

On nonconflicting languages that arise in supervisory control of discrete event systems *

Enke Chen and Stéphane Lafortune

*Department of Electrical Engineering and Computer Science,
University of Michigan, Ann Arbor, MI 48109-2122, USA*

Received 27 December 1990

Revised 22 April 1991

Abstract: We study four classes of nonconflicting sublanguages of a given language that arise in supervisory control of discrete event systems. We first present closed-form expressions for the supremal nonconflicting sublanguage and for the supremal closed nonconflicting sublanguage of a given language. The nonconflicting condition is with respect to a second given language. We then present algorithms to compute the supremal nonconflicting controllable sublanguage and the supremal closed nonconflicting controllable sublanguage of a given language. The regularity properties of these languages are also investigated.

Keywords: Discrete event systems; supervisory control; formal languages; nonconflicting languages; controllable languages.

1. Introduction

Let Σ be a non-empty finite set of events (alphabet) and denote by Σ^* the set of all finite traces of elements of Σ , including the empty trace ϵ . A subset $L \subseteq \Sigma^*$ is a *language* over Σ . Languages are used to model the logical behavior of (uncontrolled or controlled) discrete event processes. Several properties of languages such as, controllability, observability and normality, have been studied extensively in supervisory control of discrete event systems (see, e.g., [8]). This paper is concerned with the nonconflicting property of languages. This property was first introduced in [10]. Two languages L_1 and L_2 are said to be *nonconflicting* if whenever they share a prefix, they al-

so share a trace containing this prefix, i.e., $\overline{L_1 \cap L_2} = \overline{L_1} \cap \overline{L_2}$, where the overbar notation denotes the prefix-closure of a set. Closed (in the sense of prefix-closed) languages are always nonconflicting with one another.

The concept of nonconflicting languages finds applications in modular supervisory control [10] and in nonblocking supervisor design [2] of discrete event systems. For instance, it is shown in [10] that nonconflicting is a sufficient condition for the intersection of two controllable languages to be a controllable language. It is also shown in [10] that the conjunction of two nonblocking supervisors is nonblocking if and only if the two concerned languages are nonconflicting. In a different context, it is shown in [2] that the inner-blocking measure of a supervisor is empty if and only if two particular languages are nonconflicting (see [2], Section 3.2, for details).

When solving supervisor synthesis problems for discrete event systems, it is usually necessary to first calculate the supremal element of a certain class of languages, e.g., supremal controllable sublanguage [9], supremal normal sublanguage [6], etc. The same situation arises for the class of nonconflicting sublanguages of a given language (with respect to another fixed language). For instance, this is the case in [2], Section 3.3, where in order to synthesize the so-called ‘minimally restrictive non-innerblocking solution’ of the supervisory control problem with blocking, one must calculate the *supremal closed controllable nonconflicting* sublanguage of a particular language, an unsolved problem. The primary motivation of this paper is to address this computation and find algorithms to calculate the supremal closed controllable nonconflicting sublanguage. For this purpose, it is necessary to first deal with the computation of the supremal nonconflicting sublanguage, and then introduce the requirements of prefix-

* Research supported in part by the National Science Foundation under Grant ECS-9057967.

closure and controllability. From a general point of view, the results that we establish on these special classes of nonconflicting sublanguages will be of interest in other contexts as well.

More specifically, we introduce and study four nonconflicting sublanguages of a given language L : the supremal nonconflicting sublanguage (denoted L_{NC}), the supremal closed nonconflicting sublanguage (denote L_{NC}^c), the supremal nonconflicting controllable sublanguage (denoted L_{CNC}), and the supremal closed nonconflicting controllable sublanguage (denoted L_{CNC}^c). Here, the nonconflicting condition is with respect to a second given language, and the controllability condition is with respect to a third given language and a fixed set of uncontrollable events. We present closed-form expressions for the first two nonconflicting sublanguages and present algorithms for the computation of the last two nonconflicting sublanguages. We establish the finite convergence of these algorithms in the regular case based on a finite-state machine implementation of these algorithms.

Our presentation is organized as follows. Necessary background and preliminary results are presented in Section 2. L_{NC} and L_{NC}^c are defined and studied in Section 3, while Section 4 is devoted to L_{CNC} and L_{CNC}^c . Section 5 concludes the paper.

2. Preliminaries

We need to introduce some necessary background for the work that follows. If $s, s', t \in \Sigma^*$ with $s't = s$, then s' is a *prefix* of s ; thus both ε and s are prefixes of s . The *closure* \bar{L} of L is the language consisting of all the prefixes of traces in L ; if $L = \emptyset$ then $\bar{L} = \emptyset$, and if $L \neq \emptyset$ then $\varepsilon \in \bar{L}$. Clearly $L \subseteq \bar{L}$. L is *closed* if $L = \bar{L}$. A language is *regular* if and only if it is accepted by a finite automaton [3].

Let M be a fixed language over Σ , and let Σ_u be a fixed subset of Σ denoting the set of 'uncontrollable' events (in the sense that their occurrence cannot be disabled). A language $K \subseteq \Sigma^*$ is said to be *controllable* with respect to (w.r.t.) M and Σ_u if $\bar{K}\Sigma_u \cap M \subseteq \bar{K}$ [8]. The class of controllable sublanguages of a given language L is defined as

$$\mathcal{CL} := \{K : (K \subseteq L) \wedge (\bar{K}\Sigma_u \cap M \subseteq \bar{K})\}.$$

\mathcal{CL} has a *supremal* element (w.r.t. set inclusion) denoted $L^\dagger := \sup \mathcal{CL}$, i.e., $L^\dagger \in \mathcal{CL}$ and $K \in \mathcal{CL} \Rightarrow K \subseteq L^\dagger$. L^\dagger is called the *supremal controllable sublanguage* of L . Its computation is discussed in several references, among these [9,1,4].

We recall a property which is stated in [1].

Lemma 2.1 [1]. *If $B \subseteq \Sigma^*$ is closed, then for $\forall L \subseteq \Sigma^*$, the language $B - L\Sigma^*$ is also closed.* \square

The following result (whose proof is straightforward) will also be needed.

Lemma 2.2. *Let $L, R \subseteq \Sigma^*$ and $\bar{L} \cap R = \emptyset$. Then $\bar{L} \cap R\Sigma^* = \emptyset$ and $L \cap R\Sigma^* = \emptyset$.* \square

3. Supremal nonconflicting sublanguages

3.1. General case

Consider the following class of languages:

$$\mathcal{L}_{\text{NC}} := \{K : (K \subseteq L) \wedge (\overline{K \cap P} = \bar{K} \cap \bar{P})\} \quad (3.1)$$

where $L, P \subseteq \Sigma^*$ are two fixed languages. In words, \mathcal{L}_{NC} is the class of sublanguages of L that are nonconflicting with P . We characterize the supremal element (w.r.t. set inclusion) of \mathcal{L}_{NC} by the following result.

Theorem 3.1. (i) $L_{\text{NC}} := \sup \mathcal{L}_{\text{NC}}$ is well defined.

(ii) $L_{\text{NC}} = L - (\bar{L} \cap \bar{P} - \bar{L} \cap \bar{P})\Sigma^*$.

(iii) $\overline{L_{\text{NC}} \cap P} = \bar{L}_{\text{NC}} \cap \bar{P} = \bar{L} \cap \bar{P}$.

Proof. (i) We assume that $K_\alpha \in \mathcal{L}_{\text{NC}}$ for α in some index set, i.e.,

$$K_\alpha \subseteq L, \quad \overline{K_\alpha \cap P} = \bar{K}_\alpha \cap \bar{P}.$$

Then, $(\cup_\alpha K_\alpha) \subseteq L$. Also,

$$\begin{aligned} \overline{(\cup_\alpha K_\alpha) \cap P} &= \overline{\cup_\alpha (K_\alpha \cap P)} \\ &= \cup_\alpha (\overline{K_\alpha \cap P}) \\ &= \cup_\alpha (\bar{K}_\alpha \cap \bar{P}) \\ &= (\cup_\alpha \bar{K}_\alpha) \cap \bar{P} \\ &= \overline{\cup_\alpha K_\alpha} \cap \bar{P}. \end{aligned}$$

This shows that \mathcal{L}_{NC} is closed under arbitrary unions. Thus, $L_{\text{NC}} := \sup \mathcal{L}_{\text{NC}}$ is well defined.

(ii) Let

$$\text{RHS} := L - (\overline{L \cap \bar{P}} - \overline{L \cap P})\Sigma^*.$$

Obviously, when $\overline{L \cap \bar{P}} = \overline{L \cap P}$ (e.g., if $L = \emptyset$ or $P = \emptyset$), then $L_{\text{NC}} = \text{RHS} = L$. The equation $L_{\text{NC}} = \text{RHS}$ is valid in this case. The proof that $L_{\text{NC}} = \text{RHS}$ when $\overline{L \cap \bar{P}} \subset \overline{L \cap P}$ (thus $L \neq \emptyset$ and $P \neq \emptyset$) is organized into three steps.

Step 1. We need to show that $\text{RHS} \subseteq L$, which is obviously true.

Step 2. We need to show that RHS is nonconflicting with P , i.e.,

$$\overline{\text{RHS}} \cap \bar{P} = \overline{\text{RHS} \cap P}.$$

Let

$$\overline{L \cap \bar{P}} = \overline{L \cap P} \dot{\cup} R \quad (3.2)$$

where $\dot{\cup}$ denotes disjoint union, $R \neq \emptyset$ and $\overline{L \cap P} \cap R = \emptyset$. Lemma 2.2 implies that

$$\overline{L \cap \bar{P}} \cap R\Sigma^* = \emptyset, \quad (3.3)$$

$$(L \cap P) \cap R\Sigma^* = \emptyset. \quad (3.4)$$

Then, $\text{RHS} = L - R\Sigma^* \subseteq \overline{L} - R\Sigma^*$. Hence:

$$\begin{aligned} \overline{\text{RHS}} &\subseteq \overline{\overline{L} - R\Sigma^*} \\ &= \overline{L} - R\Sigma^* \quad (\text{by Lemma 2.1}). \end{aligned}$$

$$\begin{aligned} \overline{\text{RHS}} \cap \bar{P} &\subseteq (\overline{L} - R\Sigma^*) \cap \bar{P} \\ &= (\overline{L \cap \bar{P}}) - R\Sigma^* \\ &= (\overline{L \cap \bar{P}} \dot{\cup} R) - R\Sigma^* \\ &= \overline{L \cap \bar{P}} - R\Sigma^* \quad (\text{since } \varepsilon \in \Sigma^*) \\ &= \overline{L \cap \bar{P}} \quad (\text{by (3.3)}). \end{aligned}$$

$$\begin{aligned} \overline{\text{RHS} \cap P} &= \overline{(L - R\Sigma^*) \cap P} \\ &= \overline{(L \cap P) - R\Sigma^*} \\ &= \overline{L \cap P} \quad (\text{by (3.4)}). \end{aligned}$$

Therefore

$$\overline{\text{RHS}} \cap \bar{P} \subseteq \overline{\text{RHS} \cap P}. \quad (3.5)$$

Since the reverse inclusion of (3.5) is always true,

$$\overline{\text{RHS}} \cap \bar{P} = \overline{\text{RHS} \cap P} = \overline{L \cap \bar{P}} \quad (3.6)$$

which completes Step 2.

Step 3. It remains to show that RHS is the supremal nonconflicting (with P) sublanguage of

L . Let us proceed as in (3.2). Thus (3.3), (3.4) and (3.6) are still valid. Also, let

$$L_{\text{NC}} = \text{RHS} \dot{\cup} R_m. \quad (3.7)$$

As we know,

$$L_{\text{NC}} \subseteq L, \quad (3.8)$$

$$\overline{L_{\text{NC}}} \cap \bar{P} \subseteq \overline{L_{\text{NC}}} \cap \bar{P}. \quad (3.9)$$

Since

$$\text{RHS} = L - R\Sigma^* = L - (L \cap R\Sigma^*),$$

it follows that

$$\begin{aligned} L &= [L - (L \cap R\Sigma^*)] \dot{\cup} (L \cap R\Sigma^*) \\ &= \text{RHS} \dot{\cup} (L \cap R\Sigma^*) \end{aligned}$$

and

$$\begin{aligned} L_{\text{NC}} &= \text{RHS} \dot{\cup} R_m \\ &\subseteq L \quad (\text{by (3.8)}) \end{aligned}$$

Thus

$$R_m \subseteq (L \cap R\Sigma^*) \quad (3.10)$$

and so

$$\begin{aligned} R_m \cap P &\subseteq (L \cap P) \cap R\Sigma^* \\ &= \emptyset \quad (\text{by (3.4)}). \end{aligned}$$

Hence

$$R_m \cap P = \emptyset. \quad (3.11)$$

Substituting (3.7) in (3.9), we have

$$\overline{\text{RHS} \dot{\cup} R_m} \cap \bar{P} \subseteq \overline{(\text{RHS} \dot{\cup} R_m)} \cap \bar{P}$$

so that

$$(\overline{\text{RHS}} \cup \overline{R_m}) \cap \bar{P} \subseteq \overline{(\text{RHS} \cap P) \dot{\cup} (R_m \cap P)}$$

and thus

$$(\overline{\text{RHS}} \cap \bar{P}) \cup (\overline{R_m} \cap \bar{P}) \subseteq \overline{\text{RHS} \cap P} \quad (\text{by (3.11)}).$$

But by (3.6), we know that

$$\overline{\text{RHS}} \cap \bar{P} = \overline{\text{RHS} \cap P} = \overline{L \cap \bar{P}},$$

which yields

$$\overline{R_m} \cap \bar{P} \subseteq \overline{L \cap \bar{P}}. \quad (3.12)$$

If $L \cap P = \emptyset$, then by (3.12) we have $\overline{R_m} \cap \overline{P} = \emptyset$. Since $\varepsilon \in \overline{P}$, then $\varepsilon \notin \overline{R_m}$ which implies that $R_m = \emptyset$. Thus

$$L_{\text{NC}} = \text{RHS} = L - (\overline{L} \cap \overline{P})\Sigma^* = \emptyset.$$

If $L \cap P \neq \emptyset$, then $\varepsilon \in \overline{L} \cap \overline{P}$. Since $\varepsilon \in \overline{L} \cap \overline{P}$, it follows $\varepsilon \notin R = \overline{L} \cap \overline{P} - \overline{L} \cap \overline{P}$. Also by (3.10) we have

$$R_m \subseteq R\Sigma^* = (\overline{L} \cap \overline{P} - \overline{L} \cap \overline{P})\Sigma^*. \quad (3.13)$$

Since $\varepsilon \notin R$, then $\varepsilon \notin R_m$. Assume that $R_m \neq \emptyset$. Then $\exists s \in R_m$, $|s| \geq 1$. Then by (3.13) and by the facts that $R \neq \emptyset$ and $\varepsilon \notin R$, we have that $\exists s_1 \in \overline{\{s\}} \subseteq \overline{R_m}$, $|s_1| \geq 1$, such that $s_1 \in R = \overline{L} \cap \overline{P} - \overline{L} \cap \overline{P}$. Thus

$$s_1 \in \overline{L} \cap \overline{P} \subseteq \overline{P}, \quad s_1 \notin \overline{L} \cap \overline{P}. \quad (3.14)$$

Since $s_1 \in \overline{R_m}$ and $s_1 \in \overline{P}$, we have $s_1 \in \overline{R_m} \cap \overline{P} \subseteq \overline{L} \cap \overline{P}$ (by (3.12)). This contradiction with (3.14) shows that $R_m = \emptyset$, and thus $L_{\text{NC}} = \text{RHS}$.

(iii) This is immediate from the proof of (ii). \square

Since $L \cap P \subseteq P$, $L \cap P$ is nonconflicting with P . Then we have the following result which will be used in Section 4.1.

Corollary 3.1. *Let $L, P \subseteq \Sigma^*$. Then*

$$L \cap P \subseteq L - (\overline{L} \cap \overline{P} - \overline{L} \cap \overline{P})\Sigma^*. \quad \square$$

Theorem 3.1(ii) provides a closed-form expression for the supremal nonconflicting sublanguage L_{NC} of L . Since all the operations in the expression of L_{NC} preserve regularity, we have:

Corollary 3.2. *If L and P are regular, then L_{NC} is also regular. \square*

In the regular case, given finite automata generating the languages L and P , a finite automaton generating L_{NC} could thus be obtained by invoking standard methods from automata theory to implement the set difference, intersection, and

concatenation operations involved in the expression of L_{NC} [3].

3.2. Special case

Recall the definition of the closed sublanguage of L that is given in [5]:

$$\underline{L} := \{s \in L : \overline{\{s\}} \subseteq L\}.$$

It is easy to show that \underline{L} is regular whenever L is regular (see [5]).

Consider the following class of languages:

$$\mathcal{L}_{\text{NC}}^c := \{K : (K \subseteq L) \wedge (K = \overline{K}) \wedge (\overline{K \cap P} = \overline{K} \cap \overline{P})\} \quad (3.15)$$

where $L, P \subseteq \Sigma^*$ are two fixed languages. In words, $\mathcal{L}_{\text{NC}}^c$ is the class of sublanguages of L that are closed and nonconflicting with P .

Theorem 3.2. (i) $L_{\text{NC}}^c := \sup \mathcal{L}_{\text{NC}}^c$ is well defined.
(ii) $L_{\text{NC}}^c = \underline{L} - (\underline{L} \cap \overline{P} - \overline{\underline{L} \cap \overline{P}})\Sigma^*$.

Proof. (i) This result follows from Theorem 3.1(i) and the fact that arbitrary unions of closed languages yield a closed language.

(ii) We know that \underline{L} is the supremal closed sublanguage of L , so $L_{\text{NC}}^c \subseteq \underline{L}$. But by Theorem 3.1(ii),

$$\underline{L} - (\underline{L} \cap \overline{P} - \overline{\underline{L} \cap \overline{P}})\Sigma^*$$

is the supremal nonconflicting sublanguage of \underline{L} , and it is also closed by Lemma 2.1. Thus the equation is true by the definition of L_{NC}^c . \square

Remark 3.1. (i) If $\varepsilon \notin L$, then $\underline{L} = \emptyset$ and $L_{\text{NC}}^c = \emptyset$.

(ii) When $L = \overline{L}$, then $\underline{L} = L$ and thus $L_{\text{NC}}^c = L_{\text{NC}}$.

Theorem 3.2(ii) provides a closed-form expression for the supremal closed nonconflicting sublanguage $\mathcal{L}_{\text{NC}}^c$ of L . Again, it follows that:

Corollary 3.3. *If L and P are regular, then L_{NC}^c is also regular. \square*

4. Supremal nonconflicting controllable sublanguages

4.1. General case

Consider the following class of languages:

$$\mathcal{L}_{\text{CNC}} := \left\{ K : (K \subseteq L) \wedge (\overline{K \cap P} = \overline{K} \cap \overline{P}) \wedge (\overline{K \Sigma_u \cap M} \subseteq \overline{K}) \right\} \quad (4.1)$$

where $L, P, M \subseteq \Sigma^*$ are fixed languages, $\Sigma_u \subseteq \Sigma$ is a fixed set, and $L, P \subseteq M = \overline{M}$. In words, \mathcal{L}_{CNC} is the class of sublanguages of L that are nonconflicting with P and controllable with respect to M and Σ_u . Recall from Section 2 that \uparrow denotes the computation of the supremal controllable sublanguage. We have the following result:

Theorem 4.1. (i) $L_{\text{CNC}} := \sup \mathcal{L}_{\text{CNC}}$ is well defined.

(ii) L_{CNC} is the largest fixed point of the operator (on the sublanguages of L) $\Omega : 2^L \rightarrow 2^L$ defined by

$$\Omega(K) := \left[K - (\overline{K \cap P} - \overline{K} \cap \overline{P}) \Sigma^* \right]^\uparrow. \quad (4.2)$$

Proof. (i) This follows from Theorem 3.1(i) and the fact that arbitrary unions of controllable languages yield a controllable language.

(ii) The proof of this result is straightforward and hence omitted. \square

L_{CNC} is the supremal nonconflicting controllable sublanguage of L . Theorem 4.1(ii) suggests the following algorithm for the computation of L_{CNC} :

$$L_0 = L^\uparrow, \quad (4.3a)$$

$$L_{i+1} = \Omega(L_i), \quad i = 0, 1, 2, \dots, \quad (4.3b)$$

which is equivalent to

$$L_0 = L^\uparrow, \quad (4.4a)$$

$$L_{i+\frac{1}{2}} = L_i - (\overline{L_i \cap P} - \overline{L_i} \cap \overline{P}) \Sigma^*, \quad (4.4b)$$

$$L_{i+1} = (L_{i+\frac{1}{2}})^\uparrow, \quad i = 0, 1, 2, \dots \quad (4.4c)$$

Algorithm (4.4) is an iterative algorithm since each of the two steps in this algorithm may destroy the property of the other step. Observe that we could also have chosen $L_0 = L$ as the initial condition. Also, the two steps in (4.4) could be interchanged.

Theorem 4.2. (i) $L_\infty := \lim_{n \rightarrow \infty} L_n$ exists.

(ii) $L_{\text{CNC}} \subseteq L_\infty$.

Proof. (i) $\{L_n\}$ is a monotonically decreasing sequence lower bounded by \emptyset , thus $L_\infty := \lim_{n \rightarrow \infty} L_n$ exists and $L_\infty = \bigcap_{n=0}^\infty L_n$.

(ii) We know that $L_{\text{CNC}} \subseteq L_0 = L^\uparrow$. Assume that $L_{\text{CNC}} \subseteq L_k$. Since $L_{k+\frac{1}{2}}$ is the supremal nonconflicting sublanguage of L_k and L_{CNC} is also nonconflicting with P , we have $L_{\text{CNC}} \subseteq L_{k+\frac{1}{2}}$.

Then,

$$L_{\text{CNC}} = L_{\text{CNC}}^\uparrow \subseteq (L_{k+\frac{1}{2}})^\uparrow = L_{k+1},$$

which completes the induction step. Thus, $L_{\text{CNC}} \subseteq L_i$, $i = 0, 1, 2, \dots$, which implies that $L_{\text{CNC}} \subseteq \bigcap_{n=0}^\infty L_n = L_\infty$ and completes the proof. \square

We now prove that $L_\infty = L_{\text{CNC}}$ in the regular case (i.e., when L, P and M are regular languages) by showing that Algorithm (4.4) converges in a finite number of steps. Our approach for proving this result is based on the representation of regular languages by generators and it requires the subgenerator relation discussed in [4] and two lemmas that follow.

A generator $G = (Q, \Sigma, \delta, q_0, Q_m)$ is a deterministic finite automaton with a partially-defined transition function $\delta : \Sigma^* \times Q \rightarrow Q$, where Q is the state space, Σ the set of events, Σ^* the Kleene closure of Σ [3], q_0 the initial state, and $Q_m \subseteq Q$ the set of marked states. G is said to be *trim* if it is *accessible* (i.e., every state $q \in Q$ is reachable from q_0) and *co-accessible* (i.e., Q_m is accessible from any state $q \in Q$). As usual [7], $L(G)$ denotes the closed language *generated* by G , and $L_m(G)$ denotes the language *marked* by G . $L(G) = \overline{L_m(G)}$ if G is trim. We recall in the Appendix the definitions of the subgenerator relation (denoted \sqsubseteq) and the biased synchronous composition (denoted $\|_r$) from [4]. When $\langle q \rangle$ is a set of states and G is a generator, we will use the notation $G - \langle q \rangle$ to denote the generator G restricted to the states $Q - \langle q \rangle$, i.e.,

$$G - \langle q \rangle := (Q - \langle q \rangle, \Sigma, \delta|_{Q - \langle q \rangle}, q'_0, Q_m - \langle q \rangle)$$

where

$$q'_0 := \begin{cases} q_0 & \text{if } q_0 \in Q - \langle q \rangle, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is not difficult to prove that this restriction operation possesses the following properties:

Lemma 4.1. *Let $G_1 = (Q_1, \Sigma, \delta_1, q_0, Q_{m1})$ and $G_2 = (Q_2, \Sigma, \delta_2, q_0, Q_{m2})$ be generators such that $G_1 \sqsubseteq G_2$. Let $\langle q \rangle \subseteq Q_2$ be a set of states. Then we have*

- (i) $G_1 - \langle q \rangle \sqsubseteq G_2 - \langle q \rangle$.
 - (ii) $L(G_1 - \langle q \rangle) = L(G_2 - \langle q \rangle) \cap L(G_1)$.
 - (iii) $L_m(G_1 - \langle q \rangle) = L_m(G_2 - \langle q \rangle) \cap L_m(G_1)$.
 - (iv) $L_m(G_1 - \langle q \rangle) = L_m(G_2 - \langle q \rangle) \cap L_m(G_1)$
- if $L_m(G_1) \subseteq L_m(G_2)$. \square

Let

$$Q_{co} := \{q \in Q : \exists s \in \Sigma^*, \delta(s, q) \in Q_m\},$$

i.e., Q_{co} is the set of co-accessible states. Define $Q_{uco} := Q - Q_{co}$, i.e., Q_{uco} is the set of un-co-accessible states. We have:

Lemma 4.2. *Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be accessible and $L(G) = L$, $L_m(G) = N$. Let $\tilde{G} = (Q, \Sigma, \delta, q_0, Q_{uco})$. Then $L(\tilde{G}) = L$ and $L_m(\tilde{G}) = L - N$.*

Proof. The first equation is obviously true since marked states only affect the marked languages. To prove the second equation, let $s \in L - N$. Then $\delta(s, q_0) \in Q_{uco}$ and thus $s \in L_m(\tilde{G})$. This shows that $L_m(\tilde{G}) \supseteq L - N$. The reverse inclusion $L_m(\tilde{G}) \subseteq L - N$ is true due to the facts that $L_m(\tilde{G}) \subseteq L(\tilde{G}) = L$ and $L_m(\tilde{G}) \cap N = \emptyset$. This completes the proof. \square

We now present:

Theorem 4.3. *When L , P and M are regular languages, Algorithm (4.4) converges in finite steps.*

Proof. If L is regular, so is $L_0 = L^\uparrow$. Let \tilde{G} , \tilde{H} and \tilde{R} be generators such that

$$L(\tilde{G}) = L_m(\tilde{G}) = M,$$

$$L(\tilde{H}) = \bar{L}_0, \quad L_m(\tilde{H}) = L_0,$$

$$L(\tilde{R}) = \bar{P}, \quad L_m(\tilde{R}) = P.$$

Let

$$R_0 := \text{Ac}[\text{Ac}(\tilde{R} \times \tilde{H}) \times \tilde{G}],$$

$$H_0 := \text{Ac}[\text{Ac}(\tilde{R} \parallel_r \tilde{H}) \times \tilde{G}],$$

$$G := \text{Ac}[\text{Ac}(\tilde{R} \parallel_r \tilde{H}) \parallel_r \tilde{G}],$$

where Ac denotes taking the accessible component [7], \times denotes the product operation (also termed intersection, see, e.g., [4]), and \parallel_r denotes the biased synchronous composition (BSC) defined in [4] (see also the Appendix). By Lemma 5.1 in [4] (also recalled in the Appendix), we have

$$R_0 \sqsubseteq H_0 \sqsubseteq G,$$

$$L(G) = L(\tilde{G}) = M,$$

$$L_m(G) = L_m(\tilde{G}) = M,$$

$$L(H_0) = L(\tilde{H}) \cap L(\tilde{G}) = \bar{L}_0 \cap M = \bar{L}_0,$$

$$L_m(H_0) = L_m(\tilde{H}) \cap L_m(\tilde{G}) = L_0 \cap M = L_0,$$

$$L(R_0) = L(\tilde{R}) \cap L(\tilde{H}) \cap L(\tilde{G})$$

$$= \bar{L}_0 \cap \bar{P} \cap M = \bar{L}_0 \cap \bar{P},$$

$$L_m(R_0) = L_m(\tilde{R}) \cap L_m(\tilde{H}) \cap L_m(\tilde{G})$$

$$= L_0 \cap P \cap M = L_0 \cap P.$$

The following steps demonstrate that Algorithm (4.4) can be implemented on generators by removing certain transitions from R_0 and H_0 . This implementation of (4.4) proceeds as follows.

Step 0. Let $i = 0$.

Step 1. Given accessible generators H_i and R_i satisfying ¹

$$L_m(H_i) = L_i, \quad L(H_i) = \bar{L}_i, \quad (4.5a)$$

$$L_m(R_i) = L_i \cap P, \quad L(R_i) = \bar{L}_i \cap \bar{P}, \quad (4.5b)$$

$$R_i \sqsubseteq H_i \sqsubseteq G, \quad H_i \sqsubseteq H_0, \quad R_i \sqsubseteq R_0, \quad (4.5c)$$

build H_{i+1} and R_{i+1} by following Steps 1a–1d.

(Observe that the above R_0 and H_0 satisfy (4.5) as shown before.)

Step 1a. $R_{i+\frac{1}{2}}$ is obtained from R_i by removing its un-co-accessible states. Then $R_{i+\frac{1}{2}}$ is trim, and

$$L_m(R_{i+\frac{1}{2}}) = L_m(R_i) = L_i \cap P,$$

$$L(R_{i+\frac{1}{2}}) = \overline{L_m(R_{i+\frac{1}{2}})} = \bar{L}_i \cap \bar{P}.$$

¹ The first two conditions in (4.5) show that generator H_i indeed implements (4.3)–(4.4), while the other conditions are for the purpose of this proof.

By Lemma 4.2, $\overline{L_i} \cap \overline{P} - \overline{L_i} \cap \overline{P}$ is the set of traces of $L(R_i)$ lost due to the above removal of states. Denote by $\langle \delta \rangle$ the set of all the transitions (q_1, σ, q_2) of R_i where q_1 or q_2 is being removed.

Step 1b. $H_{i+\frac{1}{2}}$ is obtained from H_i by removing the transitions $\langle \delta \rangle$ of Step 1a and then taking the accessible component. Then

$$L_m(H_{i+\frac{1}{2}}) = L_i - (\overline{L_i} \cap \overline{P} - \overline{L_i} \cap \overline{P})\Sigma^* = L_{i+\frac{1}{2}}.$$

Clearly, $R_{i+\frac{1}{2}} \sqsubseteq H_{i+\frac{1}{2}}$. Moreover,

$$L_m(R_{i+\frac{1}{2}}) \subseteq L_m(H_{i+\frac{1}{2}})$$

by Corollary 3.1.

Step 1c. H_{i+1} is obtained from $H_{i+\frac{1}{2}}$ by first taking the trim operation, and then applying the algorithm in [4], Section V, for the computation of the \uparrow operation (this corresponds to comparing $H_{i+\frac{1}{2}}$ with G and removing states for controllability – cf. Step 2 of the above mentioned algorithm – and then taking the trim operation again). Denote by $\langle q \rangle$ the set of all the states removed in this process. Then

$$L_m(H_{i+1}) = L_m(H_{i+\frac{1}{2}} - \langle q \rangle) = L_{i+\frac{1}{2}}^\uparrow = L_{i+1},$$

$$L(H_{i+1}) = \overline{L_m(H_{i+1})} = \overline{L_{i+1}}.$$

Step 1d. R_{i+1} is obtained from $R_{i+\frac{1}{2}}$ by removing $\langle q \rangle$ and then taking the accessible component. By Lemma 4.1, we have

$$\begin{aligned} L_m(R_{i+1}) &= L_m(R_{i+\frac{1}{2}} - \langle q \rangle) \\ &= L_m(H_{i+\frac{1}{2}} - \langle q \rangle) \cap L_m(R_{i+\frac{1}{2}}) \\ &= L_m(H_{i+1}) \cap L_m(R_{i+\frac{1}{2}}) \\ &= L_{i+1} \cap (L_i \cap P) \\ &= L_{i+1} \cap P \quad (\text{since } L_{i+1} \subseteq L_i), \end{aligned}$$

$$\begin{aligned} L(R_{i+1}) &= L(R_{i+\frac{1}{2}} - \langle q \rangle) \\ &= L(H_{i+\frac{1}{2}} - \langle q \rangle) \cap L(R_{i+\frac{1}{2}}) \\ &= L(H_{i+1}) \cap L(R_{i+\frac{1}{2}}) \\ &= \overline{L_{i+1}} \cap \overline{L_i \cap P} \\ &\subseteq \overline{L_{i+1}} \cap \overline{P}. \end{aligned}$$

Since

$$\begin{aligned} \overline{L_{i+1}} \cap \overline{P} &\subseteq \overline{L_{i+\frac{1}{2}}} \cap \overline{P} \quad (\text{because } L_{i+1} = L_{i+\frac{1}{2}}^\uparrow \subseteq L_{i+\frac{1}{2}}) \\ &= \overline{L_i} \cap \overline{P} \quad (\text{by Theorem 3.1(iii)}) \end{aligned}$$

and

$$\overline{L_{i+1}} \cap \overline{P} \subseteq \overline{L_{i+1}},$$

it follows that

$$\overline{L_{i+1}} \cap \overline{P} \subseteq \overline{L_{i+1}} \cap \overline{L_i} \cap \overline{P}$$

and hence

$$L(R_{i+1}) = \overline{L_{i+1}} \cap \overline{P}.$$

Observe that $R_{i+1} \sqsubseteq H_{i+1} \sqsubseteq G$ remains valid. Also, H_{i+1} and R_{i+1} are accessible and satisfy (4.5) (with $i+1$ in place of i).

Step 2. Let $i \leftarrow i+1$. If states or transitions were removed in Steps 1a or 1c, return to Step 1. Else stop.

Since the number of states and transitions in H_0 is finite, the algorithm will converge in a finite number of steps. \square

Corollary 4.2. *When L , P and M are regular languages, $L_\infty = L_{\text{CNC}}$ and thus L_{CNC} is also regular.*

Proof. From Theorem 4.3, we know that $\exists N$, $0 \leq N < \infty$, such that $L_i = L_{i+\frac{1}{2}} = L_N$, $\forall i \geq N$, and thus $L_\infty = L_N$. But $\{L_i\}$ is a controllable sequence, and $\{L_{i+\frac{1}{2}}\}$ is a nonconflicting sequence, so L_∞ must be both controllable and nonconflicting and thus $L_\infty \subseteq L_{\text{CNC}}$. Together with Theorem 4.2(ii), we have $L_\infty = L_{\text{CNC}}$. The regularity follows by observing that each step in Algorithm (4.4) preserves regularity (Theorem 3.1 [9] and Corollary 3.2). \square

At present, it remains an open problem whether or not $L_\infty \subseteq L_{\text{CNC}}$ in the irregular case.

4.2. Special case

Consider the following class of languages:

$$\begin{aligned} \mathcal{L}_{\text{CNC}}^c := \{ &K : (K \subseteq L) \wedge (K = \overline{K}) \\ &\wedge (\overline{K \cap P} = \overline{K} \cap \overline{P}) \\ &\wedge (\overline{K} \Sigma_u \cap M \subseteq \overline{K}) \} \end{aligned} \quad (4.6)$$

where L , P , $M \subseteq \Sigma^*$ are fixed languages, $\Sigma_u \subseteq \Sigma$ is a fixed set and L , $P \subseteq M = \overline{M}$. In words, $\mathcal{L}_{\text{CNC}}^c$ is the class of sublanguages of L that are closed, nonconflicting with P , and controllable w.r.t. M and Σ_u .

It is straightforward to show that $L_{\text{CNC}}^c :=$

$\sup \mathcal{L}_{\text{CNC}}^c$ is well defined. L_{CNC}^c is the supremal closed, nonconflicting, and controllable sublanguage of L . For the computation of L_{CNC}^c , consider the following algorithm:

$$L_0 = (\underline{L})^\dagger, \quad (4.7a)$$

$$L_{i+1} = \Omega(L_i), \quad i = 0, 1, 2, \dots, \quad (4.7b)$$

where Ω is as defined in (4.2). Observe that each step in the above algorithm preserves closure of languages. Proceeding similarly to the previous section, we obtain the following results.

Theorem 4.4. (i) In (4.7), $L_\infty := \lim_{n \rightarrow \infty} L_n$ exists.

(ii) $L_{\text{CNC}}^c \subseteq L_\infty$.

(iii) When L , P and M are regular languages, $L_{\text{CNC}}^c = L_\infty$ and L_{CNC}^c is also regular. \square

Algorithm (4.7) can be applied for the computation of the minimally restrictive non-inner-blocking solution of the supervisory control problem with blocking studied in [2] (see Section 3.3 in that reference for further details on this problem).

5. Conclusion

We have discussed four classes of nonconflicting languages that arise in supervisory control of discrete event systems and proved several results pertaining to these classes. Our two main results are:

- Theorem 3.1, which provides a closed-form expression for the supremal nonconflicting sublanguage of a given language.

- Theorem 4.3, which demonstrates that whenever L , P and M are regular languages, the supremal nonconflicting (with P) controllable (w.r.t. M) sublanguage of L can be computed by a finite-step algorithm using generators of L , P and M and is thus also a regular language.

Acknowledgment

The authors would like to acknowledge useful discussions with Peter J. Ramadge concerning Section 3.1.

Appendix

Definition of subgenerator relation [4]

Consider two generators with the same alphabet Σ :

$$G_1 = (Q_1, \Sigma, \delta_1, q_{01}) \text{ and } G_2 = (Q_2, \Sigma, \delta_2, q_{02}).$$

We say that G_1 is a subgenerator of G_2 , denoted $G_1 \sqsubseteq G_2$, if $\delta_1(s, q_{01}) = \delta_2(s, q_{02})$ for all $s \in L(G_1)$. (Note that this condition implies that $q_{01} = q_{02}$ and $L(G_1) \subseteq L(G_2)$.)

Biased synchronous composition of generators [4]

Input:

$$G_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_{m1})$$

and

$$G_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, Q_{m2}).$$

Output:

$$G_1 \parallel_r G_2 := (Q_1 \times Q_2, \Sigma_2, \delta, (q_{01}, q_{02}), Q_1 \times Q_{m2})$$

where

$$\delta(\sigma, (q_1, q_2)) := \begin{cases} (\delta_1(\sigma, q_1), \delta_2(\sigma, q_2)) & \text{if } \sigma \in \Sigma_1(q_1) \cap \Sigma_2(q_2), \\ (q_1, \delta_2(\sigma, q_2)) & \text{if } \sigma \in \Sigma_2(q_2) - \Sigma_1(q_1) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Lemma A.1 [4]. Let G_1 and G_2 be two generators with $\Sigma_1 = \Sigma_2$. Let $G_1 \times G_2$ denote the product of G_1 and G_2 . Then:

(i) $(G_1 \times G_2) \sqsubseteq (G_1 \parallel_r G_2)$.

(ii) $L(G_1 \parallel_r G_2) = L(G_2)$.

(iii) $L_m(G_1 \parallel_r G_2) = L_m(G_2)$. \square

References

- [1] R.D. Brandt, V. Garg, R. Kumar, F. Lin, S.I. Marcus and W.M. Wonham, Formulas for calculating supremal controllable and normal sublanguages, *Systems Control Lett.* **15**(2) (1990) 111–117.
- [2] E. Chen and S. Lafortune, Dealing with blocking in supervisory control of discrete event systems, *IEEE Trans. Automat. Control* **36**(6) (1991) 724–735.

- [3] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).
- [4] S. Lafortune and E. Chen, The infimal closed controllable superlanguage and its application in supervisory control, *IEEE Trans. Automat. Control* **35**(4) (1990) 398–405.
- [5] S. Lafortune and E. Chen, On controllable languages in supervisory control of discrete event systems, in: M.A. Kaashoek, J.H. van Schuppen, and A.C.M. Ran, Eds., *Realization and Modelling in System Theory, Proceedings of the International Symposium MTNS-89, Vol. 1* (Birkhäuser, Basel–Boston, 1990) 541–548.
- [6] F. Lin and W.M. Wonham, On observability of discrete-event systems, *Inform. Sci.* **44**(3) (1988) 173–198.
- [7] P.J. Ramadge and W.M. Wonham, Supervisory control of a class of discrete event systems, *SIAM J. Control Optim.* **25**(1) (1987) 206–230.
- [8] P.J. Ramadge and W.M. Wonham, The control of discrete event systems, *Proc. IEEE* **77**(1) (1989) 81–98.
- [9] W.M. Wonham and P.J. Ramadge, On the supremal controllable sublanguage of a given language, *SIAM J. Control Optim.* **25**(3) (1987) 637–659.
- [10] W.M. Wonham and P.J. Ramadge, Modular supervisory control of discrete-event systems, *Math. Control Signals Systems* **1**(1) (1988) 13–30.