Protocol Verification Using Discrete-Event Systems

Karen Rudie*
Institute for Mathematics
and its Applications
University of Minnesota
Minneapolis, MN 55455
USA

W. Murray Wonham
Systems Control Group
Department of Electrical Engineering
University of Toronto
Toronto, Ontario M5S 1A4
Canada

Abstract

It can be shown that the problem of reliable transmission of data over an unreliable communication channel can be restated as a decentralized control problem of discrete-event systems. Necessary and sufficient conditions for the existence of solutions to such decentralized supervisory control problems have been found. These conditions are used to verify the correctness of a protocol for the data transmission problem. In particular, it is demonstrated that our method provides a systematic check on whether the protocol satisfies the required safety property, as opposed to relying on finding, *ad hoc*, circumstances under which the protocol fails.

1 Introduction

Supervisory control theory is a formalism for modeling discrete-event processes and applying control theoretic techniques to solve problems involving such processes. As in classical control theory, the power of the approach lies in first identifying the fixed part of the system, i.e., the process to be controlled, which is called a *plant*, then applying control through the use of controllers. A control problem is one where at the outset the plant is not behaving as we wish it would and therefore we must find controllers that inhibit the plant's behaviour such that the plant exhibits some given desired behaviour. Typically, control theoretic work provides automatic procedures for deriving suitable controllers that solve a class of problems.

In [13] we introduced a problem formulation that captures those discrete-event systems whose problem description is given as global specifications but whose solution requires decentralized control. We also demonstrated that our formulation can be used to model the data transmission problem associated with the Alternating Bit Protocol. In [14], we found necessary and sufficient conditions for solving problems that lie in the above class. Here we demonstrate that using these conditions, we can test whether a candidate protocol for the above data transmission problem is correct.

Much work has already been done on protocol verification but the bulk of it focuses on what West calls "validation" [16]. Validation is distinguished from verification as follows. Validation is used to determine if a protocol satisfies certain correctness properties (sometimes referred to as *syntactic* properties) such as freedom from deadlock, unspecified receptions, and nonexecutable interactions[1] (sometimes referred to as *logical* errors). Verification is used to see whether a protocol achieves a desired functionality (sometimes referred to as *semantic* properties, as in [4]). In other words, syntactic properties are those common to all protocols, whereas semantic properties are specific to some given protocol.

For protocols that can be represented by communicating finite-state machines (such as the example in Section 3), the most widely used technique for protocol validation (of syntactic properties) is *reachability analysis*, an exhaustive search through the state-space of the overall system [1], [18]. For protocols that can be represented using a programming language model, a modal logic or some other formalism that admits predicates to describe system properties can be used for validation.

In this paper, we are interested in verification of the semantic properties of a protocol. In [2], Bochmann and Gecsei propose a method for such verification. A drawback to their technique is that it uses predicates over Petri nets and the verification procedure relies on producing a desired assertion but the paper does not provide a method for finding the appropriate assertion. On the other hand, their method handles both safety and liveness properties while ours can verify only safety properties. Hailpern and Owicki [6] use an approach similar to that in [2], except that their processes are given as modules (sections of program

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

[1]See [18] for definitions of these properties.

code) and their assertions are temporal logic predicates. The work described in [7], in particular that of Milner [9], also addresses verification of semantic properties. Informally, they model protocols as processes in some process algebra and then show that the protocol is "equivalent" to some behavioural description of the problem specification. However, showing such an equivalence requires finding a relation over the process space and showing that this relation is what is called a *bisimulation*.

We show that describing communication problems as instances of control problems leads to an alternative, simple method for the verification of semantic properties of a protocol.

2 Background

The material in this paper is set in the supervisory control framework for discrete-event systems initiated by Ramadge and Wonham [11], [10], [12], [17]. Supervisory control theory draws from automata theory and the theory of formal languages; except where otherwise stated, the concepts in this section are taken either from the fundamentals of formal language and automata theory or from the above references.

We assume that the process to be controlled can be modeled by a five-tuple deterministic automaton $G = (\Sigma, Q, \delta, q_0, Q_m)$ where Σ is an alphabet of event labels, Q is a set of states, $q_0 \in Q$ is the initial state, $Q_m \in Q$ is the set of terminal or *marked* states and $\delta : \Sigma \times Q \rightarrow Q$, the transition function, is a partial function defined at each state in Q for a subset of Σ. For the case where Q is finite, G can be represented by a finite state machine whose nodes are states and whose edges are transitions defined by δ. On such a transition diagram, the initial state is identified by an arrow entering it and marked states by circles around the nodes. The set Σ is the set of all edge labels on the diagram.

A *string* over a given alphabet is a sequence of labels from that alphabet. In our case, strings represent sequences of events. As is usual in formal language theory, Σ^* denotes the set of all finite strings over Σ including the null string ε. Then δ can be extended to Σ^* by defining $\delta(\varepsilon, q) := q$ and $(\forall \sigma \in \Sigma,\ s \in \Sigma^*)\ \delta(s\sigma, q) := \delta(\sigma, \delta(s, q))$. Then G is characterized by two subsets of Σ^* called the *closed behaviour of* G, written $L(G)$, and the *marked behaviour of* G, written $L_m(G)$. The language $L(G)$ is defined as

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

and is interpreted to mean the set of all possible event sequences which the plant may generate. The language $L_m(G)$ is defined as

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

and is intended to distinguish some subset of possible plant behaviour as representing completed tasks. By definition, $L_m(G) \subseteq L(G)$.

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 or disable 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 is then an agent which observes subsequences of the sequences of events generated by G and enables or disables 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 (S, ψ) where S is an automaton which recognizes a language over the observable event set of the plant G and ψ, called a *feedback map*, is a map from the observable event set and states of S to the set $\{enable, disable\}$ with the requirement that ψ enable all uncontrollable events at every state of S. The automaton S tracks and controls the behaviour of G. It changes state according to its view of the events generated by G and, in turn, at each state x of S the control rule $\psi(\sigma, x)$ dictates whether σ is to be enabled or disabled at the corresponding state of G.

The behaviour of the closed-loop system, i.e., the sequences of events generated while the plant is under the control of $\mathcal{S} = (S, \psi)$, is captured by an automaton $\mathcal{S}/G$ whose closed behaviour, denoted by $L(\mathcal{S}/G)$, permits a string to occur if the string is in both G and S and each event in the string is enabled by ψ. The closed-loop system's marked behaviour 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.

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 according to: $P(\varepsilon) = \varepsilon$ and $(\forall \sigma \in \Sigma,\ s \in \Sigma^*)\ P(s\sigma) = P(s)P(\sigma)$, i.e., P erases all unobservable events.

For supervisors $\mathcal{S} = (S, \phi)$ and $\mathcal{T} = (T, \psi)$ acting on G, the *conjunction* of $\mathcal{S}$ and $\mathcal{T}$ is the supervisor denoted by $\mathcal{S} \wedge \mathcal{T} = ((S \times T), \phi * \psi)$ where $\mathcal{S} \times \mathcal{T}$ recognizes the intersection of the languages recognized by S and T and $\phi * \psi$ disables an event if and only if either ϕ or ψ disables it. Thus, $\mathcal{S} \wedge \mathcal{T}$ models the actions of $\mathcal{S}$ and $\mathcal{T}$ operating in parallel.

Given a supervisor $\mathcal{S}$ acting on 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 $\mathcal{S}$ is referred to as a *local* supervisor and $\tilde{\mathcal{S}}$ as its *global extension* (since $\tilde{\mathcal{S}}$ acts on all of Σ while $\mathcal{S}$ acts only on a subset of Σ).

We consider the problem from [13] and [14], formulated as follows:

Decentralized Control Problem *Given a plant G over an alphabet Σ, a legal language $E \subseteq L_m(G)$, a minimally adequate language $A \subseteq E$, and sets $\Sigma_{1,c}, \Sigma_{2,c}, \Sigma_{1,o}, \Sigma_{2,o} \subseteq \Sigma$, construct local (complete) supervisors $\mathcal{S}_1$ and $\mathcal{S}_2$ 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 [5], provided A and E are prefix-closed. The solution is conveniently described using the notions of controllability and co-observability defined in [11] and [14], 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 $\{st \mid s \in L \wedge t \in M\}$ and the overbar notation denotes prefix-closure (of strings). 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 ($s\sigma \in \overline{K}$).

Now we define 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

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

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

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)). The reader is referred to [14] 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 $L_m(\tilde{\mathcal{S}}_1 \wedge \tilde{\mathcal{S}}_2/G) = E$ if and only if E is controllable and co-observable w.r.t. the plant G. While these necessary and sufficient conditions are given in [5], the paper does not contain results on how to check if the conditions hold. However, in [14] (Corollary 4.1) we show that if G is finite-state, E is a prefix-closed, regular language and $E \neq \emptyset$, then there is an effective procedure for determining if E is controllable and co-observable w.r.t. G.

The procedure we use relies on the following results, taken from [14]. Consider the class of languages

$$\underline{CCCo}(M) \ := \ \{K \mid K \supseteq M,\ K \textit{ is prefix-closed, controllable, and co-observable}\}.$$

It can be shown that this class is nonempty and closed under intersection. Consequently, it contains an infimal element, which we denote by $\inf \underline{CCCo}(M)$. When the plant G is finite-state and the legal language E is regular, we have a formula for computing $\inf \underline{CCCo}(E)$. Moreover, by definition of $\inf \underline{CCCo}(E)$, if E is prefix-closed, then E is controllable and co-observable if and only if $\inf \underline{CCCo}(E) = E$. So, to test if the decentralized control problem is solvable it suffices to compute $\inf \underline{CCCo}(E)$ and see if it equals E itself.

3 An Example of Protocol Verification

We consider the problem of reliably transmitting frames of data over an unreliable channel in the data link layer of the International Standards Organization OSI 7-layer reference model[2]. The goal of a protocol for the problem is to ensure that a sender gets data from a host[3] computer, passes that data over a communication channel to a receiver, which must then pass the data to another host in the same order and without duplicates as the data sequence received by the sender from the originating host. It is assumed that frames may be either lost or damaged by the communication channel. It is further assumed that the receiver has routines associated with it that enable it to detect damaged frames.

We informally describe some of the requirements of a protocol solution. Since the channel can lose frames, there must be some mechanism for retransmission. So that retransmissions will be prompted by channel losses, the sender has a timer that it sets after it has transmitted a frame. If the timer times out, the sender sends the frame again. Moreover, to avoid duplicates and to maintain the correct order, the receiver must be able to distinguish a frame it is seeing for the first time from a retransmission. Consequently, the sender is permitted to attach a sequence number to each frame that it sends. If we limit ourselves to stop-and-wait protocols[4], the sender must receive some acknowledgement from the receiver before it sends the next message. Consequently, the only ambiguity for a receiver is between a frame and its immediate predecessor or successor. It can, therefore, be shown that a 1-bit sequence number (0 or 1) is sufficient.

A protocol that appears to solve the problem is given in [15], p. 147. The protocol works as follows. The sender starts by sending the first frame with sequence number 0; upon receiving an acknowledgement from the receiver, the sender sends the second frame with sequence number 1. The sender continues to alternate sequence numbers between 0 and 1 in this way. The receiver starts by expecting a frame with sequence number 0; when it receives such a frame it knows that it is the first frame and passes the frame to the target host. The receiver then expects a frame with sequence number 1. If it receives any frames with sequence number 0, it knows they are retransmissions of the first frame and discards them. As with the sender, the receiver alternates between 0 and 1 in the sequence number it expects from a new frame.

In Figure 1 we present finite-state machine translations, labeled $SNDR$ and $RCVR$, of the protocol for the sender and receiver. Assuming the channel has finite capacity b, it too can be represented by a finite-state machine; in Figure 2 we provide the state transition diagram, labeled $CHNL$, for the case where $b = 1$. For all figures in this section, we consider all states to be marked and therefore omit identifying marked states on the state transition diagrams. The set of possible events is given by

$$\begin{aligned}\Sigma \;=\; & \{getframe,\ send_0,\ send_1,\ rcv_0,\ rcv_1, \\ & sendack,\ rcvack,\ passtohost,\ timer, \\ & timeout,\ cksumerr,\ cksumerrack,\ lose\},\end{aligned}$$

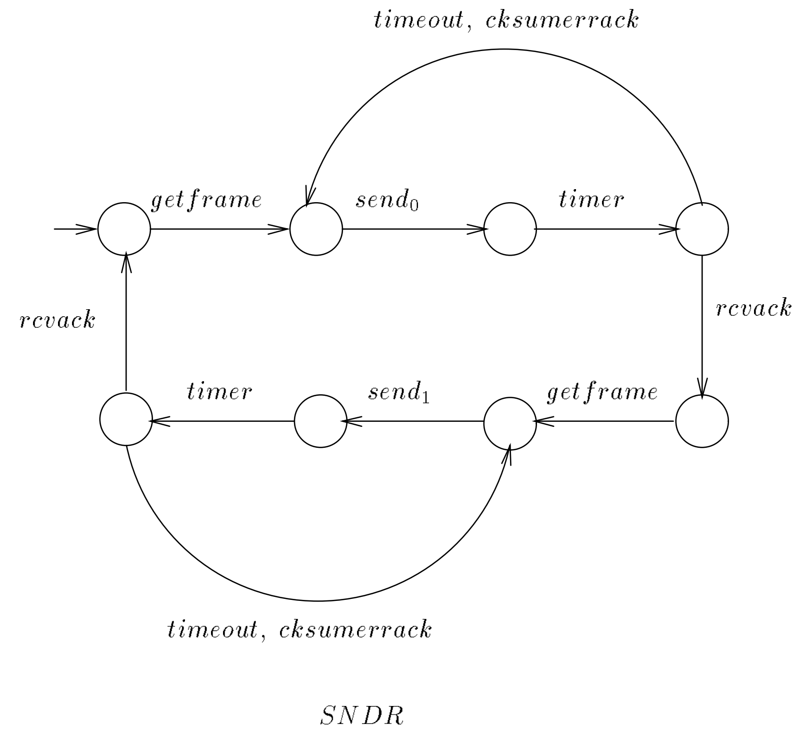
where $getframe$ means the sender gets a new frame from the source host; $send_0$ (resp. $send_1$) means the sender sends the most recent frame it got to the receiver with a sequence number of 0 (resp. 1); rcv_0 (resp. rcv_1) means the receiver receives a frame whose sequence number is 0 (resp. 1); $sendack$ means the receiver sends an acknowledgement to the sender; $rcvack$ means the sender receives an acknowledgement from the receiver; $passtohost$ means the receiver passes the frame it just received to the target host; $timer$ means the sender starts the timer; $timeout$ means the timer times out; $cksumerr$ means that a frame has arrived damaged; $cksumerrack$ means that an acknowledgement has arrived damaged; and $lose$ means the channel loses the message in it. When the channel capacity, b, is greater than 1 the channel losses must be indexed to indicate which message in the channel has just been lost.

The plant under investigation is given by $G = SNDR /\!/ CHNL /\!/ RCVR$ where $/\!/$ denotes the syn-

[2]The Open Systems Interconnection (OSI) reference model is a proposed standard for network architecture which is organized as a series of layers. Each layer offers services to higher layers and machines in the same layer communicate via protocols for that layer. The data link layer is the layer above the physical medium; it breaks data into groups called "frames" and is responsible for ensuring that frames are transmitted across the network without errors—even though errors may occur in the physical medium. See [15] for more details.

[3]A *host* is a machine that runs user programs.

[4]Stop-and-Wait protocols, as defined in [15], are those protocols in which, after having sent a frame, a sender waits for an acknowledgement before proceeding to its next step.



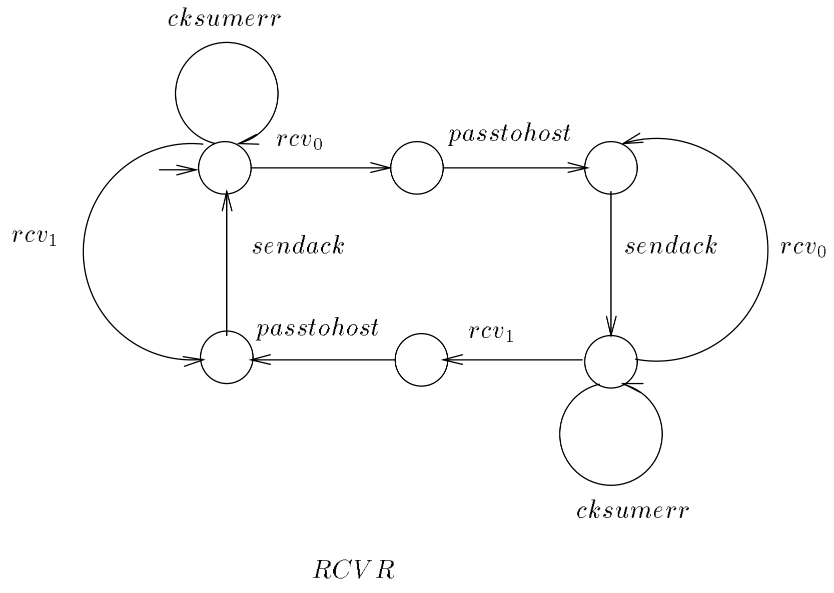


Figure 1: Protocol for Sender and Receiver

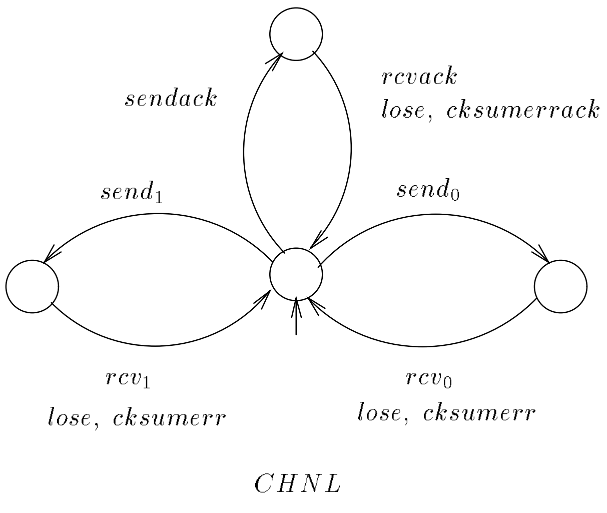


Figure 2: Channel with Capacity 1

chronous product as defined by the concurrent operation in [8] and as used in [5].

If we assume that each agent observes those events that occur at its physical location and that message receptions, channel losses, and timeouts are uncontrollable, then the partitions of event sets are as follows. The observable and controllable sets of events of the sender are

$$\Sigma_{1,o} = \{getframe,\ send_0,\ send_1,\ timer,\ timeout,\ rcvack,\ cksumerrack\}$$

and

$$\Sigma_{1,c} = \{getframe,\ send_0,\ send_1,\ timer\},$$

respectively. The observable and controllable sets of events of the receiver are

$$\Sigma_{2,o} = \{rcv_0,\ rcv_1,\ passtohost,\ sendack,\ cksumerr\}$$

and

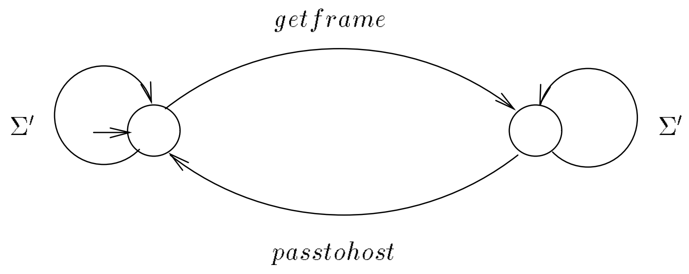
$$\Sigma_{2,c} = \{passtohost,\ sendack\},$$

respectively.

As pointed out by Tanenbaum [15], if the sender times out even though the channel has neither lost nor damaged a frame (rather, the acknowledgement—unbeknownst to the sender—is on its way but is going to take longer than expected to arrive), then the protocol given above does not solve the original problem. The offending scenario is as follows. The sender sends a frame but times out while the acknowledgement is still in the communication channel; therefore, the sender sends a duplicate. When the previous acknowledgement finally arrives, the sender may think that the duplicate frame is the one being acknowledged. If the next new frame sent is lost but the acknowledgement of the duplicate reaches the sender, the sender will not retransmit the lost frame as it will mistakenly assume that the acknowledgement just received is for the new frame.

A successful protocol for the transmission problem must possess two properties: (1) at any point in time, the sequence of data passed by the receiver to the target host must be a prefix of the sequence gotten by the sender from the source host and (2) every element in the sequence of data gotten by the sender from the source host must eventually be passed on by the receiver to the target host. The first property is a *safety* property since it does not require that any action take place but merely that *if* an action does occur, then it is not "bad". When this requirement is met the protocol is said to satisfy *partial correctness* [3]. The second property is a *liveness* property since it does require that some action take place. Since our model does not use infinite-string semantics, we cannot verify the liveness property. However, we will show that using our decentralized control formulation, the safety property can be checked.

To describe the safety property as a legality specification in supervisory control theory, we define the language $E = L(G) \cap LEGAL$ where $LEGAL$ is generated by the finite-state machine in Figure 3. Since all states in $LEGAL$ are marked, E is prefix-closed and hence $\overline{E} = E$. According to the plant description, any new frame gotten by the sender from the source host is transmitted to the receiver at least once and the receiver only passes on to the target host frames it receives. Thus, the requirement in $LEGAL$ that $getframe$ and $passtohost$ strictly alternate ensures that if new data is passed to the target host, it is passed in the correct order and without duplicates.



where $\Sigma' := \Sigma \setminus \{getframe,\ passtohost\}$

LEGAL

Figure 3: Legal Language

That is, sequences of events that are contained in the language *LEGAL* satisfy the safety property.

In terms of event occurrences, the problematic sequence of events is given by the string t below:

$$\begin{aligned} t &= getframe\ send_0\ timer\ rcv_0\ passtohost \\ &\quad sendack\ timeout\ send_0\ timer\ rcvack\ rcv_0 \\ &\quad sendack\ getframe\ send_1\ timer\ lose\ rcvack, \end{aligned}$$

where here the event *lose* refers to the loss of the frame with sequence number 1. This string t leads to illegality because upon assuming that the last frame it sent has been correctly received, the sender will attempt to get a new frame to pass to the receiver. This sequence of events is given by

$$t\sigma,$$

where $\sigma = getframe$. Observe that $t\sigma \notin E$, $t\sigma \in L(G)$, $t \in E$ and $\sigma \in \Sigma_{1,c} \setminus \Sigma_{2,c}$.

Recall that the premature timeout leads to a protocol failure because the lost frame is assumed to be received due to an earlier acknowledgement delayed in the communication pipe. If that frame were not, in fact, lost the protocol would not fail at that point. In other words, consider the sequence t' given as follows:

$$\begin{aligned} t' &= getframe\ send_0\ timer\ rcv_0\ passtohost \\ &\quad sendack\ timeout\ send_0\ timer\ rcvack\ rcv_0 \\ &\quad sendack\ getframe\ send_1\ timer\ rcv_1 \\ &\quad passtohost\ rcvack. \end{aligned}$$

Then $t'\sigma \in E$. However, the sender cannot distinguish between the sequences t and t', i.e.,

$$P_1(t) = P_1(t').$$

Therefore, E is *not* co-observable w.r.t. the given plant G (by conjunct (2) of co-observability).

Let us interpret the above plant as defining all possible executions of a protocol and the legal language as only those executions where data is passed to the target host in the correct order and without duplicates. Then, the question "Is this protocol correct?" can be translated into "Can we ensure (via control) that only correct executions of the protocol are allowed to occur?". We know already that the protocol as given does not contain only correct executions (since $t\sigma \notin E$) but the more interesting question is "Why not?" or "What can be done about it?".

Instead of searching *ad hoc* for a scenario where a given protocol fails, our results permit automatic *a priori* checking. That is, an application of Corollary 4.1 in [14] would indicate immediately that the protocol given above is not suitable since E is not co-observable w.r.t. G. In particular, as stated in Section 2 we could compute $\inf \underline{CCCo}(E)$ and would see that $\inf \underline{CCCo}(E) \neq E$. Since $\inf \underline{CCCo}(E) \neq E$ but $\inf \underline{CCCo}(E) \supseteq E$ (by definition), it must be the case that $\inf \underline{CCCo}(E) \supset E$ which, in turn, implies that $\inf \underline{CCCo}(E) - E \neq \emptyset$. Moreover, all the strings in $\inf \underline{CCCo}(E) - E$ are sequences of events in which the protocol fails but which resemble successful runs of the protocol. Consider the strings t, t' and the event σ above; it can be shown that $t\sigma \in \inf \underline{CCCo}(E)$ as follows. (First note that by definition $\inf \underline{CCCo}(E)$ is prefix-closed so we write $\inf \underline{CCCo}(E)$ for $\overline{\inf \underline{CCCo}(E)}$.) Since $t, t'\sigma \in E$, we have that $t, t'\sigma \in \inf \underline{CCCo}(E)$ (since $E \subseteq \inf \underline{CCCo}(E)$, by definition), and as stated above, $t\sigma \in L(G)$; then, since $\inf \underline{CCCo}(E)$ is co-observable (by definition) and $P_1(t) = P_1(t')$, we have that $t\sigma \in \inf \underline{CCCo}(E)$. Moreover, as stated above, $t\sigma \notin E$, so $t\sigma \in \inf \underline{CCCo}(E) - E$.

Suppose that, in addition, to finding out that the sequence of events $t\sigma$ would cause a protocol failure, we would like to determine why, then we can proceed as follows. Since $\sigma \in \Sigma_{1,c} \setminus \Sigma_{2,c}$, we know that conjunct (2) of co-observability must fail and hence we know that there exists at least one string t' such that $P_1(t) = P_1(t')$ and $t'\sigma \in E$. We can generate all such strings by computing the language $P_1^{-1}(P_1(t\sigma)) \cap E$. These strings are the successful runs of the protocol that resemble the "bad" run $t\sigma$.

The protocol designer can use the test for controllability and co-observability not only to test whether the candidate protocol is correct but also to find out which sequences of events are problematic. That is, sequences of events that are not legal (i.e., are not in the given language E) but are nevertheless possible runs of the protocol (i.e., are generated by the given plant G), are problematic if they look like other successful runs of the protocol (i.e., their projections are the same) and cannot be disambiguated by one of the agents (i.e., one of the conjuncts of co-observability does not obtain). The designer can then compute the language $\inf \underline{CCCo}(E) - E$ to determine scenarios that would cause the protocol to fail.

4 Conclusions

We have shown that for communication problems that can be formulated as cases of our decentralized supervisory control problem, our control-theoretic results can be used to verify if a given protocol for the communication problem is partially correct. Our procedure is automated and allows a protocol designer to systematically verify if a protocol satisfies the intended safety requirements. In addition, if our check indicates that a protocol would fail, then the designer may get additional insight into the problem by generating all sequences of events that would lead to failures. However, we do not as yet have any mechanism for error recovery. Moreover, our formalism does not, at present, permit liveness properties to be verified. To address the issue of liveness, we would need to extend our model to include infinite-string semantics.

References

[1] G. V. Bochmann. Finite state description of communication protocols. *Communication Networks*, 2(4/5):361–372, Oct. 1978.

[2] G. V. Bochmann and J. Gecsei. A unified method for the specification and verification of protocols. In *Information Processing 77, Proc. of IFIP Congress 77*, pages 229–234, Toronto, Canada, Aug. 1977.

[3] G. V. Bochmann and C. A. Sunshine. Formal methods in communication protocol design. *IEEE Trans. Comm.*, COM-28(4):624–631, Apr. 1980.

[4] D. Brand and P. Zafiropulo. Synthesis of protocols for an unlimited number of processes. In *Proc. IEEE 1980 Trends & Applications: Computer Network Protocols*, pages 29–40, Gaithersburg, MD, May 1980.

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

[6] B. T. Hailpern and S. S. Owicki. Modular verification of computer communication protocols. *IEEE Trans. Comm.*, COM-31(1), Jan. 1983.

[7] H. Hansson, B. Johnson, F. Orava, and B. Perhson. Specification for verification. In *Proc. Second International Conf. on Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE '89)*, pages 347–364, Vancouver, Canada, December 5–8 1989.

[8] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1985.

[9] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[10] P. J. Ramadge. *Control and Supervision of Discrete Event Processes.* PhD thesis, Dept. of Elec. Eng., Univ. of Toronto, 1983.

[11] P. J. Ramadge and W. M. Wonham. Supervision of discrete event processes. In *Proc. 21st IEEE Conf. on Decision and Control*, volume 3, pages 1228–1229, Dec. 1982.

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

[13] 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 Sci. Pub. (North-Holland), 1990.

[14] K. Rudie and W. M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Automat. Contr.*, 1992. To appear. An abbreviated version appears in *Proc. American Control Conference (ACC)*, Boston, June 26–28, pp. 898–903, 1991.

[15] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall, Inc., Englewood-Cliffs, NJ, second edition, 1988.

[16] C. H. West. General technique for communications protocol validation. *IBM J. of Research and Development*, 22(4):393–404, July 1978.

[17] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3):637–659, 1987.

[18] P. Zafiropulo, C. H. West, H. Rudin, D. D. Cowan, and D. Brand. Towards analyzing and synthesizing protocols. *IEEE Trans. Comm.*, COM-28(4):651–661, 1980.