineering, University of Toronto, 1983.

[WR88] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems*, 1:13–30, 1988.