

Input/Output Discrete Event Processes and Communication Delays

SILVANO BALEMI

Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), 8092 Zurich, Switzerland

Received November 18, 1992; Revised May 21, 1993

Abstract. This paper proposes an input/output interpretation for the control of discrete event systems. In supervisory control theory, the plant is usually considered to be a process which spontaneously generates outputs. Its behavior is controlled by disabling some of its output events. In this paper the plant accepts inputs (commands) and generates outputs (responses). For many real-life problems, e.g., manufacturing problems, this model semantics seems more applicable than the original generator perspective proposed by Ramadge and Wonham. In real-life systems the message exchange between the plant and the controller is affected by communication delays. The class of supervisors that must be used in a closed-loop system subject to communication delays is a subset of the set of solutions to the original problem introduced by Ramadge and Wonham. After having characterized this class of supervisors, the present paper states conditions on the plant language that allow the supervisors solving the (delay-free) supervisory control problem of Ramadge and Wonham to be used also for the case of closed-loop systems affected by delays.

Key Words: input/output processes, communication delays, supervisory control theory, discrete event systems.

1. Introduction

The field of supervisory control theory initiated by Ramadge and Wonham [1989] provides a framework for the treatment of theoretical and computational issues related to the control of discrete event systems. However, the interpretation and implementation of the results obtained in this research area is not always straightforward. This is partly due to the model semantics of a plant spontaneously generating events with a supervisor only passively preventing the generation of some of the events.

This model semantics is not appropriate for modeling many real-life systems because it does not naturally allow the supervisor to generate events of its own. A partial step in the direction of a supervisor able to force the occurrence of certain events in the plant was done by Golaszewski and Ramadge [1987]. In the present paper we use an explicit input/output semantics for the plant. The plant reads inputs, called commands, and produces outputs, called responses. Symmetrically, a controller accepts as inputs the responses produced by the plant and generates the commands as outputs.

In modeling real systems using this input/output semantics one soon encounters the issue of delays in the communication of messages (commands and responses) between the plant and the controller. In some cases, the delays can be taken care of with the design of an appropriate communication protocol, requiring for instance the acknowledgement of the receipt of some messages before a new one can be sent. However, in other cases such as manufacturing systems, the plant and the controller spontaneously produce outputs (responses respectively commands) and no such protocol can be designed. Then, it is the supervisor and/or the plant which must be designed in such a way that possible delays have no influence on the closed-loop behavior.

1.1. Motivation

The motivation for this work is provided by an application: the rapid thermal multiprocessor (RTM), a semiconductor manufacturing piece of equipment developed at the Center for Integrated Systems of Stanford University [Saraswat et al. 1989]. The RTM consists of a processing chamber in which a single wafer can undergo several operations such as cleaning, annealing, oxidation and chemical vapor deposition. The project of designing the control software for this machine was aimed at proving the applicability of supervisory control to real-life problems.

A first necessary step was the introduction of a full input/output semantics in order to satisfactorily model the reactive nature of the RTM. In fact, the RTM needs explicit commands (e.g., providing actuator commands) to which it reacts with responses (e.g., reporting sensor readings).

Second, the design of the control software, because of its distributed implementation, had to take care of delays in the communication of commands and responses between the plant and the controller. Thus, some techniques were developed in order to extend supervisory control theory to deal with systems affected by communication delays. Particular emphasis was put on methods which allowed the direct use of controller synthesis algorithms already present in the literature.

Some of the theoretical concepts that arose from this effort are reported in the present paper. For a specific report on the design and implementation of the control software of the above-mentioned application the reader is referred to Balemi [1992b], Hoffmann and Wong-Toi [1992], and Balemi et al. [1993].

1.2. Organization of the Paper

In Sections 3 and 4 the input/output semantics described above is introduced by means of the concept of prioritized synchronous composition [Heymann 1990]. The outputs of the plant and of the supervisor in the closed-loop are the priority sets of the respective processes. For one process to be able to track the outputs of the other process, mutual completeness of supervisor and plant (called well-posedness) is required. A comparison between the original problem of Ramadge and Wonham and the control problem with the new semantics is given in Section 4.3.

In Section 6, communication delays are first analyzed by assuming that the plant and the specification are described by prefix-closed formal languages. The concept

of well-posedness is extended to the case when the closed loops are affected by communication delays. The controller synthesis problem with delays is formulated in Section 6.4. In Sections 6.5 and 6.6 simpler solutions to the control problem are given when the plant language satisfies certain conditions (self-well-posedness or memorylessness). In Section 7 the control problem for plants described by a marked language is presented. Finally, some comments on the applicability of the results presented in this paper are given in Section 8.

2. Preliminaries and Notation

The behavior of discrete event systems (DES) is naturally described by records (or *traces*) of certain qualitative changes (or *events*) in the system, by ignoring microchanges occurring between two events.

Ramadge and Wonham proposed a formal description of the behavior of DES using formal languages. Their main idea is to represent events by *symbols*. The finite set of all such symbols $\{\sigma', \sigma'', \dots, \sigma^{(N)}\}$ is called an *alphabet* and is denoted by Σ . Any sequence of events can be therefore represented by a sequence of symbols $s = \sigma_1.\sigma_2.\sigma_3.\dots.\sigma_n$, called a *string* (or word) over the alphabet Σ . The set of all possible strings composed of elements from Σ is denoted by Σ^* . A set $L \subseteq \Sigma^*$ is called a *language* over the alphabet Σ . Thus, the behavior of a system, i.e., the collection of all possible traces, can be described by a language L .

The concatenation $s.s'$ of two strings $s = \sigma_1.\sigma_2.\dots.\sigma_n$ and $s' = \sigma'_1.\sigma'_2.\dots.\sigma'_m$ is defined to be $s.s' = \sigma_1.\sigma_2.\dots.\sigma_n.\sigma'_1.\sigma'_2.\dots.\sigma'_m$. The empty string is denoted by the symbol ϵ . A finite string $s' \in \Sigma^*$ is a *prefix* of s if there exists a string $t \in \Sigma^*$ such that $s'.t = s$. The set of all prefixes of the strings in L is denoted by \bar{L} . We say the language L is *prefix-closed* if all prefixes of the strings in L are also contained in L , i.e., if $L = \bar{L}$.

The *natural projection* of a language L on the alphabet Σ' is the language obtained by removing from the strings in L all occurrences of symbols not contained in the alphabet Σ' . It is denoted by $\mathcal{P}[\Sigma'](L)$. Its converse for a language $L' \subseteq (\Sigma')^*$ is $\mathcal{P}^{-1}[\Sigma'](L') = \sup\{L \subseteq \Sigma^* \mid \mathcal{P}[\Sigma'](L) = L'\}$.

2.1. Language Representation via Finite-State Automata

An important representation of formal languages, which is both practical and suitable for computation, is the deterministic finite-state automaton. A *deterministic finite-state automaton* \mathcal{A} [Hopcroft and Ullman 1979] is a five-tuple $(\Sigma, Q, \delta, q_0, Q_m)$, where Q is a finite set of states, Σ is a finite alphabet of symbols, denoting state transitions, $\delta : Q \times \Sigma \mapsto Q$ is a partial transition function mapping a state and a symbol to a state, q_0 is the initial state and Q_m is the set of marked states.

An *accepting run* of \mathcal{A} on the string $s = \sigma_1.\sigma_2.\dots.\sigma_n$, $s \in \Sigma^*$, is a sequence $q_0.q_1.q_2.\dots.q_n$ of $n+1$ states in Q , starting with the initial state q_0 , $q_{i+1} = \delta(q_i, \sigma_{i+1})$ for $0 \leq i < n$, and $q_n \in Q_m$. The language generated by \mathcal{A} , denoted $L(\mathcal{A})$, is the set of all strings s with accepting runs. In other words, the language of a deterministic automaton is the set of all sequences of symbols corresponding

to sequences of state transitions leading from the initial state to a marked state. The languages that can be generated by finite-state automata are only a subset of all languages \mathcal{L} and are called *regular languages*. A language L is *regular* if and only if there is some finite-state automaton \mathcal{A} such that $L(\mathcal{A}) = L$.

3. Review of Supervisory Control Theory

The theory of supervisory control introduced by Ramadge and Wonham [1987a] uses formal languages to model both the plant and its specification. In this section we restate the main result of supervisory control theory by using a process model for discrete event systems. A similar approach was already taken by Heymann [1990], by Golaszewski and Kurshan [1991], and by the author [Balemi 1991, Balemi et al. 1993].

3.1. The Prioritized Synchronous Composition

A *process* is a triple $P = (\Sigma, L_P, M_P)$ composed of two languages of finite traces over the alphabet Σ . L_P is a prefix-closed language (i.e., $L_P = \overline{L_P}$) representing all possible traces of P , while $M_P \subseteq L_P$ stands for a set of distinguished traces, the *marked language*. M_P can indicate the set of successfully *completed* traces.

The main process operator is the *prioritized synchronous composition* operator [Heymann 1990]. Before we can define this formally, we need additional notation. The *generalized projection* of a string s onto a prefix-closed language L , denoted by $\mathcal{GP}[L](s)$, is a string defined recursively by $\mathcal{GP}[L](\epsilon) = \epsilon$ and with $\sigma \in \Sigma$ by

$$\mathcal{GP}[L](s.\sigma) = \begin{cases} \mathcal{GP}[L](s).\sigma & \text{if } \mathcal{GP}[L](s).\sigma \in L, \\ \mathcal{GP}[L](s) & \text{otherwise.} \end{cases}$$

Thus, the generalized projection of a string onto a language yields the string which is obtained from the given string by discarding in a row those symbols that could not “follow” the language. Consider, for instance, two strings $s = a.b.c.a$ and $s' = b.a.c.d$ and a language $L = \overline{\{a.c, b.d.a\}}$. Then, $\mathcal{GP}[L](s) = a.c$ and $\mathcal{GP}[L](s') = b.d$. The expression $\mathcal{GP}[L](K)$ denotes the obvious extension to the generalized projection of a language K . Note that for an arbitrary alphabet Σ' , $\mathcal{P}[\Sigma'](K) = \mathcal{GP}[(\Sigma')^*](K)$. We are now ready to define the composition operator.

DEFINITION 3.1 (Prioritized Synchronous Composition). The prioritized synchronous composition $P_{1A} \parallel_B P_2$ of $P_1 = (\Sigma_1, L_{P_1}, M_{P_1})$ and $P_2 = (\Sigma_2, L_{P_2}, M_{P_2})$ with respect to the sets $A \subseteq \Sigma_1$ and $B \subseteq \Sigma_2$ is

$$P_{1A} \parallel_B P_2 = (\Sigma_1 \cup \Sigma_2, L_{P_{1A} \parallel_B P_2}, M_{P_{1A} \parallel_B P_2}), \quad (1)$$

where $L_{P_{1A} \parallel_B P_2}$ is defined by $\epsilon \in L_{P_{1A} \parallel_B P_2}$ and by the recursive condition that for $s \in L_{P_{1A} \parallel_B P_2}$ and $\sigma \in \Sigma_1 \cup \Sigma_2$ the string $s.\sigma$ is in $L_{P_{1A} \parallel_B P_2}$ if and only if the

following conditions are satisfied:

$$\mathcal{GP}[L_{P_1}](s).\sigma \in L_{P_1} \wedge \mathcal{GP}[L_{P_2}](s).\sigma \in L_{P_2} \quad \forall \sigma \in A \cap B, \quad (2a)$$

$$\mathcal{GP}[L_{P_1}](s).\sigma \in L_{P_1} \quad \forall \sigma \in A - B, \quad (2b)$$

$$\mathcal{GP}[L_{P_2}](s).\sigma \in L_{P_2} \quad \forall \sigma \in B - A, \quad (2c)$$

$$\mathcal{GP}[L_{P_1}](s).\sigma \in L_{P_1} \vee \mathcal{GP}[L_{P_2}](s).\sigma \in L_{P_2} \quad \forall \sigma \notin B \cup A. \quad (2d)$$

The marked language $M_{P_1 \parallel_B P_2}$ is described by

$$M_{P_1 \parallel_B P_2} = \left\{ s \in L_{P_1 \parallel_B P_2} \mid \mathcal{GP}[L_{P_1}](s) \in M_{P_1} \wedge \mathcal{GP}[L_{P_2}](s) \in M_{P_2} \right\}. \quad (3)$$

The sets A and B are called the priority sets of process P_1 and P_2 respectively. The priority set of a process indicates the events that can occur in the composition only if the process agrees on their execution. In particular, the introduction of priority sets leads to the partition of the events into four classes.

- (a) $A \cap B$: *the strict synchronization events*, executed by both processes or none.
- (b) $A - B$: *the priority events of P_1 only*. These events can occur only if process P_1 participates. If process P_2 can, it must execute the events simultaneously with process P_1 .
- (c) $B - A$: *the priority events of P_2 only*. Symmetrical to the set $A - B$.
- (d) $(\Sigma_1 \cup \Sigma_2) - A - B$: *the broadcasting events*. Any of the two processes can execute such an event independently. However, if both processes can execute the event, then they can execute it only jointly.

The marked language of the composition is the set of strings that are marked by both processes. We can see an example illustrating the prioritized synchronous composition of two processes in Figure 1.

The special case when the priority set of the two processes is the process's own alphabet, i.e., when $A = \Sigma_1$ and $B = \Sigma_2$, is called *full synchronous composition* and is denoted by $P_1 \parallel P_2 = P_{1 \Sigma_1 \parallel \Sigma_2} P_2$. Two processes subject to full synchronization must simultaneously execute common events (in $\Sigma_1 \cap \Sigma_2$), while the other events occur independently in each process.

3.2. Supervisor Synthesis Problem

The basic problem of supervisory control is to modify the open-loop behavior of the plant by eliminating event sequences from the plant behavior. For a plant process $P = (\Sigma, L_P, M_P)$, the behavior modification is achieved by synchronization of the plant process with another process $S = (\Sigma, L_S, M_S)$, called supervisor.

In the synthesis procedure we are free to choose the supervisor while the plant process is given and cannot be altered. The plant process P and the supervisor process S jointly executing events are described by the full synchronous composition $P \parallel S = (\Sigma, L_{P \parallel S}, M_{P \parallel S})$.

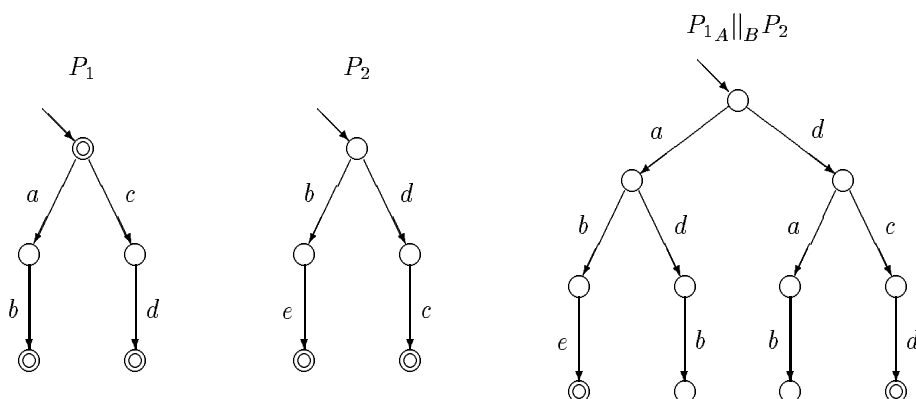


Figure 1. Example of prioritized synchronous composition $P_1 A ||_B P_2$ of two processes P_1 and P_2 with respective alphabets $\Sigma_1 = \{a, b, c, d\}$ and $\Sigma_2 = \{b, c, d, e\}$ and priority sets $A = \{b, c\}$ and $B = \{c, e\}$. The processes are here represented by automata: the arrows on the top point to the initial state, the states marked by two concentric circles are marked states.

Ramadge and Wonham postulate that not all events of the plant can be forced to synchronize with the supervisor. The event set Σ of the plant is correspondingly partitioned into *controllable* events Σ_c and *uncontrollable* events Σ_u . The occurrence of controllable events can be prevented from occurring by synchronization of the plant with the supervisor, while the uncontrollable events are those over which the supervisor has no control authority.

Therefore, a supervisor process should not be chosen arbitrarily. In fact, as the supervisor cannot prevent the occurrence of uncontrollable events, we are interested only in those supervisors that can always track uncontrollable events generated by the plant; then the joint behavior of plant and supervisor can still be described by the composition $P||S$. The closed loop of the plant with such a supervisor can then be interpreted as in Figure 3 (left side); the supervisor passively tracks all events generated by the plant but has the ability of disabling or enabling the occurrence of the controllable events only. For a formal characterization of such supervisors, we introduce the definition of completeness.

DEFINITION 3.2 (Complete Supervisor). A supervisor S is called *complete* for plant P if

$$P||S = P_{\Sigma}||_{\Sigma_c} S \quad (4)$$

holds.

Intuitively, a complete supervisor is such that it can always track the uncontrollable events produced by the plant either with or without the ability to prevent them. A supervisor S is complete for plant P if and only if the following equation

is satisfied:

$$L_S.\Sigma_u \cap L_P \subseteq L_S. \quad (5)$$

Proof. Consider a string s in the language $L_{P\parallel S} = L_P \cap L_S$ of the process $P\parallel S = P_{\Sigma}\parallel_{\Sigma}S$ and a symbol $\sigma \in \Sigma$. Then, from the definition of prioritized synchronous composition, the condition for a string $s.\sigma$ to belong to $L_{P\parallel S}$ is given by (2a). Consider now, instead, a string s in the language of the process $P_{\Sigma}\parallel_{\Sigma_c}S$. Then, $s.\sigma$ is also in this language if condition (2a) is satisfied when $\sigma \in \Sigma_c$ or condition (2b) is satisfied when $\sigma \in \Sigma_u$. Equation (4) therefore means that for any $\sigma \in \Sigma_u$ the two conditions (2a) and (2b) are equivalent, i.e., that

$$\mathcal{GP}[L_P](s).\sigma \in L_P \Rightarrow \mathcal{GP}[L_S](s).\sigma \in L_S.$$

Noting that $\mathcal{GP}[L_P](s) = \mathcal{GP}[L_S](s) = s$ for any string $s \in L_P \cap L_S$, the above implication yields

$$\begin{aligned} s.\sigma \in L_P &\Rightarrow s.\sigma \in L_S && \forall \sigma \in \Sigma_u, \quad s \in L_P \cap L_S, \\ (L_P \cap L_S).\sigma \cap L_P &\subseteq L_S && \forall \sigma \in \Sigma_u, \\ (L_P \cap L_S).\Sigma_u \cap L_P &\subseteq L_S, \\ L_S.\Sigma_u \cap L_P &\subseteq L_S, \end{aligned}$$

proving the desired inclusion. The other direction of the statement can be proved by reversing the order of the above steps. Q.E.D.

In addition, we further restrict our attention to the class of supervisors that also guarantee that a sequence of events in the closed loop may always be completed to a marked string in both plant and supervisor. For this we now introduce the concept of nonblockingness [Ramadge and Wonham 1989].

DEFINITION 3.3 (Nonblocking Process). A process P is *nonblocking* if

$$\forall s \in L_P \Rightarrow s.(L_P/s) \cap M_P \neq \emptyset$$

Here, the language L/s denotes the post-language of L after string s , i.e., the set of all strings t such that $s.t \in L$. In other words, a process is nonblocking if any string in L_P can be completed to a marked string in M_P , i.e., if for all strings $s \in L_P$ there exists a string t such that $s.t \in M_P$. In particular, a process P is nonblocking if and only if $L_P = \overline{M_P}$.

We are now ready to restate the standard supervisor synthesis problem [Ramadge and Wonham 1987b].

SUPERVISOR SYNTHESIS PROBLEM. *Given a plant $P = (\Sigma, L_P, M_P)$, and a specification language $M_{\text{spec}} \subseteq \Sigma^*$ for the closed-loop behavior, find a supervisor $S = (\Sigma, L_S, M_S)$ such that*

1. $P\parallel S$ is nonblocking.

2. $\emptyset \subset M_{P\parallel S} \subseteq M_{\text{spec}}$.
3. S is complete for P .

In particular, we note that if $P\parallel S$ is nonblocking, then any string in $L_{P\parallel S}$ can be extended to a string in $M_{P\parallel S}$, i.e., to a string that is marked both in the plant and in the supervisor.

Ramadge and Wonham [1987b] introduced the notion of *controllability* to characterize the supervised languages of the plant $P = (\Sigma, L_P, M_P)$. A language $K \subset \Sigma^*$ is called $[L_P, \Sigma_c]$ -controllable if the inclusion $\overline{K}.\Sigma_u \cap L_P \subseteq \overline{K}$ holds. Note that completeness of the supervisor requires $[L_P, \Sigma_c]$ -controllability of L_C , as shown by inclusion (5).

The class $\mathcal{C}(L)$ of $[L_P, \Sigma_c]$ -controllable sublanguages of a language L is closed under language union; therefore there exists a supremal element of this class, which we denote by $\sup \mathcal{C}(L)$. The following theorem from [Ramadge and Wonham 1987b, Theorem 7.1] gives a necessary and sufficient condition for the existence of a solution to the supervisor synthesis problem.

THEOREM 3.1. The supervisor synthesis problem has a solution if and only if $\emptyset \subset \sup \mathcal{C}(M_P \cap M_{\text{spec}})$.

The particular supervisor

$$S_{\text{sup}} = (\Sigma, \overline{\sup \mathcal{C}(M_P \cap M_{\text{spec}})}, \sup \mathcal{C}(M_P \cap M_{\text{spec}})) \quad (6)$$

is a solution if and only if $\sup \mathcal{C}(M_P \cap M_{\text{spec}})$ is nonempty. It is also the (nonunique) least restrictive supervisor in the sense that it does not prevent any sequence of events that is allowed by any other supervisor that meets the specification. Wonham and Ramadge [1987] proposed an algorithm for the computation of $\sup \mathcal{C}(L)$ for a given marked language L . Hence, the supervisor S_{sup} can be computed by applying their algorithm to the language $L = M_P \cap M_{\text{spec}}$.

4. An Input/Output Semantics

4.1. Plant Model

The model interpretation proposed by Ramadge and Wonham consists of a plant spontaneously generating events; the only way to affect the behavior of the plant is by enabling or disabling controllable events. In their semantics, the plant alone *schedules* the occurrence of both controllable and uncontrollable events. This model interpretation is graphically rendered by the left-hand side of Figure 2.

In our opinion the model of a plant generating all events is not accurate for most real systems; an input/output perspective is required. In fact, events do not usually occur spontaneously, but only as *responses* to *commands*. For instance, system actuators are activated by commands, while responses report changes detected by sensors.

The formal plant model of Section 3 consists of a process $P = (\Sigma, L_P, M_P)$ with the alphabet Σ partitioned into two disjoint subalphabets $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. The

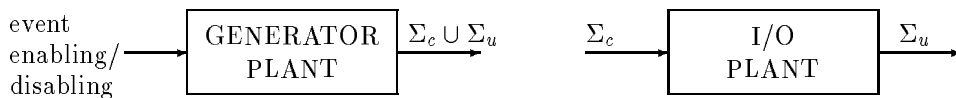


Figure 2. Generator plant and input/output plant.

partition of Σ is now interpreted differently. The elements of Σ_c model the inputs of the plant, whereas the elements of Σ_u stand for the plant outputs. From now on we refer to the inputs as *commands* and to the outputs as *responses* (see right-hand side of Figure 2).

4.2. Supervisor Model

In the original model, the supervisor acts as a passive device, tracking events produced by the plant and restricting the behavior of the plant by dynamically disabling the controllable events (see Figure 3, left-hand side).

In the input/output model, the supervisor does not simply prevent controllable events from occurring (by means of the synchronization $P||S$) but actually sends commands to the input of the plant. The generation of events is therefore initiated not only by the plant, but by both the plant and the supervisor. Commands are produced by the supervisor, responses by the plant (see Figure 3, right-hand side).

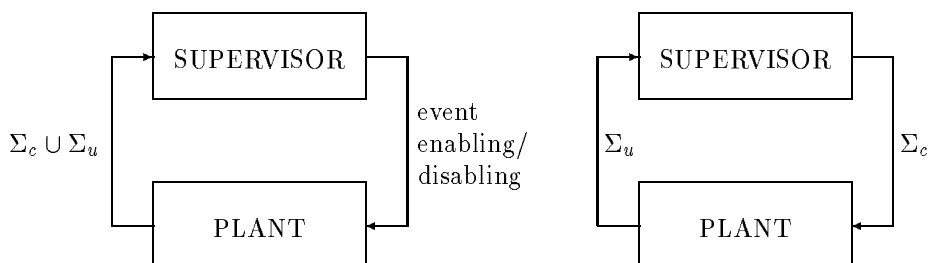


Figure 3. Feedback loops with generator plant and disabling supervisor (left) and with I/O plant and I/O supervisor (right).

4.3. Closed-Loop System

Ramadge and Wonham assume in their model that the set of uncontrollable events cannot be prevented from occurring. The uncontrollable events will occur regardless of whether the supervisor can “accept” them. Both in the original Ramadge and Wonham model and from an input/output perspective, equation (4) can be understood as the condition under which the supervisor is always able to synchronize with the plant on an event in Σ_u arbitrarily produced by the plant.

The input/output view of the connection of plant and supervisor leads to the analogous assumption that also the plant cannot prevent the occurrence of commands generated by the supervisor. The assumption implies that the plant must be complete for the supervisor.

DEFINITION 4.1 (Complete Plant). A plant P is *complete for supervisor* S if

$$P||S = P_{\Sigma_u}||_{\Sigma}S \quad (7)$$

holds.

Equation (7) can be seen as a *dual* to equation (4). Similarly to (5), a plant P is complete for supervisor S if and only if the equation

$$L_P.\Sigma_c \cap L_S \subseteq L_P \quad (8)$$

is satisfied. The mutual completeness of supervisor and plant can be simultaneously expressed by the equality $P||S = P_{\Sigma_u}||_{\Sigma_c}S$. Remark that Σ_u and Σ_c are then the disjoint priority sets of the plant process P and of the supervisor process S respectively.

We say the composition of a supervisor and a plant is *well-posed* if the supervisor and the plant are mutually complete. This is analogous to the concept of well-posedness of the interconnection of linear systems in control theory, where well-posedness means that there exists a state space description of the interconnection, i.e., that the interconnection “makes sense.” In our case, well-posedness of the composition means that no message is ever lost, i.e., that a message arriving at a process input can always be accepted.¹

4.4. Controller Synthesis Problem

To distinguish the newly introduced input/output semantics from the generator disabling semantics, we call the input/output supervisor a “controller.” We are now ready to present the synthesis problem which incorporates the new input/output semantics and the additional condition (7).

CONTROLLER SYNTHESIS PROBLEM. *Given a plant $P = (\Sigma, L_P, M_P)$ and a specification language $M_{\text{spec}} \subseteq \Sigma^*$ for the closed-loop behavior, find a controller $C = (\Sigma, L_C, M_C)$ such that*

1. $P||C$ is nonblocking.
2. $\emptyset \subset M_{P||C} \subseteq M_{\text{spec}}$.
3. the connection of C and P is well-posed.

The third condition requires that the supervisor and the plant be mutually complete, i.e., that both conditions (4) and (7) be satisfied. Before stating the condition for the existence of a solution to the problem, we need the following lemma.

LEMMA 4.1. Let $P = (\Sigma, L_P, M_P)$ be a plant and $S = (\Sigma, L_S, M_S)$ be a supervisor. If $L_S \subseteq \overline{M_P}$ then equation (7) is satisfied.

Proof. We note that $L_S \subseteq \overline{M_P} \subseteq L_P$ implies $L_P \cdot \Sigma_c \cap L_S \subseteq L_P \cdot \Sigma_c \cap L_P \subseteq L_P$. Therefore, (8) is satisfied, and this is equivalent to (7). Q.E.D.

We can now state the following theorem.

THEOREM 4.1. The controller synthesis problem has a solution if and only if $\emptyset \subset \sup \mathcal{C}(M_P \cap M_{\text{spec}})$.

Proof. (ONLY IF) follows directly from Theorem 3.1.

(IF) The condition presented in the statement of the theorem implies that there exists a solution to the supervisor synthesis problem. Then, the supervisor S_{sup} of equation (6) is such a solution. Moreover, this supervisor also satisfies the additional condition (7) because of Lemma 4.1, and therefore it is also a solution to the controller synthesis problem. Q.E.D.

In Figure 4 we can see graphically the set of languages M_S of supervisors $S = (\Sigma, \overline{M_S}, M_S)$ solving the supervisor synthesis problem and the set of languages M_C of controllers $C = (\Sigma, \overline{M_C}, M_C)$ solving the controller synthesis problem for the same given plant and specification languages. The set solving the controller synthesis problem is contained in the set solving the supervisor synthesis problem; the additional mathematical constraint given by equation (7) reduces the solution set. However, as shown by Theorem 4.1, this additional constraint does not affect the condition for existence of a solution: the two solution sets will always be either both empty or both nonempty.

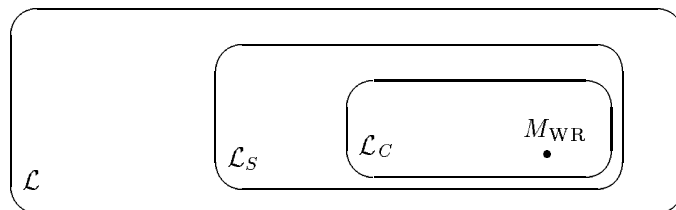


Figure 4. \mathcal{L} : set of all languages. $\mathcal{L}_S, \mathcal{L}_C$: set of languages of supervisors respectively controllers solving the supervisor respectively the controller synthesis problem. M_{WR} : language computed by the synthesis algorithm of Wonham and Ramadge.

If a solution to the controller synthesis problem exists, any supervisor S that solves the supervisor synthesis problem *and* that satisfies $L_S \subseteq \overline{M_P}$ is a solution to the controller synthesis problem. Then, the synthesis algorithm of Wonham and Ramadge, computing the particular supervisor S_{sup} of equation (6), also yields a solution to the controller synthesis problem. Moreover, this is the least restrictive solution. If a supervisor S solves the supervisor synthesis problem, but $L_S \not\subseteq \overline{M_P}$ (e.g., if S is a reduced supervisor [Vaz and Wonham 1986]), the controller $C = S \parallel P$

is a solution to the controller synthesis problem. Thus, any supervisor solving the supervisor synthesis problem can be used (directly or composed with a copy of the plant process) to obtain a solution to the controller synthesis problem.

4.5. Controller Synthesis Problem with Local Specifications

Until here we have considered the specification on the behavior of the controlled system as given by a language $M_{\text{spec}} \subset \Sigma^*$. In the sequel, however, we restrict our attention to so called *local specifications* given on the responses Σ_u only. Then condition 2 of the controller synthesis problem becomes

$$2. \quad \emptyset \subset \mathcal{P}[\Sigma_u](M_{P\parallel C}) \subseteq M'_{\text{spec}} \subset \Sigma_u^*.$$

This type of specification implies that we are only interested in making sure that the sequences of messages observed at the output of the plant (the sequence of responses) satisfy given properties. Thus, it is unimportant to know what commands and when these commands are sent to the plant input as long as the resulting response sequences are acceptable to us. This may seem very restrictive to readers acquainted with the framework of Ramadge and Wonham. However, in our opinion this is not a relevant restriction in an input/output perspective. In fact, commands represent the request of some desired changes, and we do not want to impose any constraint on simple requests. In order to model the limited availability of resources, additional responses can be embedded in the language of the plant at modeling time. Moreover, the restriction to this class of specifications will be of importance later dealing with communication delays. The condition for the existence of a solution to the problem with local specification is given by the next corollary.

COROLLARY 4.1. The controller synthesis problem with local specification $M'_{\text{spec}} \subset \Sigma_u^*$ has a solution if and only if $\emptyset \subset \mathcal{P}[\Sigma_u](\text{sup } \mathcal{C}(M_P \cap \mathcal{P}^{-1}[\Sigma](M'_{\text{spec}})))$.

Proof. Follows directly from the condition for existence of a solution to the original problem with specification language $M_{\text{spec}} = \mathcal{P}^{-1}[\Sigma](M'_{\text{spec}})$, and from the requirement that a marking must be reached in the projection of $M_{P\parallel C}$. Q.E.D.

Then a solution to the controller synthesis problem with local specification $M'_{\text{spec}} \subset \Sigma_u^*$ is given by the controller

$$C = (\Sigma, \overline{\text{sup } \mathcal{C}(M_P \cap \mathcal{P}^{-1}[\Sigma](M'_{\text{spec}}))}, \text{sup } \mathcal{C}(M_P \cap \mathcal{P}^{-1}[\Sigma](M'_{\text{spec}}))). \quad (9)$$

EXAMPLE. Consider the example of Figure 5, showing a plant language on the left and a local specification language on the right. Using (9) we find the least restrictive controller solving the controller synthesis problem for this plant and specification. Its language is shown in Figure 6.

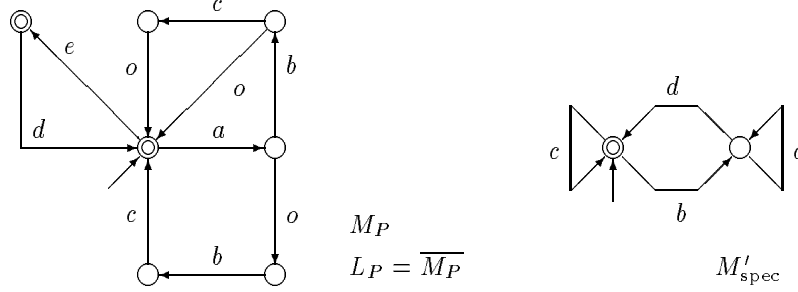


Figure 5. Plant languages L_P and M_P and local specification language M'_{spec} . Vowels (the set $\{a, e, o\}$) represent commands while responses are given by consonants (the set $\{b, c, d\}$).

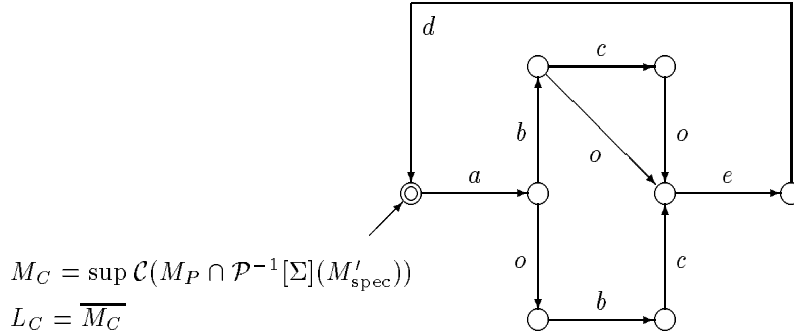


Figure 6. Automaton representing the least restrictive controller solving the controller synthesis problem for the plant and the specification given in Figure 5.

4.6. Input/Output Processes

In accordance with the input/output perspective, we redefine the process to include also its *priority set*, denoted for a process P by Σ_P . This set represents also the *outputs* of the process while the complementary set $\Sigma - \Sigma_P$ denotes the *inputs*. For the sequel, an input/output process P will be described by a quadruple $(\Sigma, \Sigma_P, L_P, M_P)$ where Σ is the alphabet of the process containing the inputs and the outputs of the system, Σ_P is the set of outputs, L_P the language and M_P the marked language of the process. We extend the full synchronous composition of two processes to include the outputs by defining the output set of the composed process as the union of the output sets of the single processes.

For a controller C and a plant P , the set Σ_C (until now Σ_c) stands for the outputs produced by the controller (the commands), while the set Σ_P (until now Σ_u) represent for the outputs generated by the plant (the responses).

5. Closed-Loop System Subject to Communication Delays

So far we have implicitly assumed that a message sent by the plant instantaneously reaches the controller, and vice versa. In general, however, the connection between the plant and the controller is affected by delays. A message sent by a process needs some time to reach the other process.

Here, we are concerned about the way communication delays affect the logical behavior of the controlled system. Real-time models of processes (and thus of communication delays) are beyond the scope of this paper. Within the original model of Ramadge and Wonham, Li and Wonham [1987] presented some results on supervision of discrete event systems affected by delays. However, the different model semantics does not allow the use of their results for input/output systems.

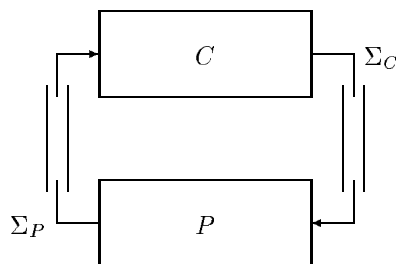


Figure 7. Closed-loop connection of processes P and C with communication delays.

In our model we introduce communication delays by considering the two processes as being connected by two communication channels, one from the output of the plant P to the input of the controller C and one from the output of the controller C to the input of the plant P (see Figure 7). Like before, we suppose that both the controller and the plant have the same alphabet $\Sigma = \Sigma_P \dot{\cup} \Sigma_C$, with Σ_P being the outputs of the plant and Σ_C the outputs of the controller. The initial condition of the channels is such that they contain no messages. The messages put by a process into the respective channel appear only with some delay at the exit. However, the relative order of the transmitted messages is unaffected: they exit the channel in the same order as they entered it. The channels can therefore be seen as FIFO buffers. We suppose that the channels have infinite buffering length: an infinite number of messages can be put into a channel without any exit at the channel output. Finite buffers as *models* for the channels are not suitable in combination with the input/output processes described in the previous section because the outputs of these processes are spontaneously generated and cannot be prevented by possibly full communication channels. If the buffering capacity of the channels is finite, then channel overflow can be avoided by making sure that the languages of the processes satisfy given properties [Balemi 1992a]. So, it is possible to limit the number of messages in the channels (to reflect the finite buffering capacity) while being consistent with the input/output process model used here.

In the next section we focus our attention on processes whose marked languages

are prefix-closed. We then characterize in Section 6.2.1 the (different) closed-loop languages of controller and plant. In Section 6.3 we extend the concept of well-posedness to include communication delays. If the closed loop is well-posed, then the closed-loop languages are shown to assume a simpler form. In Section 6.4 we introduce the unmarked controller synthesis problem and state conditions for the existence of a solution.

In Sections 6.5 respectively 6.6 we show that if the plant language is self-well-posed respectively memoryless, the solution to the unmarked controller synthesis problem can be more easily characterized. Finally, in Section 7 we present the marked controller synthesis problem for plants with memoryless languages.

6. Communication Delays and Processes with Prefix-Closed Languages

We first consider the case when the plant and the controller have prefix-closed marked languages, i.e., when they are of the form $P = (\Sigma, \Sigma_P, L_P, L_P)$ respectively $C = (\Sigma, \Sigma_C, L_C, L_C)$. Now we can ask the following question:

“What is the *logical behavior* of the two processes P and C connected in a closed loop *affected by communication delays*?”

Before addressing this question we have to choose a suitable representation for this behavior. One possibility is to register all messages that go into and come from the communication channels. A message σ registered as σ^{enter} when put into a channel will eventually be registered as σ^{exit} when exiting at the other end of the same channel. A set of sequences of the form $\sigma_1^{\text{enter}}.\sigma_2^{\text{enter}}.\sigma_1^{\text{exit}}.\sigma_3^{\text{enter}}.\sigma_2^{\text{exit}}\dots$ would therefore uniquely determine the behavior of the closed-loop system shown in Figure 7. This set of sequences can be seen as the language of a new process of the form $X = (\Sigma^{\text{enter}} \cup \Sigma^{\text{exit}}, \Sigma^{\text{enter}} \cup \Sigma^{\text{exit}}, L_X, L_X)$. The sets Σ^{enter} and Σ^{exit} are the sets of symbols for the messages entering respectively exiting a channel. It is thus possible to formally introduce a “delayed composition” of two processes C and P resulting in a new process X . We abstain from that, however, and we restrict our attention to a simpler description.

The description we choose uses the view of two observers locally registering inputs and outputs of the plant and of the controller. This choice is suggested by the fact that the controller has to influence the behavior of the plant only with the knowledge of the commands and responses that it has processed so far, and that it has to infer from that knowledge the possible behavior of the plant. We note that, in opposition to the case of delay-free communication, the behavior of the plant and the behavior of the controller are not necessarily identical. Then, it is not sufficient to define one closed-loop language as the language of the composed process, but we have to define two local closed-loop languages, L_P^c for the plant and L_C^c for the controller.

6.1. The Delay of a Language

A key concept that we will use to describe the behavior of the controller and of the plant in a closed loop subject to communication delays is the delay of a language.

DEFINITION 6.1 (Delay of a Language). The *delay* of a language $L \subseteq \Sigma^*$ with respect to the subalphabet $\Sigma' \subseteq \Sigma$, denoted by $\text{delay}[\Sigma'](L)$, is defined as the smallest superlanguage of L such that for $s, t \in \Sigma^*$ and any $\sigma' \in \Sigma'$ and $\sigma \in \Sigma - \Sigma'$ the conditions

$$L \subseteq \text{delay}[\Sigma'](L),$$

$$s.\sigma'.\sigma.t \in \text{delay}[\Sigma'](L) \Rightarrow s.\sigma.\sigma'.t \in \text{delay}[\Sigma'](L)$$

are satisfied.

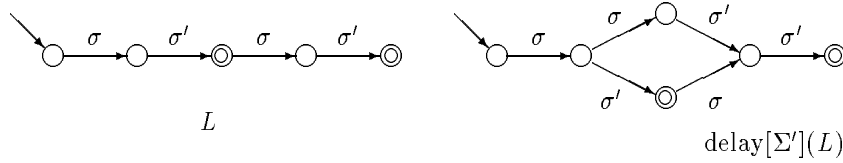


Figure 8. The delay of the language $L = \{\sigma.\sigma', \sigma.\sigma'.\sigma.\sigma'\}$ with respect to the set $\Sigma = \{\sigma'\}$ is the language $\text{delay}[\Sigma'](L) = L \cup \{\sigma.\sigma.\sigma'.\sigma'\}$.

An example of delay of a language is shown in Figure 8. Note that the $\text{delay}[\cdot]$ operator can transform a regular language into a nonregular one. This is shown by the next example.

EXAMPLE. Take $L = (\sigma.\sigma')^* = \{\epsilon, \sigma.\sigma', \sigma.\sigma'.\sigma.\sigma', \sigma.\sigma'.\sigma.\sigma'.\sigma.\sigma', \dots\}$. Then, with $\Sigma' = \{\sigma'\}$ the delayed language $\text{delay}[\Sigma'](L)$ is the set of all strings composed of the same number of symbols σ and σ' , and whose prefixes contain at least as many symbols σ as σ' . This is a nonregular language. The example is illustrated in Figure 9.

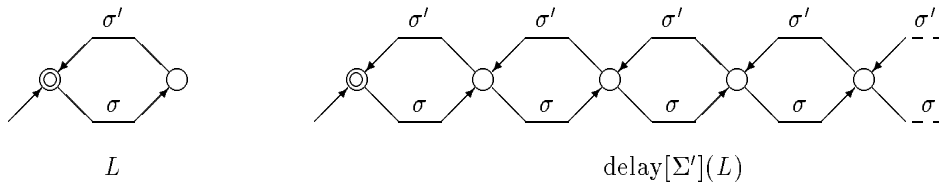


Figure 9. The delay of the regular language L with respect to the set $\Sigma = \{\sigma'\}$ (finite-state automaton on the left) is the nonregular language $\text{delay}[\Sigma'](L)$ (infinite-state automaton on the right).

Some additional properties of the delay operator can be found in the appendix. For an interpretation of the delay of a language, consider a process with alphabet Σ , output set $\Sigma' \subset \Sigma$ and language L . The output of this process is connected to a communication channel as shown in Figure 10.



Figure 10. Process with communication channel at the output.

Imagine now an observer registering the sequence of inputs accepted by the process and of outputs coming from the exit of the output channel (we assume that if an input given to the process cannot be accepted, it is not registered, and no further input will be possible). This sequence is not necessarily a string in L , because the observer registers the outputs with some delay. In fact, the observer could register some inputs while outputs already produced by the process are still in the channel and have not yet reached the exit. The effect is to shift to the right the position of the elements of Σ' in the strings of L , while the relative order of elements of either Σ' or $\Sigma - \Sigma'$ is maintained. The delay of a language expresses exactly the possible distortion of the language L due to the delay that the outputs in Σ' can undergo.

6.2. Closed-Loop System

The operator $\text{delay}[\cdot]$ allows us to characterize the strings that the processes P and C in a closed loop subject to communication delays have processed at a given time. A string s_P processed by the plant P and a string s_C processed by the controller C must satisfy the following condition:

$$\begin{aligned} s_P &\in L_P \cap \overline{\text{delay}[\Sigma_C](s_C \cdot \Sigma_P^*)}, \\ s_C &\in L_C \cap \overline{\text{delay}[\Sigma_P](s_P \cdot \Sigma_C^*)} \end{aligned} \quad (10)$$

(we assume again that if a message is refused by a process, it is not registered and no further input will be accepted). The terms Σ_P^* and Σ_C^* take into account the messages that are present at that particular instant in the communication channels. The string s_P must be contained in the language L_P and, at the same time, be a prefix of a string in $\text{delay}[\Sigma_C](s_C \cdot \Sigma_P^*)$. The set of all such prefixes contains all strings which can be registered at the plant with the knowledge of s_C having been processed by the controller C .

6.2.1. Closed-Loop Languages. Let us consider an arbitrary collection of such pairs of strings, each denoted by (s_P^i, s_C^i) for some index i . For this collection, we denote by L_P^x the set of all strings s_P^i and by L_C^x the set of all strings s_C^i . Then the pair of languages (L_P^x, L_C^x) is a reduced description of the considered collection, because we lose information about the actual pairs. Consider now the union of two such collections of pairs (s_P^i, s_C^i) . The reduced description of the union of the two collections can be determined using the two reduced descriptions of the collections

$(L_{P_1}^x, L_{C_1}^x)$ respectively $(L_{P_2}^x, L_{C_2}^x)$ with

$$(L_{P_1}^x, L_{C_1}^x) \cup (L_{P_2}^x, L_{C_2}^x) = ((L_{P_1}^x \cup L_{P_2}^x), (L_{C_1}^x \cup L_{C_2}^x)).$$

This fact allows us to define the local closed-loop languages of a delayed composition.

DEFINITION 6.2 (Closed-Loop Languages with Communication Delays). The *closed-loop languages* L_P^c and L_C^c of a delayed composition of two processes P and C are the set of all strings s_P respectively s_C that are elements of a pair satisfying condition (10).

It can be seen from (10) that the languages L_P^c and L_C^c satisfy the equations

$$L_C^c = L_P \cap \overline{\text{delay}[\Sigma_C](L_C^c, \Sigma_P^*)}, \quad (11)$$

$$L_P^c = L_C \cap \overline{\text{delay}[\Sigma_P](L_P^c, \Sigma_C^*)}. \quad (12)$$

Moreover, they are the individually largest sublanguages of L_P respectively L_C satisfying simultaneously (11) and (12).

With the definition above, the closed-loop languages (L_P^c, L_C^c) are given as the result of an implicit expression. We are however interested in determining, if possible, an explicit expression for the pair (L_P^c, L_C^c) . This is given by the next theorem.

THEOREM 6.1. The closed-loop languages resulting from the composition of two processes P and C subject to communication delays are given by the explicit expressions

$$L_P^c = L_P \cap \overline{\text{delay}[\Sigma_C](L_C, \Sigma_P^*)}, \quad (13)$$

$$L_C^c = L_C \cap \overline{\text{delay}[\Sigma_P](L_P, \Sigma_C^*)}. \quad (14)$$

The proof of this theorem is given in the appendix. Finally, we give an example of closed-loop languages.

EXAMPLE. Consider the plant language L_P of Figure 5 and the controller language L_C of Figure 6. The closed-loop connection of a plant and a controller with these languages yields the closed-loop languages $L_P^c = L_C^c = L_C$.

6.3. Well-Posed Connection with Delays

Now, we want to generalize the concept of well-posedness to include process connections with communication delays. A connection of processes P and C with communication delays is well-posed if and only if the connection of the corresponding processes P' and C' shown in Figure 11 is well-posed. This makes sure that any message coming from a channel exit can be accepted by the destination process.

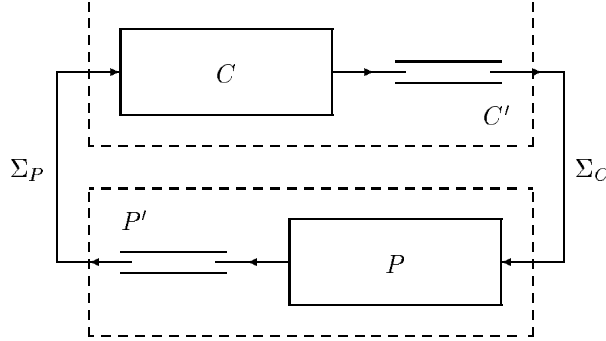


Figure 11. Equivalence of well-posedness of process connections with and without communication delays.

We give here a formal definition of well-posedness of a connection subject to communication delays.

DEFINITION 6.3 (Well-Posed Composition with Delays). The composition of two processes P and C subject to communication delays is *well-posed* if their *open-loop* languages L_P and L_C satisfy

$$L_P \supseteq L_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}, \quad (15)$$

$$L_C \supseteq L_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}. \quad (16)$$

Intuitively, inclusion (15) requires that any command $\sigma_C \in \Sigma_C$ coming from the controller to the plant must be accepted by the plant; the continuation of the current string in the plant with σ_C must form a new string contained again in the language L_P . The expression (15) is also equivalent to the condition that for any pair (s_P, s_C) satisfying (10) the following inclusion holds:

$$L_P \supseteq s_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](s_C.\Sigma_P^*)}.$$

The same interpretation can be given for (16).

EXAMPLE. Consider the plant language L_P of Figure 5 and the controller language L_C of Figure 6. The closed-loop connection of a plant and a controller with these languages is not well-posed. In fact, $a.b.c.o \in L_P$, $a.b.o \in L_C$, and $c \in \Sigma_P$, thus $a.b.o.c \in L_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$ but this violates (16), since $a.b.o.c \notin L_C$. This means that the controller having executed the string $a.b.o$ could suddenly receive from the plant output channel the additional response c , but would not be able to accept it.

We remark that the condition for well-posedness given by (15) and (16) is very similar to the condition for well-posedness of a composition free from delays. In

fact, conditions (5) and (8) can be rephrased as

$$L_P \supseteq L_P.\Sigma_C^* \cap L_C, \quad L_C \supseteq L_C.\Sigma_P^* \cap L_P$$

(here we used the equivalence $L_1 \supseteq L_1.\Sigma' \cap L_2 \Leftrightarrow L_1 \supseteq L_1.(\Sigma')^* \cap L_2$, where L_1 and L_2 are any prefix-closed language in Σ^* and Σ' any subset of Σ ; see Brandt et al. [1990] for the proof).

As we have noted in presenting condition (10) at the beginning of this chapter, the terms Σ_C^* and Σ_P^* in those two equations stand for the messages that are currently in the communication channels. The condition for well-posedness, on the other hand, makes sure that these messages can be accepted by the respective destination processes. Therefore, it is to be expected that the terms Σ_C^* and Σ_P^* in the description of the closed-loop languages given by equations (13) and (14) are superfluous if the composition is well-posed. This conjecture is in fact true as stated by the following corollary to Theorem 6.1.

COROLLARY 6.1. The closed-loop languages resulting from the well-posed composition of two processes P and C subject to communication delays are given by

$$L_P^c = \overline{L_P \cap \text{delay}[\Sigma_C](L_C)}, \quad (17)$$

$$L_C^c = \overline{L_C \cap \text{delay}[\Sigma_P](L_P)}. \quad (18)$$

Proof. We have to prove two inclusions. From (13) we have directly

$$\begin{aligned} L_P^c &= L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)} \\ &\supseteq L_P \cap \overline{\text{delay}[\Sigma_C](L_C)} \\ &\supseteq \overline{L_P \cap \text{delay}[\Sigma_C](L_C)}. \end{aligned}$$

For the other inclusion, take an arbitrary string $s \in L_P^c$. Equation (13) implies

$$s \in L_P, \quad \exists t \in L_C \mid s \in \overline{\text{delay}[\Sigma_C](\{t.\Sigma_P^*\})}.$$

Note that $\mathcal{P}[\Sigma_C](s)$ is a prefix of $\mathcal{P}[\Sigma_C](t)$. Choose now the shortest prefix t' of t such that $\mathcal{P}[\Sigma_C](s) = \mathcal{P}[\Sigma_C](t')$. Then

$$s \in \text{delay}[\Sigma_C](\{t'.t_P\}) \quad \text{for some } t_P \in \Sigma_P^*,$$

from which it follows that $t'.t_P \in \text{delay}[\Sigma_P](\{s\})$. Using (16), the steps

$$t'.t_P \in \{t'.t_P\} \cap \overline{\text{delay}[\Sigma_P](\{s\})} \subseteq L_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} \subseteq L_C,$$

prove $t'.t_P \in L_C$. Finally, inserting s and $t'.t_P$ into the right-hand side of (17) we obtain

$$\begin{aligned} \overline{s \cap \text{delay}[\Sigma_P](\{t'.t_P\})} &\subseteq \overline{L_P \cap \text{delay}[\Sigma_C](L_C)}, \\ s \in \overline{L_P \cap \text{delay}[\Sigma_C](L_C)}, \end{aligned}$$

showing that $L_P^c \subseteq \overline{L_P \cap \text{delay}[\Sigma_C](L_C)}$ and thus proving (17). The same proof holds for (18). Q.E.D.

In the sequel we will only consider controllers such that the connection with the given plant is well-posed. Then, we are interested in knowing if the union of the languages of two such controllers is the language of a controller that has a well-posed connection with the given plant. This property for controllers would be very useful, as it allows us to define an optimal, i.e., a largest controller. This desired property holds as shown with the next proposition.

PROPOSITION 6.1. The class of languages L_C of controllers $C = (\Sigma, \Sigma_C, L_C, L_C)$ enforcing well-posedness of the connection with a given plant $P = (\Sigma, \Sigma_P, L_P, L_P)$ is closed under union of languages.

Proof. Consider two controllers C_1 and C_2 with language L_{C_1} and L_{C_2} and such that their connection with P is well-posed. Then, we have for both C_1 and C_2 a pair of equations of the form of (15) and (16). From this it can be shown with (A2) that inclusions (15) and (16) hold also for a controller with language $L_{C_1} \cup L_{C_2}$. Q.E.D.

The proposition implies that the class $\mathcal{W}[L_P, \Sigma_C](L)$ of sublanguages of L for controllers enforcing well-posedness of the connection with a plant having language L_P is also closed under union. Therefore, there exists a supremal element of $\mathcal{W}[L_P, \Sigma_C](L)$, which we denote by $\sup \mathcal{W}[L_P, \Sigma_C](L)$.

6.4. The Unmarked Controller Synthesis Problem

Now we formulate and solve a controller synthesis problem with delays for plant and specification described by prefix-closed languages. We first have to address the problem of enforcing a desired behavior on the plant with a controller. In particular, we are interested in knowing if the class of languages of controllers enforcing $L_P^c \subseteq L_{\text{spec}} \subseteq \Sigma^*$ is closed under union. Then, there exists a largest controller enforcing the constraint and having well-posed composition with the given plant. The next proposition proves that this is the case.

PROPOSITION 6.2. Given a plant $P = (\Sigma, \Sigma_P, L_P, L_P)$ and a specification language $L_{\text{spec}} \subseteq \Sigma^*$. The class of languages L_C of controllers $C = (\Sigma, \Sigma_C, L_C, L_C)$ such that the closed-loop language of equation (13) satisfies $L_P^c \subseteq L_{\text{spec}}$ is closed under union.

Proof. Consider two controllers C_1 and C_2 enforcing the specification L_{spec} . Then

$$L_{\text{spec}} \supseteq \overline{L_P \cap \text{delay}[\Sigma_P](L_{C_1})}, \quad L_{\text{spec}} \supseteq \overline{L_P \cap \text{delay}[\Sigma_P](L_{C_2})}.$$

Taking the union of the two expressions and using equality (A2), we obtain the inclusion $L_{\text{spec}} \supseteq \overline{L_P \cap \text{delay}[\Sigma_P](L_{C_1} \cup L_{C_2})}$, proving the desired result. Q.E.D.

We are now ready to state the unmarked controller synthesis problem with delays.

UNMARKED CONTROLLER SYNTHESIS PROBLEM WITH DELAYS. *Given a plant $P = (\Sigma, \Sigma_P, L_P, L_P)$ and a prefix-closed specification language $\overline{L'_{\text{spec}}} = L'_{\text{spec}} \subseteq \Sigma_P^*$ for the plant behavior in a closed loop subject to communication delays, find a controller $C = (\Sigma, \Sigma_C, L_C, L_C)$ such that*

1. $L_C \subseteq L_P$.
2. $\emptyset \subset \mathcal{P}[\Sigma_P](L_P^c) \subseteq L'_{\text{spec}}$.
3. *The connection of C and P is well-posed.*

Note that here we do not have any condition for nonblockingness as the languages of both plant and controller are prefix-closed. Also note that we have the new constraint $L_C \subseteq L_P$. This assumption is justified by the wish to have a best possible image of the behavior of the plant with the controller.

From the two previous propositions we can state the conditions for existence of a solution to this problem.

THEOREM 6.2. The unmarked controller synthesis problem with delays has a solution if and only if for the language

$$L_C = \sup\{K : K \in \mathcal{W}[L_P, \Sigma_C](L_P) \wedge L_P^c \subseteq \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}})\},$$

where $L_P^c = \overline{L_P \cap \text{delay}[\Sigma_C](K)}$, we have $\emptyset \subset \mathcal{P}[\Sigma_P](L_C)$. If a solution exists, the controller with language L_C is a solution.

Proof. Direct from Propositions 6.1 and 6.2 and from the remark that the intersection of two classes of languages which are closed under union of languages is also closed under union of languages. Q.E.D.

As already explained in the previous section, we are actually interested in local specifications, i.e., in specifications on the subalphabet Σ_P only. While it is difficult to give direct conditions on the language of a controller that enforces a global specification, this is very easy if the specification is local, as stated in the next proposition.

PROPOSITION 6.3. Consider a prefix-closed language $L_{\text{spec}} \subseteq \Sigma_P^*$, a plant $P = (\Sigma, \Sigma_P, L_P, L_P)$ and a controller $C = (\Sigma, \Sigma_C, L_C, L_C)$ such that the connection of P and C is well-posed. If $\mathcal{P}[\Sigma_P](L_C) \subseteq L_{\text{spec}}$ then $\mathcal{P}[\Sigma_P](L_P^c) \subseteq L_{\text{spec}}$.

Suppose, in addition, that $L_C \subseteq L_P$, then $\mathcal{P}[\Sigma_P](L_C) = \mathcal{P}[\Sigma_P](L_P^c)$.

Proof. We start with equation (17):

$$\begin{aligned} L_P^c &= \overline{L_P \cap \text{delay}[\Sigma_C](L_C)} \subseteq \overline{\text{delay}[\Sigma_C](L_C)}, \\ \mathcal{P}[\Sigma_P](L_P^c) &\subseteq \mathcal{P}[\Sigma_P](\overline{\text{delay}[\Sigma_C](L_C)}) = \overline{\mathcal{P}[\Sigma_P](\text{delay}[\Sigma_C](L_C))}, \\ &= \mathcal{P}[\Sigma_P](L_C) \subseteq L_{\text{spec}} \end{aligned}$$

finally proving the desired inclusion.

The equality of the projections of L_C and L_P^c is proved by remarking that $L_C \subseteq L_P$ implies $L_C = L_C^c \subseteq L_P^c$ (this is proved just by inspection of (17) and (18)).
Q.E.D.

Let us suppose that we have already a controller $C = (\Sigma, \Sigma_C, L_C, L_C)$ and that its connection with the plant P is well-posed, but that a given local specification L_{spec} on the closed-loop language of the plant L_P^c is not satisfied. From the previous proposition we know that a controller C' , further restricting the possible commands in Σ_C sent to P until $\mathcal{P}[\Sigma_P](L'_C) \subseteq L_{\text{spec}}$ holds, guarantees the enforcement of the specification L_{spec} on the plant. The language L'_C is by choice, as we restrict only the commands, $[L_C, \Sigma_C]$ -controllable. Then we hope that this new controller C' has also a well-posed composition with P . This holds as shown in the next proposition.

PROPOSITION 6.4. Consider $P = (\Sigma, \Sigma_P, L_P, L_P)$ and $C = (\Sigma, \Sigma_C, L_C, L_C)$ such that their composition is well-posed. The composition of P with a controller $C' = (\Sigma, \Sigma_C, L'_C, L'_C)$ with $L'_C \subseteq L_C \subseteq L_P$ is well-posed if and only if the language L'_C is $[L_C, \Sigma_C]$ -controllable.

Proof. (IF) In order to prove that a controller with language L'_C which is $[L_C, \Sigma_C]$ -controllable and the plant P have a well-posed connection, we need to show

$$\begin{aligned} L_P &\supseteq L_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L'_C.\Sigma_P^*)}, \\ L'_C &\supseteq L'_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}. \end{aligned}$$

The first inclusion comes from (15) (the connection of P and C is well-posed by assumption) and $L'_C \subseteq L_C$. For the other inclusion we use (16) and conclude with

$$\begin{aligned} L_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} &\subseteq L_C, \\ L'_C.\Sigma_P^* \cap L_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} &\subseteq L'_C.\Sigma_P^* \cap L_C, \\ L'_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} &\subseteq L'_C. \end{aligned}$$

Here, in the last step we used step the inclusion $L'_C.\Sigma_P^* \cap L_C \subseteq L'_C$ which comes from the $[L_C, \Sigma_C]$ -controllability of L'_C .

(ONLY IF) We prove this direction by contradiction. Let us suppose that L'_C is not $[L_C, \Sigma_C]$ -controllable, but that C' and P have a well-posed connection. Then, there exists a string $s_C \in L'_C \subseteq L_C$ and $\sigma_P \in \Sigma_P$, with $s_C.\sigma_P \in L_C$ but $s_C.\sigma_P \notin L'_C$. Then, from $s_C.\sigma_P \in L_P$ (as $L_C \subseteq L_P$) we have $s_C.\sigma_P \in \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$ and $s_C.\sigma_P \in L'_C.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$, but $s_C.\sigma_P \notin L'_C$, showing that inclusion (16) does not hold for L'_C , and that the composition of C' and not P is not well-posed.
Q.E.D.

This proposition allows us to restate the condition for existence of a solution to the unmarked controller synthesis problem with delays.

THEOREM 6.3. The unmarked controller synthesis problem with delays has a solution if and only if for the language

$$L_C = \sup \mathcal{C}(\sup \mathcal{W}[L_P, \Sigma_C](L_P) \cap \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}})).$$

we have $\emptyset \subset \mathcal{P}[\Sigma_P](L_C)$. If a solution exists, the controller with language L_C is a solution.

Proof. From its definition we know that $\sup \mathcal{W}[L_P, \Sigma_C](L_P)$ is the largest sublanguage of L_P for a controller with well-posed connection with the given plant. Proposition 6.3 ensures sure that the specification is enforced, while Proposition 6.4 guarantees well-posedness of the connection. Q.E.D.

EXAMPLE. Consider the plant language L_P of Figure 5. The supremal sublanguage of L_P for a controller guaranteeing well-posedness of the composition with the plant with language L_P is given by the left automaton of Figure 12.

Consider now the language M'_{spec} from Figure 5. The least restrictive controller solving the unmarked controller synthesis problem with delays for the plant with language L_P and the specification $L'_{\text{spec}} = \overline{M'_{\text{spec}}}$ is given by the right automaton of Figure 12. Note that $L_P^c \supset L_C^c = \overline{L_C}$, since $a.b.o \in L_P^c$ but $a.b.o \notin L_C^c$.

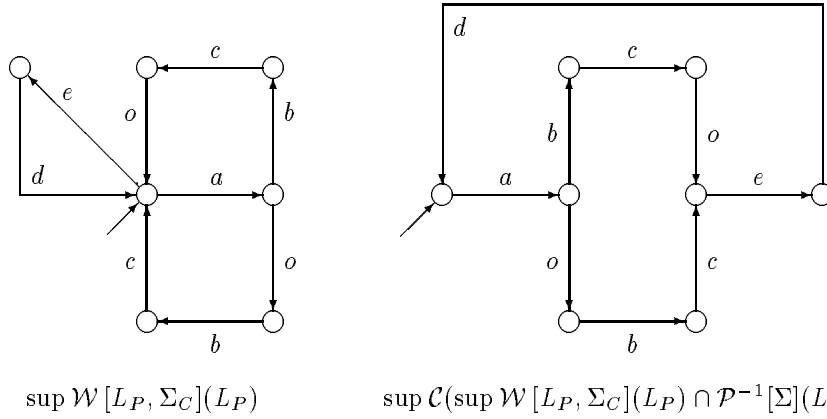


Figure 12. (Left) Automaton for the supremal sublanguage of L_P for a controller guaranteeing well-posedness of the composition with the plant with language L_P . (Right) Automaton for the least restrictive controller solving the unmarked controller synthesis problem with delays for the plant with language L_P and specification $L'_{\text{spec}} = \overline{M'_{\text{spec}}}$ (see Figure 5 for L_P and M'_{spec}).

We note, however, that the computation of $\sup \mathcal{W}[L_P, \Sigma_C](L_P)$ is not easy, as the conditions for well-posedness are expressed in terms of nonregular languages even for a plant with regular language (see (15) and (16) and the example in Figure 9). We are therefore interested in special classes of plants, for which the computation of a controller can be performed using known algorithms. For this we introduce the class of self-well-posed languages.

6.5. Self-Well-Posed Languages

We now investigate the important class of plants $P = (\Sigma, \Sigma_P, L_P, L_P)$ whose connection with a controller $(\Sigma, \Sigma - \Sigma_P, L_P, L_P)$ is well-posed. We can see this in Figure 13 (note the slight abuse of notation: both the controller and the plant are denoted by the symbol P , although the output sets are complementary). The languages of plants in this class are called *self-well-posed*.

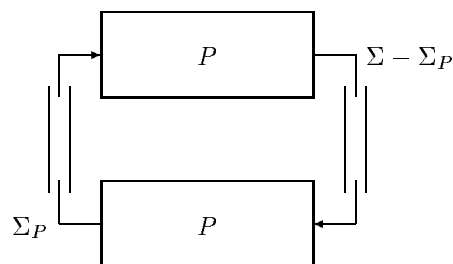


Figure 13. The language of a plant yielding a well-posed composition with a controller having the plant's language is self-well-posed.

DEFINITION 6.4 (Self-Well-Posed Language). A prefix-closed language L is *self-well-posed* with respect to the partition $\Sigma = \Sigma_P \dot{\cup} \Sigma_C$ if the following inclusions hold:

$$L \supseteq L.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L.\Sigma_P^*)}, \quad (19)$$

$$L \supseteq L.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L.\Sigma_C^*)}. \quad (20)$$

Suppose the plant $P = (\Sigma, \Sigma_P, L_P, L_P)$ has a self-well-posed language L_P . Then we have the following corollary to Theorem 6.3.

COROLLARY 6.2. The unmarked controller synthesis problem with delays for a plant with self-well-posed language L_P has a solution if and only if for the language

$$L_C = \sup \mathcal{C}(L_P \cap \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}}))$$

we have $\emptyset \subset \mathcal{P}[\Sigma_P](L_C)$. If a solution exists, the controller with language L_C is a solution.

Proof. The corollary is an immediate consequence of Propositions 6.3 and 6.4. Q.E.D.

The condition for the existence of a solution to the unmarked controller synthesis problem with delays is therefore exactly the same as the condition for the existence of a solution to the controller synthesis problem (where all languages are prefix-closed) stated in Theorem 4.1.

We note that the class of self-well-posed languages is not closed under union of languages. This is shown by the next example

EXAMPLE. Consider $\sigma \in \Sigma - \Sigma_P$, $\sigma' \in \Sigma_P$, $L_1 = \{\sigma\}$ and $L_2 = \{\sigma'\}$. Both L_1 and L_2 are self-well-posed, but $L_1 \cup L_2$ is not, as the conditions for self-well-posedness of this language require $\sigma.\sigma' \in L_1 \cup L_2$.

Closedness under union, however, is true for the particular class of languages contained in a language L and also either $[L, \Sigma_P]$ - or $[L, \Sigma_C]$ -controllable.

PROPOSITION 6.5. The class $\mathcal{SC}[L_P, \Sigma_C](L)$ of self-well-posed and $[L_P, \Sigma_C]$ -controllable sublanguages of $L \subseteq L_P$ is closed under language union. There exists a supremal element of this class, denoted by $\sup \mathcal{SC}[L_P, \Sigma_C](L)$

The proof can be found in the appendix.

Assume now that the language L_P of the plant is not self-well-posed. While Theorems 6.2 and 6.3 provided a necessary and sufficient condition for the existence of a solution to the unmarked controller synthesis problem with delays, the above proposition suggests an alternative sufficient characterization of the solution. In particular, we can proceed as follows:

1. Compute the language $L'_P = \sup \mathcal{SC}[L_P, \Sigma_C](L_P)$.
2. Consider this as the language of a new fictitious plant and compute the controller for this plant after Corollary 6.2.
3. Use this controller together with the original plant.

Because L'_P is $[L_P, \Sigma_C]$ -controllable (by construction), it is possible to show that the behavior of the well-posed closed-loop formed by the controller and the fictitious plant does not change if we replace this plant by the original one (see Lemma A.1 in the appendix). Thus, if we find a solution to the unmarked controller synthesis problem for the fictitious plant we automatically have a solution to the problem for the original plant.

Another possibility is the direct computation of the controller for the original plant using the newly defined class of self-well-posed and controllable languages. We state the two solution methods in the next theorem.

THEOREM 6.4. The unmarked controller synthesis problem with delays has a solution if for the language

$$\begin{aligned} L_C &= \sup \mathcal{C}(\sup \mathcal{SC}[L_P, \Sigma_C](L_P) \cap \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}})) \\ &= \sup \mathcal{SC}[L_P, \Sigma_C](L_P \cap \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}})) \end{aligned}$$

we have $\emptyset \subset \mathcal{P}[\Sigma_P](L_C)$. If a solution is proven to exist, the controller with language L_C is a solution.

Proof. The proof is direct from Proposition 6.4 and from the definition of the class of self-well-posed and controllable languages. Q.E.D.

Remark that Theorem 6.4 (as opposed to Theorem 6.3) only gives a sufficient condition for the existence of a solution. This is due to the inclusion

$$\sup \mathcal{SC}[L_P, \Sigma_C](L_P) \subseteq \sup \mathcal{W}[L_P, \Sigma_C](L_P).$$

In fact a controller with language $\sup \mathcal{SC}[L_P, \Sigma_C](L_P)$ (and any other controller with a $[L_P, \Sigma_C]$ -controllable sublanguage of $\sup \mathcal{SC}[L_P, \Sigma_C](L_P)$) not only ensures well-posedness of the connection with the plant but also forces a certain symmetry in the behavior of the closed-loop (the closed-loop languages of controller and plant are identical, although the two processes may still execute different strings). Thus, the controller with language L_C given by Theorem 6.4 is not least restrictive, since the inclusion

$$L_C \subset \sup \mathcal{C}(\sup \mathcal{W}[L_P, \Sigma_C](L_P) \cap \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}}))$$

is strict in general. Moreover, if $\mathcal{P}[\Sigma_P](L_C)$ is empty, a solution to the supervisor synthesis problem may still exist.

EXAMPLE. Consider the plant language L_P of Figure 5. The supremal self-well-posed and $[L_P, \Sigma_C]$ -controllable sublanguage of L_P is given by the left automaton of Figure 14. The inclusion $\sup \mathcal{SC}[L_P, \Sigma_C](L_P) \subset \sup \mathcal{W}[L_P, \Sigma_C](L_P)$ is strict (compare with Figure 12).

Consider now the language M'_{spec} from Figure 5. A controller computed from $\sup \mathcal{SC}[L_P, \Sigma_C](L_P)$ and solving the unmarked controller synthesis problem with delays for the plant with language L_P and the specification $L'_{\text{spec}} = \overline{M'_{\text{spec}}}$ is given by the right automaton of Figure 14. The closed-loop languages of the plant and of the controller are identical, i.e., $L_P^c = L_C^c = L_C$. Note that this controller is not least restrictive (compare with Figure 12).

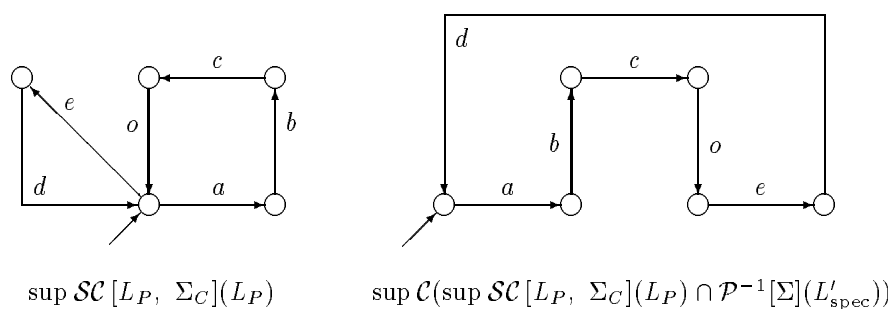


Figure 14. (Right) Automaton for a (suboptimal) controller solving the unmarked controller synthesis problem with delays for the plant with language L_P and the specification $L'_{\text{spec}} = \overline{M'_{\text{spec}}}$ (see Figure 5 for L_P and M'_{spec}).

6.6. Memoryless Languages

We have now characterized a suboptimal solution to the unmarked controller synthesis problem with delays with the help of self-well-posed languages. Unfortunately, the computation of the suboptimal solution to the unmarked controller synthesis problem given by Theorem 6.4 presents the same difficulties as the computation of the optimal solution given by Theorem 6.3. However, we will show that if the language of the plant satisfies a property called memorylessness, the solution given by Theorem 6.4 can be efficiently computed. Moreover, it becomes also optimal. We now define memorylessness.

DEFINITION 6.5 (Memoryless Language). A prefix-closed language L is *memoryless* with respect to the partition $\Sigma = \Sigma_P \dot{\cup} \Sigma_C$ if for every $s, s' \in L$ such that $s \in \text{delay}[\Sigma_P](\{s'\})$ and an arbitrary string $t \in \Sigma^*$

$$s.t \in L \Leftrightarrow s'.t \in L. \quad (21)$$

Remark that the definition is symmetric, i.e., the definition is equivalent to the one using $s \in \text{delay}[\Sigma_C](\{s'\})$ instead of $s \in \text{delay}[\Sigma_P](\{s'\})$.

EXAMPLE. Consider the plant language L_P in Figure 5. This language is not memoryless. In fact, $a.b.o, a.o.b \in L_P$ and $a.o.b.c \in L_P$ but $a.b.o.c \notin L_P$, violating the condition for memorylessness given by (21).

EXAMPLE. Consider the new plant language L_P given in Figure 15. This language satisfies condition (21) and is therefore memoryless.

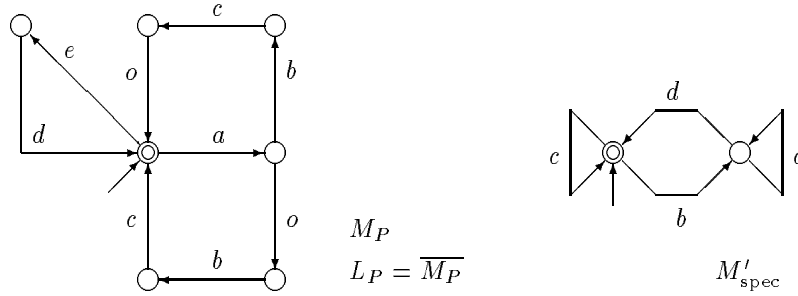


Figure 15. Memoryless plant language L_P and local specification language M'_{spec} . Vowels (the set $\{a, e, o\}$) represent commands while responses are given by consonants (the set $\{b, c, d\}$).

Memorylessness means that the permutation of commands and responses does not affect the future evolution of the closed-loop system. This implies that the sequence of responses produced and the sequence of commands accepted by the plant uniquely determine the future behavior of the closed-loop system and that

we can forget the plant “dynamics” given by the specific interleaving of commands and responses.

The computational advantage of memoryless languages is due to their easier characterization of self-well-posedness. This is stated in the next proposition.

PROPOSITION 6.6. A prefix-closed memoryless language L is self-well-posed if and only if for $s \in L$, $t_P \in \Sigma_P^*$, and $t_C \in \Sigma_C^*$

$$s.t_P \in L, s.t_C \in L \Rightarrow s.\text{delay}[\Sigma_C](\{t_C.t_P\}) = s.\text{delay}[\Sigma_P](\{t_P.t_C\}) \subseteq L. \quad (22)$$

The proof is given in the appendix.

From Propositions 6.5 and 6.6 we know that we can find $\sup \mathcal{SC}(L_P, \Sigma_C)(L_P)$, the supremal self-well-posed and $[L_P, \Sigma_C]$ -controllable sublanguage of a memoryless language L_P , simply by finding the largest sublanguage L'_P of L_P satisfying $L'_P.\Sigma_P \cap L_P \subseteq L'_P$ and (22). Therefore, the computation of $\sup \mathcal{SC}(L_P, \Sigma_C)(L_P)$ can be performed with an algorithm of polynomial complexity for a memoryless regular language L_P .

EXAMPLE. Consider the memoryless language L_P given in Figure 15. The largest sublanguage L'_P of L_P satisfying $L'_P.\Sigma_P \cap L_P \subseteq L'_P$ and condition (22) is given by the left automaton of Figure 16. It is $\sup \mathcal{SC}(L_P, \Sigma_C)(L_P)$, the supremal self-well-posed and $[L_P, \Sigma_C]$ -controllable sublanguage of L_P .

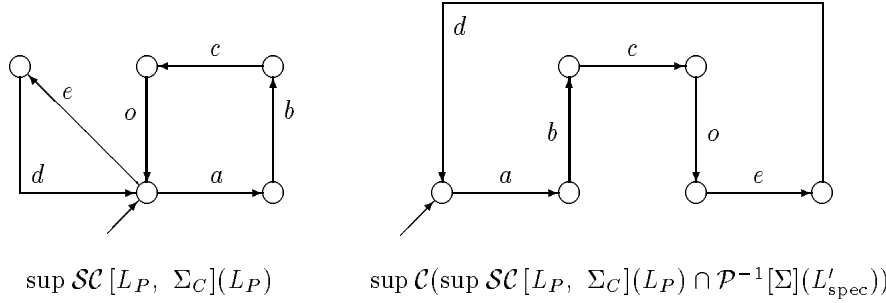


Figure 16. (Left) $\sup \mathcal{SC}(L_P, \Sigma_C)(L_P) = \sup \mathcal{W}(L_P, \Sigma_C)(L_P)$, because of memorylessness of L_P . (Right) Automaton representing the *least restrictive* controller solving the unmarked controller synthesis problem with delays for the plant with language L_P and specification $L'_{\text{spec}} = \overline{M'_{\text{spec}}}$ (see Figure 15 for L_P and M'_{spec}).

A very important property of a memoryless language L_P is the fact that the language $\sup \mathcal{W}(L_P, \Sigma_C)(L_P)$ is also self-well-posed, i.e., $\sup \mathcal{W}(L_P, \Sigma_C)(L_P)$ is also the supremal self-well-posed and $[L_P, \Sigma_C]$ -controllable sublanguage of L_P . We state this in the next theorem.

THEOREM 6.5. Given a memoryless language L_P ,

$$\sup \mathcal{SC}(L_P, \Sigma_C)(L_P) = \sup \mathcal{W}(L_P, \Sigma_C)(L_P).$$

The proof can be found in the appendix.

EXAMPLE. Consider the memoryless language L_P given in Figure 15 and the language $\sup \mathcal{SC}(L_P, \Sigma_C)(L_P)$ given in Figure 16. Then, from the above theorem, $\sup \mathcal{SC}(L_P, \Sigma_C)(L_P) = \sup \mathcal{W}(L_P, \Sigma_C)(L_P)$, i.e., $\sup \mathcal{SC}(L_P, \Sigma_C)(L_P)$ is also the supremal sublanguage of L_P for a controller guaranteeing well-posedness of the composition with a plant with language L_P .

With help of this theorem, we can state the condition for existence of a solution to the unmarked controller synthesis problem for the case when the plant has a memoryless language.

THEOREM 6.6. The unmarked controller synthesis problem with delays for a plant P with memoryless language L_P has a solution if and only if for the language

$$\begin{aligned} L_C &= \sup \mathcal{C}(\sup \mathcal{SC}[L_P, \Sigma_C](L_P) \cap \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}})) \\ &= \sup \mathcal{SC}[L_P, \Sigma_C](L_P \cap \mathcal{P}^{-1}[\Sigma](L'_{\text{spec}})) \end{aligned}$$

we have $\emptyset \subset \mathcal{P}[\Sigma_P](L_C)$. If a solution exists, the controller with language L_C is a solution.

Proof. The proof follows directly from Theorems 6.3 and 6.5.

Q.E.D.

EXAMPLE. Consider the memoryless plant language L_P and the specification language M'_{spec} of Figure 15. The least restrictive controller solving the unmarked controller synthesis problem with delays for the plant with language L_P and the specification $L'_{\text{spec}} = \overline{M'_{\text{spec}}}$ is given by the right automaton of Figure 16. Note that the controller language is computed from $\sup \mathcal{SC}[L_P, \Sigma_C](L_P)$, but in this case it yields the least restrictive controller, because of memorylessness of L_P .

Thus, given a plant with memoryless language we can obtain the least restrictive solution in polynomial time when all languages are regular. In fact, the algorithm for the computation of the controller must make sure that both controllability and condition (22) are satisfied, and this leads to an algorithm of polynomial complexity.

We now consider processes in which the marked languages are not prefix-closed.

7. Communication Delays and Processes with Marked Languages

In Section 3.2 the concept of nonblockingness of a process has been introduced. The same definition was later used to characterize nonblockingness of a closed-loop system free from delays. We now have to generalize this definition to include nonblockingness of a process in a composition with delays; the process considered corresponds to the local view that an observer has when looking at the plant or at the controller only.

A process is nonblocking in a well-posed composition of two processes if any string in the closed-loop language of this process can always be completed to a

string in the marked language. We first note that the future evolution of a closed-loop language does not only depend on the string processed so far by the process considered but also by the string in the other process. Then, the post-language of a closed-loop language must be expressed with the help of the current strings in both processes. We denote this by $L_P^c/(s_P, s_C)$ and $L_C^c/(s_P, s_C)$ for the closed-loop languages of P and C respectively.

EXAMPLE. Consider two processes P and C with respective languages

$$L_P = \overline{\{\sigma_C.\sigma_P.\sigma'_P, \sigma_P.\sigma_C.\sigma''_P\}} \quad \text{and} \quad L_C = \overline{\{\sigma_P, \sigma_C.\sigma_P.\sigma'_P, \sigma_C.\sigma_P.\sigma''_P\}}$$

where $\sigma_C \in \Sigma_C$ and $\sigma_P, \sigma'_P, \sigma''_P \in \Sigma_P$. Then we note that $L_C^c/(\sigma_C.\sigma_P, \sigma_C.\sigma_P) = \{\sigma'_P\}$ while $L_C^c/(\sigma_P.\sigma_C, \sigma_C.\sigma_P) = \{\sigma''_P\}$. This shows that the next event received by process C in fact depends on the string executed by process P .

We are now ready to formally express nonblockingness in the next definition.

DEFINITION 7.1 (Nonblocking Process in Composition with Delay). The composition of two processes $P = (\Sigma, \Sigma_P, L_P, M_P)$ and $C = (\Sigma, \Sigma_C, L_C, M_C)$ subject to communication delays is *nonblocking* for process C if every string in L_C^c can be completed to a string in M_C , i.e., if for all pairs (s_P, s_C) satisfying condition (10)

$$s_C.(L_C^c/(s_P, s_C)) \cap M_C \neq \emptyset$$

In particular, we can state the following proposition.

PROPOSITION 7.1. A necessary condition for nonblockingness for process C in a delayed composition is

$$\overline{L_C^c \cap M_C} = L_C^c. \quad (23)$$

Proof. We first remark that $\overline{L_C^c \cap M_C} \subseteq L_C^c$ always holds. The necessity of condition (23) is easy to show, noting that $\overline{L_C^c \cap M_C} \subset L_C^c$ implies that there exists a string $s \in L_C^c$ which cannot be completed to a string in M_C . Q.E.D.

To show that equation (23) it is not sufficient we present the following example.

EXAMPLE. Consider two nonblocking processes P and C with $M_P = M_C = \{\sigma_C.\sigma_P.\sigma'_P, \sigma_P.\sigma_C.\sigma'_C\}$, $\sigma_C, \sigma'_C \in \Sigma_C$ and $\sigma_P, \sigma'_P \in \Sigma_P$. It is easy to show that condition (23) is satisfied as $L_C^c = L_P^c = \overline{M_C} = \overline{M_P}$. The pair (s_P, s_C) with $s_P = \sigma_P.\sigma_C$ executed by process P and $s_C = \sigma_C.\sigma_P$ executed by process C satisfies condition (10) but $L_C^c/(s_P, s_C) = \emptyset$, in contrast with the definition of nonblockingness.

Notice that the condition stated in the proposition was necessary and sufficient for nonblockingness for the process C in a closed-loop free from delays. In order to see this, consider the delay operator to have no effect. Then equation (23) becomes $L_C \cap L_P = \overline{M_C} \cap L_P$. For a plant P with prefix-closed marked language $M_P = \overline{M_P} = L_P$, the equation is also equivalent to $L_{P\parallel C} = \overline{M_{P\parallel C}}$, which is

the condition guaranteeing that a marking (in C , as P has prefix-closed marked language) can always be reached from any string in the closed-loop language.

We are now interested in constraints so that condition (23) is also sufficient for the process C to be nonblocking in a delayed composition.

PROPOSITION 7.2. Consider the well-posed composition of $C = (\Sigma, \overline{M_C}, M_C)$ and $P = (\Sigma, \overline{M_P}, M_P)$, with $M_C \subseteq M_P$ where $\overline{M_P}$ is a self-well-posed and memoryless language. The closed-loop is nonblocking for process C if and only if equation (23) is satisfied.

The proof is given in the appendix.

We note that also this proposition assumes that $L_C \subseteq L_P$. By memorylessness of L_P and with this choice, a string s_C in the controller with empty channels can be assumed without loss of generality to have been also processed by the plant.

We now present the marked controller synthesis problem with delays.

CONTROLLER SYNTHESIS PROBLEM WITH DELAYS. Given a plant $P = (\Sigma, \Sigma_P, \overline{M_P}, M_P)$ and a specification language $M'_{\text{spec}} \subseteq \Sigma_P^*$ for the closed-loop behavior, find a controller $C = (\Sigma, \Sigma_C, \overline{M_C}, M_C)$ with $M_C \subseteq M_P$ such that

1. $\emptyset \subset \mathcal{P}[\Sigma_P](L_P^c) \subseteq \overline{M'_{\text{spec}}}$.
2. The connection with delays of P and C is well-posed.
3. The closed loop is nonblocking for controller C .
4. A marking in the controller will eventually be a marking in the plant and specification.

For the characterization of the solution we need the extension of the concept of memorylessness to marked languages.

DEFINITION 7.2 (Memoryless Marked Language). A marked language L is *memoryless* with respect to the partition $\Sigma = \Sigma_P \cup \Sigma_C$ if \overline{L} is memoryless, and if for every $s, s' \in \overline{L}$ with $s \in \text{delay}[\Sigma_P](\{s'\})$

$$s.t \in L \Leftrightarrow s'.t \in L. \quad (24)$$

Suppose that the language M_P of the controller is memoryless, and moreover that $s \in M_P$ implies $s.\Sigma_P \cap \overline{M_P} = \emptyset$ (this condition states that if a marking is reached, no additional response can occur without a next command). For this particular case, we can state the formal condition for existence of a solution in the next theorem.

THEOREM 7.1. Consider a plant with memoryless language M_P satisfying

$$s \in M_P \Rightarrow s.\Sigma_P \cap \overline{M_P} = \emptyset. \quad (25)$$

The controller synthesis problem has a solution if and only if for the language

$$M_C = \sup \mathcal{C}(\sup \mathcal{SC}[L_P, \Sigma_C](L_P) \cap M_P \cap \mathcal{P}^{-1}[\Sigma](M'_{\text{spec}}))$$

$\mathcal{P}[\Sigma_P](M_C)$ is nonempty. $C = (\Sigma, \Sigma_C, \overline{M_C}, M_C)$ is the least restrictive solution.

Proof. Proposition 6.3 makes sure that requirement 1 is satisfied. Requirement 2 is enforced because of Proposition 6.4. Memorylessness of M_P ensures that the closed loop is nonblocking for C (see requirement 3). Memorylessness of M_P and condition (25) enforce requirement 4. Q.E.D.

It is therefore possible, given the assumptions of memorylessness and the additional assumption (25) on the languages of the plant, to compute with a polynomial algorithm the solution to the controller synthesis problem with delays.

EXAMPLE. Consider the plant language M_P of Figure 15. M_P is memoryless since $L_P = \overline{M_P}$ is memoryless and condition (24) is satisfied. Moreover, also condition (25) is fulfilled. Using Theorem 7.1 we can compute the least restrictive solution to the controller synthesis problem with delays for the plant with language M_P and the specification M'_{spec} (see Figure 15). It is given by the language of the right automaton of Figure 16 with the initial state only being marked.

8. Design Issues

In the previous section we have characterized solutions to the controller synthesis problem with communication delays. In order to be able to easily compute the solutions, we assumed several properties for plant languages.

It is the opinion of the author that many if not most technical systems can be designed so as to satisfy these properties. In fact, plant design is a crucial point in the control of a discrete event system, and should not be, if possible, considered separately from the controller design problem. The plant should be designed in such a way that the controller design, with possibly often changing logical specifications, can be accomplished with little effort. For instance, in order to satisfy memorylessness, we *model* the plant to accept commands only if the different interleavings with pending responses lead to the same future behavior.

Let us consider an even stronger assumption on the plant language than those made before: the plant language $L_P = \overline{M_P}$ satisfies condition (25) and the condition

$$s.\sigma, s.\sigma' \in L_P \Rightarrow \begin{cases} \sigma, \sigma' \in \Sigma_C \\ \text{or} \\ \sigma, \sigma' \in \Sigma_P \end{cases} \quad (26)$$

for any $s \in L_P$ and $\sigma, \sigma' \in \Sigma$. This condition means that in the automaton description of the plant language the transitions leaving a given state are either all commands in Σ_P or all responses in Σ_C ; if well-posedness is to be enforced, then a command can be sent only if all responses that could reach the controller from the plant have already been registered. In particular, such a language L_P cannot contain two strings $s.\sigma_C$ and $s.\sigma_P$ and this implies that the definition of

memorylessness and the condition of self-well-posedness given by (22) are trivially satisfied.

In the design of the plant of the RTM [Balemi 1992b, Balemi et al. 1993], these issues have been taken into consideration. In fact, the independent modular subsystem have been modeled using languages satisfying the constraints above. This can be shown to be sufficient for guaranteeing self-well-posedness and condition (25) for the global plant.

Thus, by taking care of some issues at plant design time (making sure that the languages of the modular subsystems satisfy some properties in order to counteract the effect of communication delays) and at control software implementation time (making sure that a supervisor solving the supervisor synthesis problem is composed *on-line* with a copy of the plant process to yield a solution to the controller synthesis problem), the results obtained in the field of supervisory control could be directly applied to the synthesis of discrete event controllers for the RTM application.

9. Conclusions

In this paper, an input/output perspective on supervisory control has been presented. The plant generates responses in reaction to commands and, symmetrically, the controller accepts the responses as inputs and produces commands for the plant. The notion of well-posedness of a connection has been presented.

The effect of communication delays on the connection of a plant and a controller has been addressed. The languages of controllers that guarantee well-posedness of a connection subject to delays have been characterized and conditions for nonblockingness extended to connections with communication delays have been presented. The restriction to plants with memoryless languages allows one to compute in polynomial time a controller solving the controller synthesis problem with communication delays.

The author believes that plant modeling should not be considered independently from control issues but integrated from the beginning in the design of the control system. The experience with the design of the control software of a semiconductor manufacturing device suggests that, although not all discrete event systems satisfy the above restrictions, most technical systems can be designed so to enforce them.

Future research should focus on evaluating the limitations of systems designed in such a way and on looking for alternative classes of languages relaxing these restrictions but still allowing a feasible computation of controllers.

Appendix

A.1. Some Properties of the Delay Operator

We present here some properties of the operator $\text{delay}[\cdot]$. The operator $\text{delay}[\Sigma']$ switches the order of an element of Σ' preceding an element of $\Sigma - \Sigma'$. It is evident

that

$$\text{delay}[\Sigma'](L) \neq \text{delay}[\Sigma - \Sigma'](L).$$

However, the equality

$$\text{delay}[\Sigma - \Sigma'](\text{delay}[\Sigma'](L)) = \text{delay}[\Sigma](\text{delay}[\Sigma - \Sigma'](L))$$

holds. It describes the language containing all strings obtained from strings in L by repeatedly permuting adjacent elements not in the same set Σ' or $\Sigma - \Sigma'$. We remark that $\text{delay}[\cdot]$ applied to a prefix-closed language does not yield in general a prefix-closed language, i.e., the inclusion

$$\text{delay}[\Sigma'](\overline{L}) \subseteq \overline{\text{delay}[\Sigma'](\overline{L})}$$

is strict in general. To show this, take $L = \{\sigma'.\sigma\}$, with $\sigma' \in \Sigma'$ and $\sigma \in \Sigma - \Sigma'$. Then $\sigma \in \text{delay}[\Sigma'](\overline{L})$ but $\sigma \notin \overline{\text{delay}[\Sigma'](\overline{L})}$. However, the following property of the $\text{delay}[\cdot]$ operator holds.

$$\overline{\text{delay}[\Sigma'](L)} = \overline{\text{delay}[\Sigma'](\overline{L})}. \quad (\text{A1})$$

Proof. Take $t \in \overline{\text{delay}[\Sigma'](L)}$. Then

$$\exists s \in \overline{L} \mid t \in \overline{\text{delay}[\Sigma'](\{s\})}.$$

For such a string s

$$\exists t' \in \Sigma^* \text{ with } t.t' \in \text{delay}[\Sigma'](\{s\}).$$

Also, there exists a string $s' \in \Sigma^*$ such that $s.s' \in L$. This implies

$$t.t'.s' \in \text{delay}[\Sigma'](\{s.s'\}),$$

which yields $t \in \overline{\text{delay}[\Sigma'](L)}$, proving equation (A1). Q.E.D.

We now show some relations between two languages. Take $L_1, L_2 \subseteq \Sigma^*$ and any $\Sigma' \subseteq \Sigma$. Then

$$\text{delay}[\Sigma'](L_1 \cup L_2) = \text{delay}[\Sigma'](L_1) \cup \text{delay}[\Sigma'](L_2). \quad (\text{A2})$$

Also

$$\text{delay}[\Sigma'](L_1 \cap L_2) \subseteq \text{delay}[\Sigma'](L_1) \cap \text{delay}[\Sigma'](L_2). \quad (\text{A3})$$

Proof. The equality comes direct from the definition of the operator $\text{delay}[\cdot]$. The inclusion can be derived with the following steps.

$$\begin{aligned} \text{delay}[\Sigma'](L_1) &= \text{delay}[\Sigma']((L_1 \cap L_2) \cup (L_1 - L_2)) \\ &= \text{delay}[\Sigma'](L_1 \cap L_2) \cup \text{delay}[\Sigma'](L_1 - L_2) \\ &\supseteq \text{delay}[\Sigma'](L_1 \cap L_2). \end{aligned}$$

Similarly, we have $\text{delay}[\Sigma'](L_2) \supseteq \text{delay}[\Sigma'](L_1 \cap L_2)$. Taking the intersection of the two expressions yields the desired result. Q.E.D.

We remark that the inclusion is strict in general. Take for instance $\sigma' \in \Sigma'$, $\sigma \in \Sigma - \Sigma'$, $L_1 = \{\sigma.\sigma'\}$, and $L_2 = \{\sigma'.\sigma\}$. Then we have $\text{delay}[\Sigma'](L_1 \cap L_2) = \emptyset$ and $\text{delay}[\Sigma'](L_1) \cap \text{delay}[\Sigma'](L_2) = L_1$. The last inclusion we present is

$$\text{delay}[\Sigma'](L_1) \cap \overline{\text{delay}[\Sigma - \Sigma'](L_2)} \subseteq \text{delay}[\Sigma'](L_1 \cap \overline{\text{delay}[\Sigma - \Sigma'](L_2)}). \quad (\text{A4})$$

Proof. Consider an arbitrary string s contained in the left-hand side of (A4). Then,

$$s \in \overline{\text{delay}[\Sigma - \Sigma'](L_2)}, \quad \exists t \in L_1 \mid s \in \text{delay}[\Sigma'](\{t\}).$$

This implies

$$\begin{aligned} t &\in \text{delay}[\Sigma - \Sigma'](\{s\}) \\ &\subseteq \text{delay}[\Sigma - \Sigma'](\overline{\text{delay}[\Sigma - \Sigma'](L_2)}), \\ t &\in \overline{\text{delay}[\Sigma - \Sigma'](\overline{\text{delay}[\Sigma - \Sigma'](L_2)})} \\ &= \overline{\text{delay}[\Sigma - \Sigma'](\text{delay}[\Sigma - \Sigma'](L_2))} = \overline{\text{delay}[\Sigma - \Sigma'](L_2)}. \end{aligned}$$

In the last step we used (A1). Now, substituting t in the right-hand side of (A4) we obtain

$$\begin{aligned} \text{delay}[\Sigma'](\{t\} \cap \overline{\text{delay}[\Sigma - \Sigma'](L_2)}) &\subseteq \text{delay}[\Sigma'](L_1 \cap \overline{\text{delay}[\Sigma - \Sigma'](L_2)}), \\ \text{delay}[\Sigma'](\{t\}) &\subseteq \text{delay}[\Sigma'](L_1 \cap \overline{\text{delay}[\Sigma - \Sigma'](L_2)}), \\ s &\in \text{delay}[\Sigma'](L_1 \cap \overline{\text{delay}[\Sigma - \Sigma'](L_2)}), \end{aligned}$$

finally proving (A4). Q.E.D.

A.2. Proofs

A.2.1. Proof of Theorem 6.1.

THEOREM 6.1. The closed-loop languages resulting from the composition of two processes P and C subject to communication delays are given by the explicit expressions

$$\begin{aligned} L_P^c &= L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}, \\ L_C^c &= L_C \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}. \end{aligned}$$

Proof. From equation (12) we have directly $L_C^c \subseteq L_C \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$. Now, we have to prove the other inclusion $L_C^c \supseteq L_C \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$. This is proved by showing that equations (13) and (14) solve (11) and (12). For this it is sufficient

to show that the substitution of equations (13) and (14) into equation (12) yields the equality

$$\underbrace{L_C \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}}_{L_C^e} = \overline{\underbrace{L_C \cap \text{delay}[\Sigma_P]((L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}).\Sigma_C^*)}_{L_C \cap \overline{\text{delay}[\Sigma_P](L_P^e.\Sigma_C^*)}}}. \quad (\text{A5})$$

The inclusion

$$L_C \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} \supseteq \overline{L_C \cap \text{delay}[\Sigma_P]((L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}).\Sigma_C^*)}$$

is trivial because $L_1 \supseteq L_2$ implies $\overline{\text{delay}[\Sigma'](L_1)} \supseteq \overline{\text{delay}[\Sigma'](L_2)}$. Therefore, we must prove only the other inclusion in order to prove equation (A5). Let us consider equation (14) and an arbitrary string $s \in L_C \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$. Then there exists a string $t \in L_P$ such that

$$s.\Sigma_P^* \cap \text{delay}[\Sigma_P](t.\Sigma_C^*) \neq \emptyset$$

is satisfied. Then we proceed with the following steps:

$$\begin{aligned} s.\Sigma_P^* \cap \text{delay}[\Sigma_P](t.\Sigma_C^*) &\neq \emptyset, \\ \text{delay}[\Sigma_C](s.\Sigma_P^*) \cap t.\Sigma_C^* &\neq \emptyset, \\ \overline{\text{delay}[\Sigma_C](s.\Sigma_P^*)} \cap \{t\} &\neq \emptyset, \\ t &\in \overline{\text{delay}[\Sigma_C](s.\Sigma_P^*)}. \end{aligned}$$

Insertion of the strings s and t instead of L_C and L_P into the right side of equation (A5) yields the inclusion

$$\begin{aligned} \{s\} \cap \overline{\text{delay}[\Sigma_P]((\{t\} \cap \overline{\text{delay}[\Sigma_C](s.\Sigma_P^*)}).\Sigma_C^*)} \\ \subseteq \overline{L_C \cap \text{delay}[\Sigma_P]((L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}).\Sigma_C^*)}. \end{aligned}$$

Finally, using first the expression $t \in \overline{\text{delay}[\Sigma_C](s.\Sigma_P^*)}$ just derived above, and then the initial assumption $s \in \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$ we obtain

$$\begin{aligned} \{s\} \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} &\subseteq \overline{L_C \cap \text{delay}[\Sigma_P]((L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}).\Sigma_C^*)} \\ s &\in \overline{L_C \cap \text{delay}[\Sigma_P]((L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}).\Sigma_C^*)}. \end{aligned}$$

As this is true for any $s \in \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}$, we conclude

$$\overline{L_C \cap \text{delay}[\Sigma_P](L_P.\Sigma_C^*)} \subseteq \overline{L_C \cap \text{delay}[\Sigma_P]((L_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}).\Sigma_C^*)}.$$

This proves equality (A5), which in turn proves (13) and (14). Q.E.D.

A.2.2. Proof of Proposition 6.5. Before proving the proposition, we need the following lemma.

LEMMA A.1. Consider $P = (\Sigma, \Sigma_P, L_P, L_P)$ and $C = (\Sigma, \Sigma_C, L_C, L_C)$ with well-posed connection. The closed-loop behavior remains unchanged if we replace the original plant P by a plant $P' = (\Sigma, \Sigma_P, L'_P, L'_P)$ where $L'_P \supseteq L_P$ and L_P is $[L'_P, \Sigma_C]$ -controllable.

Proof. Define

$$\begin{aligned} L'_P{}^c &= L'_P \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}, \\ L'_C{}^c &= L_C \cap \overline{\text{delay}[\Sigma_P](L'_P.\Sigma_C^*)} \end{aligned}$$

to represent the closed-loop languages of the connection of C and P' . It is sufficient to show that $L'_P{}^c = L_P{}^c$ as this implies directly with equation (12)

$$L'_C{}^c = L_C \cap \overline{\text{delay}[\Sigma_P](L'_P.\Sigma_C^*)} = L_C \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} = L_C{}^c.$$

To show that $L'_P{}^c = L_P{}^c$ suppose that this is false, and therefore that there exists a string $t \in L'_P{}^c$ but $t \notin L_P{}^c$. Then there exists a smallest prefix t' of t with $t' \in L'_P{}^c$ not in $L_P{}^c$. We see that the string t' is of the form $t' = s.\sigma$ with $\sigma \in \Sigma$ and $s \in L_P$, $s.\sigma \in L'_P$, $s.\sigma \notin L_P$. Because L_P is $[L'_P, \Sigma_C]$ -controllable, i.e., $L_P.\Sigma_P^* \cap L'_P \subseteq L_P$, it follows that $\sigma \in \Sigma_C$. Also, from the assumption $s.\sigma \in L'_P{}^c$ we obtain $s.\sigma \in \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}$. Then with $s \in L_P$ and $\sigma \in \Sigma_C^*$, we have $s.\sigma \in L_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L_C.\Sigma_P^*)}$. But this contradicts the assumption on $s.\sigma$, since the condition for well-posedness of the connection given by (15) implies $s.\sigma \in L_P$.
Q.E.D.

Now we are ready to state our proposition.

PROPOSITION 6.5. The class $\mathcal{SC}[L_P, \Sigma_C](L)$ of self-well-posed and $[L_P, \Sigma_C]$ -controllable sublanguages of $L \subseteq L_P$ is closed under language union. There exists a supremal element of this class, denoted by $\sup \mathcal{SC}[L_P, \Sigma_C](L)$

Proof. Let L_1 and L_2 be two $[L_P, \Sigma_C]$ -controllable and self-well-posed sublanguages of L . Because of closedness under union of the class of controllable sublanguages, $L_1 \cup L_2$ is a $[L_P, \Sigma_C]$ -controllable sublanguage of L . We must therefore prove that

$$L_1 \cup L_2 \supseteq (L_1 \cup L_2).\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C]((L_1 \cup L_2).\Sigma_P^*)}, \quad (\text{A6})$$

$$L_1 \cup L_2 \supseteq (L_1 \cup L_2).\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P]((L_1 \cup L_2).\Sigma_C^*)}. \quad (\text{A7})$$

From Lemma A.1 we know that the behavior of the closed loop formed by a plant and a controller both with language L_i ($i = 1, 2$) remains the same if we use a plant with language L_P instead (the closed-loop languages do not change). As the connection of the controller with a plant with language L_P must also be well-posed,

we have

$$\begin{aligned} L_i &\supseteq L_i.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}, \\ L_P &\supseteq L_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L_i.\Sigma_P^*)}, \end{aligned}$$

for $i = 1, 2$. From the first inclusion it follows that

$$\begin{aligned} L_1 \cup L_2 &\supseteq (L_1 \cup L_2).\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} \\ &\supseteq (L_1 \cup L_2).\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P]((L_1 \cup L_2).\Sigma_C^*)}, \end{aligned}$$

proving (A6). Intersect both sides of the second inclusion with $\overline{\text{delay}[\Sigma_C](L_i.\Sigma_P^*)}$, then

$$\begin{aligned} L_P \cap \overline{\text{delay}[\Sigma_C](L_i.\Sigma_P^*)} &\supseteq L_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L_i.\Sigma_P^*)}, \\ (L_P^c)_i &\supseteq L_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L_i.\Sigma_P^*)}, \end{aligned}$$

where $(L_P^c)_i$ denotes the closed-loop language of a plant with language L_P connected to a controller with language L_i . From the arguments above $L_i \supseteq (L_P^c)_i$ (since $(L_P^c)_i$ is also the closed-loop language of a plant with language L_i), thus

$$\begin{aligned} L_1 \cup L_2 &\supseteq L_P.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L_1.\Sigma_P^*) \cup \text{delay}[\Sigma_C](L_2.\Sigma_P^*)} \\ &\supseteq (L_1 \cup L_2).\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L_1.\Sigma_P^*) \cup \text{delay}[\Sigma_C](L_2.\Sigma_P^*)} \\ &= (L_1 \cup L_2).\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C]((L_1 \cup L_2).\Sigma_P^*)}, \end{aligned}$$

proving also (A7). The existence of $\text{sup } \mathcal{SC}[L_P, \Sigma_C](L)$ follows directly from the closedness under union of the class. Q.E.D.

A.2.3. Proof of Proposition 6.6.

PROPOSITION 6.6. A prefix-closed memoryless language L is self-well-posed if and only if for $s \in L$, $t_P \in \Sigma_P^*$, and $t_C \in \Sigma_C^*$

$$s.t_P \in L, s.t_C \in L \Rightarrow s.\text{delay}[\Sigma_C]({t_C.t_P}) = s.\text{delay}[\Sigma_P]({t_P.t_C}) \subseteq L. \quad (22)$$

Proof. (ONLY IF) We prove this direction by induction. First we show that for $s \in L$, $\sigma_P \in \Sigma_P$, and $\sigma_C \in \Sigma_C$, the self-well-posedness of L implies $s.\sigma_C, s.\sigma_P \in L \Rightarrow s.\sigma_P.\sigma_C, s.\sigma_C.\sigma_P \in L$. This is readily proven by noting that $s.\sigma_P.\sigma_C \in L.\Sigma_C^*$ and $s.\sigma_C.\sigma_P \in L.\Sigma_P^*$. Then, from self-well-posedness of L we obtain

$$\begin{aligned} s.\sigma_P.\sigma_C &\in \{s.\sigma_P.\sigma_C\} \cap \overline{\text{delay}[\Sigma_C]({s.\sigma_C.\sigma_P})} \subseteq L.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](L.\Sigma_P^*)} \subseteq L, \\ s.\sigma_C.\sigma_P &\in \{s.\sigma_C.\sigma_P\} \cap \overline{\text{delay}[\Sigma_P]({s.\sigma_P.\sigma_C})} \subseteq L.\Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L.\Sigma_C^*)} \subseteq L. \end{aligned}$$

Now, we prove the necessary induction step. Define t'_P and t'_C with

$$t_P = \sigma_P.t'_P, \quad t_C = \sigma_C.t'_C.$$

Then, following steps similar to the ones carried above, now with $s.\sigma_P$ and $s.t_C$, we prove that $s.\sigma_P.t_C \in L$. Similarly we prove $s.\sigma_C.t_P \in L$. We have now reduced our problem to two smaller problems. The first problem arises when the next character after s is σ_P ; then, with $s' = s.\sigma_P$, we are left to prove

$$s'.t'_P, s'.t_C \in L \Rightarrow s'.\text{delay}[\Sigma_C](\{t_C.t'_P\}) = s'.\text{delay}[\Sigma_P](\{t'_P.t_C\}) \subseteq L.$$

The second problem, when the next character after s is σ_C and with the notation $s'' = s.\sigma_C$, requires proving

$$s''.t_P, s''.t'_C \in L \Rightarrow s''.\text{delay}[\Sigma_C](\{t'_C.t_P\}) = s''.\text{delay}[\Sigma_P](\{t_P.t'_C\}) \subseteq L.$$

This induction step shows that we can recursively split a problem into two smaller problems until we are able to prove that all strings in the set $s.\text{delay}[\Sigma_C](\{t'_C.t_P\})$ belong to L .

(IF) Suppose that the language L is memoryless, and that it satisfies condition (22). In order to prove self-well-posedness of L , it is sufficient to show, because of symmetry, that $s.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](t.\Sigma_P^*)} \subseteq L$ for any $s, t \in L$. For a finite string $s = \sigma_1.\sigma_2.\sigma_3.\dots.\sigma_n$ let us define the length of s , denoted by $\text{len}(s)$, to be n . We then remark that

$$\begin{aligned} \text{len}(\mathcal{P}[\Sigma_P](s)) = n_P^s &\geq n_P^t = \text{len}(\mathcal{P}[\Sigma_P](t)), \\ \text{len}(\mathcal{P}[\Sigma_C](s)) = n_C^s &\leq n_C^t = \text{len}(\mathcal{P}[\Sigma_C](t)), \end{aligned}$$

where n_P^x (n_C^x) denotes the number of elements of Σ_P (Σ_C) contained in the string x . Because of memorylessness and of condition (22) we have

$$\overline{\text{delay}[\Sigma_C](\{t\})} \cap \overline{\text{delay}[\Sigma_P](\{s\})} \subseteq L.$$

Take one of the longest strings from the left side of this inclusion. This string, which we denote by x , has length $n_P^t + n_C^s$. Then, because of construction of x , there exists $x_P \in \Sigma_P^*$, $\text{len}(x_P) = (n_P^s - n_P^t)$ such that $s \in \text{delay}[\Sigma_C](\{x.x_P\})$ and $x.x_P \in L$. Also there exists $x_C \in \Sigma_C^*$, $\text{len}(x_C) = (n_C^t - n_C^s)$ such that $t \in \text{delay}[\Sigma_P](\{x.x_C\})$ and $x.x_C \in L$. From (22) and $x.x_P, x.x_C \in L$ we have $x.x_P.x_C \in L$. Then, from memorylessness of L and with $x.x_P.x_C \in L$, we obtain $s.x_C \in L$. Finally, this implies $s.\Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](t.\Sigma_P^*)} = \{s.x_C\} \subseteq L$, which proves self-well-posedness of the language L . Q.E.D.

A.2.4. Proof of Theorem 6.5. We first need the following lemma.

LEMMA A.2. The sublanguage $\text{sup } \mathcal{W}[L_P, \Sigma_C](L)$ of L is $[L, \Sigma_C]$ -controllable.

Proof. We must prove that

$$\sup \mathcal{W}[L_P, \Sigma_C](L) \cdot \Sigma_P \cap L \subseteq \sup \mathcal{W}[L_P, \Sigma_C](L).$$

For this it is sufficient to show that for any $s \in \Sigma^*$ and $\sigma \in \Sigma$ satisfying

$$\begin{aligned} s &\in \sup \mathcal{W}[L_P, \Sigma_C](L), \\ s \cdot \sigma &\in L, \\ s \cdot \sigma &\notin \sup \mathcal{W}[L_P, \Sigma_C](L), \end{aligned}$$

we obtain $\sigma \in \Sigma_C$. Suppose instead $\sigma \in \Sigma_P$. Then, because the connection of the controller $C = (\Sigma, \Sigma_C, L_C, L_C)$ with language $L_C = \sup \mathcal{W}[L_P, \Sigma_C](L)$ with the plant $P = (\Sigma, \Sigma_P, L_P, L_P)$ is well-posed, we have from (15) and (16),

$$\begin{aligned} L_P &\supseteq L_P \cdot \Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](s \cdot \Sigma_P^*)}, \\ \sup \mathcal{W}[L_P, \Sigma_C](L) &\supseteq s \cdot \Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P \cdot \Sigma_C^*)}, \end{aligned}$$

and from $\sigma \cdot \Sigma_P^* \subseteq \Sigma_P^*$ we obtain

$$\begin{aligned} L_P &\supseteq L_P \cdot \Sigma_C^* \cap \overline{\text{delay}[\Sigma_C](s \cdot \sigma \cdot \Sigma_P^*)}, \\ \sup \mathcal{W}[L_P, \Sigma_C](L) &\supseteq s \cdot \sigma \cdot \Sigma_P^* \cap \overline{\text{delay}[\Sigma_P](L_P \cdot \Sigma_C^*)}. \end{aligned}$$

We remark that this implies that $\sup \mathcal{W}[L_P, \Sigma_C](L) \cup \{s \cdot \sigma\}$ has also a well-posed connection with L_P . But this contradicts the definition of $\sup \mathcal{W}[L_P, \Sigma_C](L)$ as being the supremal sublanguge of L with well-posed connection with L_P . Q.E.D.

THEOREM 6.5. Given a memoryless language L_P ,

$$\sup \mathcal{W}(L_P, \Sigma_C)(L_P) = \sup \mathcal{SC}(L_P, \Sigma_C)(L_P).$$

Proof. Let us use L_C as a shorthand notation for $\sup \mathcal{W}(L_P, \Sigma_C)(L_P)$. It is sufficient to prove that for a plant $P = (\Sigma, \Sigma_P, L_P, L_P)$ and a controller $C = (\Sigma, \Sigma_C, L_C, L_C)$ we obtain $L_P^c \subseteq L_C$, in which case the closed-loop behavior is unchanged, using L_C instead of L_P in the plant. The composition of a plant and a controller both with language L_C would then be well-posed, implying that the language L_C is self-well-posed.

For this we must show that there does not exist any string $s \in L_P^c$ with $s \notin L_C$. Suppose there exists such a string. It is actually sufficient to consider a string s such that $s' \in L_P^c$, $s' \in L_C$, $s = s' \cdot \sigma \in L_P^c \subseteq L_P$, and $s = s' \cdot \sigma \notin L_C$, where the symbol σ is in Σ_C because of $[L_P, \Sigma_C]$ -controllability of L_C (see the previous lemma). Then, from equation (13) there exists a string $t' \in L_C$ with $s \in \text{delay}[\Sigma_C](t' \cdot \Sigma_P^*)$ and $\mathcal{P}[\Sigma_C](t') = \mathcal{P}[\Sigma_C](s)$, i.e., with the same elements of Σ_C in s . Moreover, because of well-posedness of the composition, there exists also a string $t \in t' \cdot \Sigma_P^*$

and $t \in L_C$ with $s \in \text{delay}[\Sigma_C](\{t\})$. We now make two observations. The first one arises from the definition of L_C and inclusion (15):

$$\begin{aligned} L_P &\supseteq L_P.\Sigma_C^* \overline{\text{delay}[\Sigma_C](t.(L_C/t).\Sigma_P^*)} \\ &= L_P.\Sigma_C^* \overline{\text{delay}[\Sigma_C](\text{delay}[\Sigma_C](\{t\}).(L_C/t).\Sigma_P^*)} \\ &\supseteq L_P.\Sigma_C^* \overline{\text{delay}[\Sigma_C](s.(L_C/t).\Sigma_P^*)}. \end{aligned}$$

Our second observation starts with equation (16).

$$\begin{aligned} t.(L_C/t) &\supseteq t.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}, \\ \text{delay}[\Sigma_C](t.(L_C/t)) &\supseteq \text{delay}[\Sigma_C](t.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}) \\ &\supseteq \text{delay}[\Sigma_C](t.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}) \\ &\supseteq \text{delay}[\Sigma_C](\{t\}).(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)} \\ &\supseteq s.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}. \end{aligned}$$

Here, we used (A4) in the second step. Then, intersecting both sides of the inclusion with $s.(L_C/t).\Sigma_P^*$

$$\begin{aligned} s.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_C](t.(L_C/t))} &\supseteq s.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}, \\ s.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_C](L_C/t)} &\supseteq s.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}, \\ s.(L_C/t) &\supseteq s.(L_C/t).\Sigma_P^* \overline{\text{delay}[\Sigma_P](L_P.\Sigma_C^*)}, \end{aligned}$$

where we used in the last step the property $L = L.(\Sigma')^* \cap \text{delay}[\Sigma - \Sigma'](L)$. From the two observations just made and inclusions (15) and (16) we conclude that also a controller with language $L_C \cup s.(L_C/t)$ has a well-posed connection with a plant having language L_P .

Moreover, by memorylessness of L_P , we have $L_C/t \subseteq L_P/t = L_P/s$ and therefore $L_C \cup s.(L_C/t)$ is a sublanguage of L_P . But this is a contradiction of the assumption stating that L_C was defined to be $\sup \mathcal{W}(L_P, \Sigma_C)(L_P)$, i.e., the largest sublanguage of L_P for a controller yielding a well-posed connection with the given plant. Thus, we have proven that the language $\sup \mathcal{W}(L_P, \Sigma_C)(L_P)$ is self-well-posed. With the previous lemma and the final observation that

$$\sup \mathcal{W}(L_P, \Sigma_C)(L_P) \supseteq \sup \mathcal{SC}(L_P, \Sigma_C)(L_P)$$

we have proved the desired equality.

Q.E.D.

A.2.5. Proof of Proposition 7.2. We first need to introduce the following lemma.

LEMMA A.3. A well-posed composition of processes C and P with communication delays is nonblocking for C if and only if for all pairs (s_P, s_C) satisfying

$$\begin{aligned} s_C &\in L_C \cap \text{delay}[\Sigma_P](L_P) \subseteq L_C^c, \\ s_P &\in L_P \cap \text{delay}[\Sigma_C](\{s_C\}) \end{aligned}$$

the relation

$$s_C.(L_C^c/(s_P, s_C)) \cap M_C \neq \emptyset$$

holds.

Proof. We first remark that the pairs above are all pairs accepted by the processes when the buffers are empty.

We only need to prove the sufficiency of the condition stated in the lemma. Assume that there exists a pair (s_P, s_C) with $s_C.(L_C^c/(s_P, s_C)) \cap M_C = \emptyset$. Then, there must be a message in one or both communication channels. By well-posedness of the connection, the messages in the buffers $x_C \in \Sigma_C^*$ and $x_P \in \Sigma_P^*$ can be accepted by the respective processes. The pair $(s_P.x_C, s_C.x_P)$ therefore also satisfies condition (10), and from the statement of the lemma we obtain

$$s_C.x_P.(L_C^c/(s_P.x_C, s_C.x_P)) \cap M_C \neq \emptyset.$$

But this contradicts the condition stated in the proposition, as

$$s_C.(L_C^c/(s_P, s_C)) \supseteq s_C.x_P.(L_C^c/(s_P.x_C, s_C.x_P)),$$

implies

$$s_C.(L_C^c/(s_P, s_C)) \cap M_C \neq \emptyset$$

for the considered pair (s_P, s_C) , in contradiction with the assumption. Q.E.D.

Now we can prove the proposition.

PROPOSITION 7.2. Consider the well-posed composition of $C = (\Sigma, \overline{M_C}, M_C)$ and $P = (\Sigma, \overline{M_P}, M_P)$, with $M_C \subseteq M_P$ where $\overline{M_P}$ is a self-well-posed and memoryless language. The closed-loop is nonblocking for process C if and only if equation (23) is satisfied.

Proof. The necessity of equation (23) was proved in Proposition 7.1. Let us suppose that equation (23) holds, but the closed-loop is blocking for C . After Lemma A.3 there exist strings s_C and s_P and

$$\begin{aligned} s_C &\in L_C \cap \text{delay}[\Sigma_P](L_P) \subseteq L_C^c, \\ s_P &\in L_P \cap \text{delay}[\Sigma_C](\{s_C\}) \end{aligned}$$

such that

$$s_C.(L_C^c/(s_P, s_C)) \cap M_C = \emptyset$$

Let us consider any other possible pair of the form (s'_P, s_C) also satisfying condition (10). We first remark that as the proposition states $L_C \subseteq L_P$, we have $s_C \in L_P$. Furthermore, from well-posedness and inclusion (15), all strings s'_P will satisfy $s'_P.\Sigma_C^* \cap \text{delay}[\Sigma_C](s_C.\Sigma_P^*) \neq \emptyset$. Exploiting again the ideas used in the proof of Proposition 6.6, there exists a (not unique) longest string

$$x \in \overline{\text{delay}[\Sigma_P](s'_P)} \cap \overline{\text{delay}[\Sigma_C](s_C)} \subseteq L_P$$

and $x_P \in \Sigma_P^*$, $x_C \in \Sigma_C^*$ such that $s_C \in \text{delay}[\Sigma_P](x.x_C)$ and $s'_P \in \text{delay}[\Sigma_C](x.x_P)$ and $x.x_C, x.x_P \in L_P$. We also note that any possible continuation of s'_P in L_P^c after the pair (s'_P, s_C) is contained in a set of the form $s'_P.\text{delay}[\Sigma_C](x_C.t_P).t$ for some $t_P \in \Sigma_P^*$ and $t \in \Sigma^*$. Then, using memorylessness and self-well-posedness of L_P , we have the following steps:

$$\begin{aligned} s'_P.\text{delay}[\Sigma_C](x_C.t_P).t &\subseteq L_P, \\ x.x_P.x_C.t_P.t &\subseteq L_P, \\ x.x_C.x_P.t_P.t &\subseteq L_P, \\ s_C.x_P.t_P.t &\subseteq L_P. \end{aligned}$$

Noting that these steps hold for any $t_P \in \Sigma_P^*$ and $t \in \Sigma^*$ described above and noting also that $L_P/s_P = L_P/s_C$ by memorylessness of L_P , we continue with

$$\begin{aligned} s'_P.\text{delay}[\Sigma_C](x_C.t_P).t &\subseteq \text{delay}[\Sigma_C](s_C.x_P.t_P.t), \\ \overline{\text{delay}[\Sigma_C](s'_P.\text{delay}[\Sigma_C](x_C.t_P).t)} &\subseteq \overline{\text{delay}[\Sigma_C](s_C.x_P.t_P.t)}, \\ \overline{\text{delay}[\Sigma_C](s'_P.(L_P^c/(s'_P, s_C)))} &\subseteq \overline{\text{delay}[\Sigma_C](s_C.(L_P/s_C))}, \\ s_C.(L_C/s_C) \cap \overline{\text{delay}[\Sigma_C](s'_P.(L_P/s'_P))} &\subseteq s_C.(L_C/s_C) \cap \overline{\text{delay}[\Sigma_C](s_C.(L_P/s_C))} \\ &= s_C.(L_C/s_C) \cap \overline{\text{delay}[\Sigma_C](s_P.(L_P/s_P))}, \\ s_C.(L_C^c/(s'_P, s_C)) &\subseteq s_C.(L_C^c/(s_P, s_C)). \end{aligned}$$

Noting that s'_P is element of a pair (s'_P, s_C) described above but otherwise arbitrary, we conclude with the steps

$$\begin{aligned} s_C.(L_C^c/(s_P, s_C)) \cap M_C &= \emptyset, \\ s_C.(L_C^c/(s'_P, s_C)) \cap M_C &= \emptyset, \\ s_C.(L_C^c/s_C) \cap M_C &= \emptyset, \end{aligned}$$

in contradiction with equation (23).

Q.E.D.

Note

1. Following this argument, the term “well-posed” is more appropriate for denoting the property of a composition than the property of a supervisor, as proposed by Li and Wonham [1987].

References

- Balemi, S. 1991. A setup for real discrete event system control. Technical Report #91.07. Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland.
- Balemi, S. 1992a. *Control of Discrete Event Systems: Theory and Application*. Ph.D. thesis. Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH), Zürich, Switzerland.
- Balemi, S. 1992b. Discrete-event systems control of a rapid thermal multiprocessor. *Proc. of 7th IFAC/IFIP/IFORS/IMACS/ISPE Symp. on Information Control Problems in Manufacturing Technology (INCOM)*. Toronto, Canada. pp. 53–58.
- Balemi, S., Hoffmann, G., Gyugyi, P., Wong-Toi, H. and Franklin, G. 1993. Supervisory control of a rapid thermal multiprocessor. *IEEE Trans. Autom. Control* 38(7), pp. 1040–1059.
- Brandt, R., Garg, V., Kumar, R., Lin, F., Marcus, S. and Wonham, W. 1990. Formulas for calculating supremal controllable and normal sublanguages. *System and Control Letters* 15(2), pp. 111–117.
- Golaszewski, C. and Kurshan, R. 1991. Task-driven supervisory control of discrete event systems. in *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. Vol. 3, American Mathematical Society. pp. 251–273.
- Golaszewski, C. and Ramadge, P. 1987. Control of discrete event processes with forced events. *Proc. 26th IEEE Conf. Decision and Control*. Los Angeles, pp. 247–251.
- Heymann, M. 1990. Concurrency and discrete event control. *IEEE Control System Magazine* 10(4), pp. 103–112.
- Hoffmann, G. and Wong-Toi, H. 1992. Symbolic synthesis of supervisory controllers. *Proc. of 1992 American Control Conf.*. Chicago, pp. 2789–2793.
- Hopcroft, J. and Ullman, J. 1979. *Introduction to Automata Theory, Languages, and Computation*. Reading, MA: Addison-Wesley.
- Inan, K. 1989. Asynchronous dynamical systems I. Technical Report Memorandum No. UCB/ERL M89/59. Electronics Research Lab., Univ. of California at Berkeley.
- Li, Y. and Wonham, W. 1987. On supervisory control of real-time discrete event systems. *Proc. 1987 American Control Conf.*. Green Valley, AZ, American Autom. Control Council. Minneapolis, MN, pp. 1715–1720.
- Ramadge, P. and Wonham, W. 1987a. Modular feedback logic for discrete event systems. *SIAM J. Control Optim.* 25(5), pp. 1202–1218.
- Ramadge, P. and Wonham, W. 1987b. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.* 25(1), pp. 206–230.
- Ramadge, P. and Wonham, W. 1989. The control of discrete event systems. *Proc. of the IEEE* 77(1), pp. 81–98.
- Saraswat, K., Moslehi, M., Grossman, D., Wood, S., Wright, P. and Booth, L. 1989. Single wafer rapid thermal processing. *Mat. Res. Soc. Proc.*. Vol. 146. Material Research Society, pp. 3–13.
- Vaz, A. and Wonham, W. 1986. On supervisor reduction in discrete event systems. *Int. J. Control* 44(2), pp. 475–491.
- Wonham, W. and Ramadge, P. 1987. On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim.* 25(3), pp. 637–659.