Supervisory Control Theory with Event Forcing

Michel Reniers, *Senior Member, IEEE,* and Kai Cai, *Senior Member, IEEE*

***Abstract*—In the Ramadge-Wonham supervisory control theory the only interaction mechanism between supervisor and plant is that the supervisor may enable/disable events from the plant and the plant makes a final decision about which of the enabled events is actually taking place. In this paper, the interaction between supervisor and plant is enriched by allowing the supervisor to force specific events (called forcible events) that are allowed to preempt uncontrollable events. A notion of *forcible-controllability* is defined that captures the interplay between controllability of a supervisor w.r.t. the uncontrollable events provided by a plant in the setting with event forcing. Existence of a maximally permissive, forcibly-controllable, nonblocking supervisor is shown and an algorithm is provided that computes such a supervisor. The approach is illustrated by two small case studies.**

***Index Terms*—Discrete event systems, finite automata, forcible events, forcible controllability supervisory control, nonblocking.**

I. Introduction

THE main interaction mechanism between a supervisor and the plant it controls in supervisory control theory (SCT) [1], [2] is the mechanism of enabling/disabling events offered by the plant. In such a setting the supervisor indicates which of the events that are enabled in the plant are allowed to occur, but the supervisor does not dictate which of these allowed events will occur.

One can observe that in many implementations of supervisory controllers a different interaction involving event forcing is used between supervisor and plant.

We mention two important consequences of this 'mismatch' between this assumption in SCT and the actual setting in implementations: (1) a proper supervisory controller does not necessarily result in a proper implementation (especially w.r.t. the infamous property of nonblocking there are complications), and (2) loss of permissiveness because final choice of executed event is assumed to be in the plant. In [3], [4], [5], the mismatch between a proper supervisory controller and an implementation is discussed in detail.

In this paper we study the second consequence in more detail. We propose a SCT that does not only allow enabling/disabling by the supervisor, but also, for some events, allows forcing of such an event. Forcing of an event results in preemption of other events and may thus contribute to obtaining proper supervisors.

In [6], supervisory control of untimed discrete event systems with forcible events is studied. In [6, Example 2], a version of controllability was introduced (when interpreted in a setting with uncontrollable events) which requires that a sublanguage is closed under all uncontrollable events available in the plant or allows for a single forced event only. A clear difference with our definition is that we generally allow multiple forcible events to be available for the sake of maximal freedom of choices. In [6], this specific condition that only one forcible event is considered an acceptable continuation is not further elaborated outside that example.

In [1, section 3.8] forcible events are introduced also for a setting without timing. It is argued (informally) that forcible events are a modeling issue by showing how synthesis in the context of forcible events can be achieved by traditional synthesis on a transformed plant. Drawbacks of this approach are the possible increase of the number of states in the state space on which synthesis is to be performed, and the fact that the transformation as suggested takes place on the state space and not on the individual plant components typically used in modeling an uncontrolled system. This paper considers the same setting, but studies a direct synthesis algorithm for the setting of supervisory control with forcible events (thus countering both disadvantages).

In extensions of SCT with notions of timing, such as different types of Timed DES [7], [8], [9], [10], [11] and TA [12], forcible events are introduced to preempt progress of time (which is generally considered uncontrollable). In all these works, the result is an adapted notion of controllability where progress of time is consider uncontrollable only in cases where no forcible events are enabled. A main difference with our approach is that we consider the status of every event as being either controllable or uncontrollable as fixed and given. Additionally, events may be forcible, which allows them to be used by the supervisor to preempt events (not only progress of time, but also other events).

In [7], a distinction is made between strongly preemptive forcing, where a forcible event preempts any other eligible event, and weakly preemptive forcing, where premption is only assumed w.r.t. the passage of time. In [8], [9], weakly preemptive forcing is considered. In contract, the notion of preemption adopted in our paper is that of strong preemption with the side note that a supervisor may have multiple forcible event alternatives instead of exactly one. A notion of controllability is defined that depending on the eligibility of a forcible event treats the event $tick$ as uncontrollable or not (uncontrollable when no forcible event is eligible).

In [10], the supervisory control of a plant is studied where the interaction between supervisor and plant is assumed to result in delays. Time is modeled by means of a tick event, and forcible events are used to preempt this tick event (and no other events). The notion of controllability used in [10] aligns well with the notion defined in this paper. Although the synthesis algorithm presented in [10] is quite different from the version in this paper, the underlying idea that a tick event

M. Reniers is with the Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. E-mail: m.areniers@tue.nl.

K. Cai is with Department of Core Informatics, Osaka Metropolitan University, Osaka, Japan. E-mail: cai@omu.ac.jp.

may be preempted by a forcible event is recognizable and also the situation that during synthesis all forcible events may disappear resulting in these states being bad states after all is there.

In both [11] and [12], forcible events are used to preempt the progress of time in a context of timed automata. In such automata the progress of time is, in contrast with Timed DES, described by real-valued clocks and separated from the occurrence of events.

In [13], forcible events are used in the context of hybrid automata. The treatment is restricted to the setting where all forcible events are also controllable, an assumption that is not adopted in the present paper.

In [14], existence and synthesis of safe and nonblocking directed control for discrete event systems is studied. A directed controller selects at most one controllable event to be executed from each system state. Our forcibly-controllable supervisor is possibly more permissive than a directed controller as multiple controllable events may still be allowed in order to maximize permissiveness in the sense of choosing alternative forcible events.

In [4], supervised control is applied. In supervised control a supervisor is meant to be implemented together with a separate controller. The supervisor is used to monitor the behavior of the plant and disables some events, whereas the controller forces some of the events enabled by the supervisor to occur in the plant. Forcible events are used in [4], but not for the synthesis of a maximally permissive supervisor, but for the implementation of such a supervisor as a controller. In all the four case studies mentioned in the paper all controllable events were assumed to be forcible by the controller.

In [15], the authors point to three reasons for limited applicability of supervisory control theory. One of these three reasons is that the model interpretation of SCT does not connect well with applications: "The logical plant model proposed in supervisory control theory assumes a plant that 'generates' events spontaneously unless it is prevented from doing so. The control mechanism available to the supervisor is the ability to prevent the occurrence of some events, called controllable events. This model is not appropriate for most real systems. In fact, real systems usually react to commands as inputs with responses as outputs." The authors assume an input-output model where the supervisor has to be controllable and the plant has to be complete for commands issued by the supervisor.

In the present paper, we enrich the traditional options for a supervisor to enable/disable events with the option to preempt events by forcing some event to occur. We however, do not assume that the plant is complete for such commands as [15] does. Instead we formulate a new forcible-controllability property that the supervisor must satisfy.

The contributions of this paper can be summarized as follows:

1) In this paper we provide a basic untimed supervisory control framework where besides the traditional enabling/disabling of events generated by the plant also event forcing is considered.
2) We formulate a notion of forcible controllability that captures the interplay between controllability of a supervisor w.r.t. the uncontrollable events provided by a plant in the setting where forcing of events may be used to preempt such uncontrollable events.
3) We provide the basic properties of forcible-controllability and show that a maximally permissive forcibly-controllable supervisor exists.
4) We provide a synthesis algorithm that computes such a supervisor for a given plant automaton.

Structure of the paper: In Section II some preliminaries about supervisory control theory are introduced that are related to the subject material of this paper, and the forcing supervisory control problem is formulated. Then, in Section III, the notion of forcible-controllability is introduced that is crucial in capturing the interplay between plant and (forcing) supervisor. Properties of this notion are discussed in detail, and it is proved that this notion is necessary and sufficient for the solvability of the formulated forcing supervisory control problem. In Section IV it is shown that the supremal forcibly-controllable sublanguage of a given language exists, which provides a maximally permissive solution to the forcing supervisory control problem. Section V presents an algorithm for computing the maximally permissive supervisor and states its properties such as termination and correctness. Sections VI and VII illustrate the outcome of the algorithm and showcase the benefit of introducing event forcing. The paper is concluded in Section VIII.

II. Forcible Events and Forcing Supervisory Control

Consider that a plant to be controlled is modeled by a finite automaton $P = (Q, \Sigma, \delta, q_0, Q_m)$ where Q is a finite set of states, Σ is a finite set of events partitioned into controllable events Σ_c and uncontrollable events Σ_u, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $q_0 \in Q$ is the initial state, and $Q_m \subseteq Q$ is the set of marker states. A controllable event $c \in \Sigma_c$ can be disabled by an external agent (supervisor), whereas an uncontrollable event $u \in \Sigma_u$ cannot be disabled.

Plant P generates a closed language $L(P)$ and a marked language $L_m(P)$, as defined in the usual way [1].

Definition 1 (Nonblocking)**.** A finite automaton $P = (Q, \Sigma, \delta, q_0, Q_m)$ is called *nonblocking* if $L(P) = \overline{L_m(P)}$.

For a string $s \in L(P)$, let $E_P(s) := \{\sigma \in \Sigma \mid s\sigma \in L(P)\}$ be the subset of eligible events that can occur after s in P.

Definition 2 (Controllable sublanguage)**.** Given a (nonblocking) plant P, a sublanguage $F \subseteq L_m(P)$ is *controllable* w.r.t. P if

$$(\forall s \in \overline{F},\ \forall u \in \Sigma_u)\ [u \in E_P(s) \implies su \in \overline{F}].$$

Controllable sublanguages are closed under union (see e.g., [1]). Thus given an arbitrary language $F \subseteq L_m(P)$, there exists the supremal controllable sublanguage of F.

Now bring in a subset of *forcible events* $\Sigma_f \subseteq \Sigma$. A forcible event $f \in \Sigma_f$ can be forced by an external agent in order to preempt other event occurrences. Thus forcible

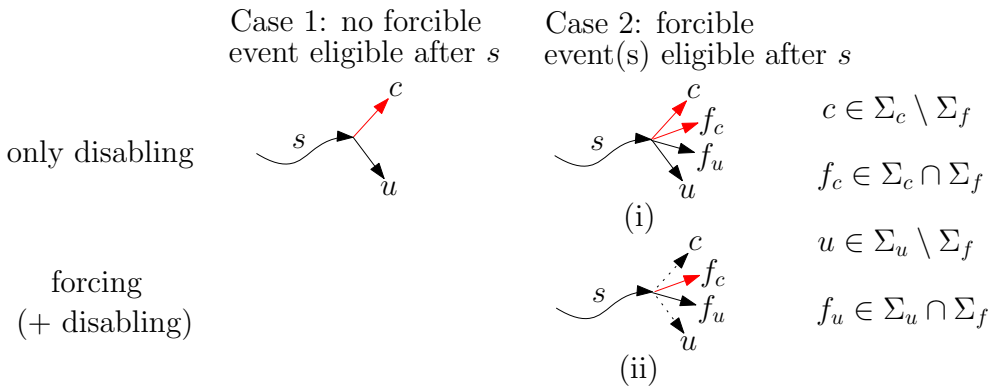


Fig. 1. Control mechanisms for supervisory control.

events provide a new mechanism for control, which is distinct from controllable events (that can be disabled themselves but cannot preempt other events).

A forcible event $f \in \Sigma_f$ can be either controllable or uncontrollable. If $f \in \Sigma_f \cap \Sigma_c$, then f can either be forced or be disabled. For example, 'cross_street' can be an event that a pedestrian can force (by accelerating upon a slowly approaching car) or disabled (i.e. giving up crossing upon a fast approaching car). On the other hand, if $f \in \Sigma_f \cap \Sigma_u$, then f can be forced but cannot be disabled. An example of such an uncontrollable forcible event is 'land_plane', which may be forced (say within 15 minutes) but cannot be prevented (plane has to land eventually) [1].

With this added forcing mechanism, we define the corresponding *supervisory control* for plant P which is any function $V : L(P) \to Pwr(\Sigma)$ (here $Pwr(\cdot)$ denotes powerset). Let us analyze the characteristics of this supervisory control V. Consider a strings $s \in L(P)$, and let c, f_c, u, f_u respectively be a nonforcible controllable, forcible controllable, nonforcible uncontrollable, forcible uncontrollable event. As displayed in Fig. 1, we discuss two cases.

Case 1: $E_P(s) \cap \Sigma_f = \varnothing$, i.e. no forcible event is eligible after s. In this case, only controllable events may be disabled (e.g. c in Fig. 1, Case 1).

Case 2: $E_P(s) \cap \Sigma_f \neq \varnothing$, i.e., there exists at least one forcible event eligible after s. In this case, the eligible forcible event(s) may be either forced or not forced. When all the eligible forcible events are not forced (see Fig. 1(i), Case 2), again only controllable events may be disabled (e.g., c, f_c in Fig. 1(i)). On the other hand, if an eligible forcible event is forced (see Fig. 1(ii), Case 2), then all the other eligible events after s are preempted. Since in general there are multiple forcible events that may be forced after s (e.g., f_c, f_u in Fig. 1(ii)), we will define the supervisory control function to include all such forcible events for the sake of maximal freedom of choice. In implementation, one of these forcible events will be chosen (say by an external forcing agent) such that other events are preempted; our model will leave such a choice nondeterministic (much like the nondeterministic occurrence of one event among multiple enabled events, e.g., u, f_u in Fig. 1(i)). It is also noted that disabling a forcible controllable event is also possible (e.g., f_c in Fig. 1(ii)), and such disabled forcible events cannot be forced. On the other hand, a forcible uncontrollable event cannot be disabled (e.g., f_u in Fig. 1(ii)), and such forcible events may be either forced or not forced.

With the above discussion on possible scenarios of disabling and forcing control mechanisms, we define the supervisory control $V : L(P) \to Pwr(\Sigma)$ such that

$$V(s) = \begin{cases} \Sigma_u \cup \Sigma'_c & \text{if } E_P(s) \cap \Sigma_f = \varnothing, \\ \text{either } \Sigma_u \cup \Sigma'_c \text{ or } \Sigma'_f & \text{if } E_P(s) \cap \Sigma_f \neq \varnothing. \end{cases} \tag{1}$$

Here $\Sigma'_c \subseteq \Sigma_c$, $\Sigma'_f \subseteq \Sigma_f$. The first line corresponds to Case 1 above, whereas the second line to Case 2. In the second line, 'either $\Sigma_u \cup \Sigma'_c$' is when none of the eligible forcible events are forced (Fig. 1(i)), while 'or Σ'_f' is when at least one eligible forcible event is forced (Fig. 1(ii)).

Write V/P for the *closed-loop system* where the plant P is under the supervisory control of V. The closed language generated by V/P is defined as follows:

(i) $\epsilon \in L(V/P)$;
(ii) $s \in L(V/P), \sigma \in V(s), s\sigma \in L(P) \Rightarrow s\sigma \in L(V/P)$;
(iii) no other strings belong to $L(V/P)$.

Let $F \subseteq L_m(P)$. The marked language of V/P w.r.t. F is

$$L_m(V/P) := L(V/P) \cap F.$$

We say that V/P is nonblocking if

$$L(V/P) = \overline{L_m(V/P)}.$$

Now we are ready to formulate the forcing supervisory control problem.

Problem 1 (Forcing supervisory control problem)**.** *Given a plant P and a specification $\varnothing \neq F \subseteq L_m(P)$, design a supervisory control $V : L(P) \to Pwr(\Sigma)$ such that*

(i) *V/G is nonblocking;*

(ii) $L_m(V/G) = F$.

In words, Problem 1 is after a supervisory control that not only renders the closed-loop system nonblocking but also synthesizes the imposed specification.

III. Forcibly-Controllable Sublanguages

s

To solve Problem 1, the following new language property is key.

Definition 3 (Forcibly-controllable sublanguage)**.** Given a (nonblocking) plant P, a sublanguage $F \subseteq L_m(P)$ is *forcibly-controllable* w.r.t. P if

$$(\forall s \in \overline{F}) \begin{bmatrix} (\forall u \in \Sigma_u)\ [u \in E_P(s) \implies su \in \overline{F}] \\ \vee \\ (\exists f \in \Sigma_f)\ [sf \in \overline{F}] \wedge (\forall \sigma \in \Sigma \setminus \Sigma_f)\ [s\sigma \notin \overline{F}] \end{bmatrix}.$$

In words, a sublanguage F is forcibly-controllable if either F is controllable or there exists a forcible event that keeps $\overline{F}$ invariant by preempting all non-forcible events.

Thus by definition, if a sublanguage F is controllable, F is also forcibly-controllable. The difference between these two properties is illustrated in Fig. 2(ii) and (iv). While controllability does not allow uncontrollable event u to take string s out of $\overline{F}$ (Fig. 2(ii)), forcible-controllability allows so if there exists a forcible event f after s to keep sf inside $\overline{F}$ by preempting u (Fig. 2(iv)). Based on Fig. 2(iv), it is readily seen that a forcibly-controllable sublanguage need not be controllable.

Now that we have seen that controllability involves pure disabling, forcible-controllability invovles both disabling and forcing, it is natural to define another language property which involves pure forcing. In the following, we define this language property – forcible sublanguages – which is not needed in our main development but interesting in itself as compared to the other language properties.

Definition 4 (Forcible sublanguage)**.** Given a (nonblocking) plant P, a sublanguage $F \subseteq L_m(P)$ is *forcible* w.r.t. P if

$$(\forall s \in \overline{F}) \begin{bmatrix} (\forall \sigma \in \Sigma)\ [\sigma \in E_P(s) \implies s\sigma \in \overline{F}] \\ \vee \\ (\exists f \in \Sigma_f)\ [sf \in \overline{F}] \wedge (\forall \sigma \in \Sigma \setminus \Sigma_f)\ [s\sigma \notin \overline{F}] \end{bmatrix}.$$

Compared to Definition 3, forcible sublanguage is defined by only replacing the universal quantification on the first line with $u \in \Sigma_u$ by $\sigma \in \Sigma$. This change means that whenever a string s is taken out of $\overline{F}$ by any event σ, controllable or uncontrollable, there must exist at least one forcible event f after s to keep sf inside $\overline{F}$ by preempting σ. This is illustrated in Fig. 2(v) and (vi).

Thus by definition, if a sublanguage F is forcible, F is also forcibly-controllable. But the reverse need not be true: a forcibly-controllable sublanguage allows s to exit $\overline{F}$ by a controllable event without the presence of a forcible event (Fig. 2(iii)), which is not allowed by a forcible sublanguage.

In addition, it is not difficult to see that forcible sublanguages and controllable sublanguagess generally are not related. A forcible sublanguage need not be controllable, because the scenario in Fig. 2(vi) is allowed by a forcible sublanguage but not by a controllable sublanguage. Conversely, since the scenario in Fig. 2(i) is allowed by controllable sublanguage but not by forcible sublanguage, a controllable sublanguage need not be forcible.

We summarize the above relationships among the three language properties.

Fact 1. Let plant P be a finite automaton over $\Sigma = \Sigma_c \cup \Sigma_u$ and a forcible event set $\Sigma_f \subseteq \Sigma$. Also let $F \subseteq L_m(P)$.

1) If F is controllable, it is also forcibly-controllable.
2) If F is forcible, it is also forcibly-controllable.
3) There are focibly-controllable sublanguages that are not controllable.
4) There are forcibly-controllable sublanguages that are not forcible.
5) There are controllable sublanguages that are not forcible.
6) There are forcible sublanguages that are not controllable.

At this point, it is also convenient to state the following facts about the three language properties.

Fact 2. Let plant P be a finite automaton over $\Sigma = \Sigma_c \cup \Sigma_u$ and a forcible event set $\Sigma_f \subseteq \Sigma$.

1) The empty language $\varnothing$ is controllable, forcibly-controllable, and forcible.
2) The closed language $L(P)$ is controllable, forcibly-controllable, and forcible.
3) Let $\Sigma_f = \varnothing$. Then F is forcibly-controllable iff F is controllable.
4) Let $\Sigma_c = \varnothing$. Then F is controllable iff F is forcible iff F is forcibly-controllable.
5) Let $\Sigma_u = \varnothing$. Any sublanguage $F \subseteq L_m(P)$ is controllable and forcibly-controllable.

Proof.

1) Since $\overline{\varnothing} = \varnothing$, by Definitions 2, 3, and 4, the empty language is vacuously controllable, forcibly-controllable, and forcible.
2) Since $\overline{L(P)} = L(P)$, we have both $\overline{L(P)} \subseteq L(P)$ and $s\sigma \in L(P) \implies s\sigma \in \overline{L(P)}$ for any s and σ. Therefor by definition $L(P)$ is controllable, forcibly-controllable, and forcible.
3) For $\Sigma_f = \varnothing$, the second line of logic formula in Definition 3 is always false. Thus the definition of forcible-controllability reduces to the definition of controllability, and the conclusion holds.
4) For $\Sigma_c = \varnothing$ we have $\Sigma = \Sigma_u$. In this case the definitions of forcible-controllability and forcibility are identical, and hence the conclusion holds.
5) Let $\Sigma_u = \varnothing$ and $F \subseteq L_m(P)$. According to the logic formulas in the definitions of controllable and forcibly-controllable sublanguages, F is vacuously controllable and forcibly-controllable.

□

Now we state the main result of this section, which characterizes solvability of Problem 1 by forcible-controllability.

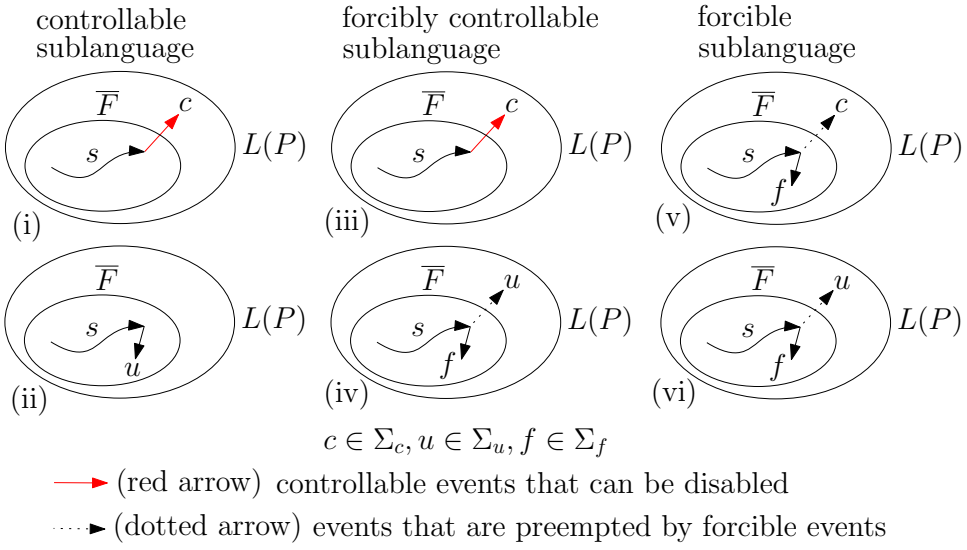


Fig. 2. Comparisons among controllable, forcibly-controllable, and forcible sublanguages.

Theorem 1. Consider a plant P and a specification $\varnothing \neq F \subseteq L_m(P)$. There exists a supervisory control $V : L(P) \to Pwr(\Sigma)$ such that V/P is nonblocking and $L_m(V/P) = F$ iff F is forcibly-controllable.

The result of Theorem 1, as well as its proof below, is a direct generalization of the fundamental result of supervisory control theory [16] by adding the new control mechanism of forcible events. In the proof, for convenience we introduce the notation $E_F(s) = \{\sigma \in \Sigma \mid s\sigma \in \overline{F}\}$, which collects the events that can continue after a string s in $\overline{F}$. If $F \subseteq L_m(P)$, then $E_F(s) \subseteq E_P(s)$.

Proof. Define a supervisory control $V : L(P) \to Pwr(\Sigma)$ according to

$$V(s) = \begin{cases} \Sigma_u \cup \{\sigma \in \Sigma_c \mid s\sigma \in \overline{F}\} & \text{if } \big[E_F(s) \cap \Sigma_f = \varnothing \\ \qquad\qquad \vee\, (\forall \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \Rightarrow s\sigma \in \overline{F}]\big] \\ \{\sigma \in \Sigma_f \mid s\sigma \in \overline{F}\} & \text{if } \big[E_F(s) \cap \Sigma_f \neq \varnothing \\ \qquad\qquad \wedge\, (\exists \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \wedge s\sigma \notin \overline{F}]\big]. \end{cases} \tag{2}$$

Since $E_F(s) \cap \Sigma_f \neq \emptyset$ implies $E_P(s) \cap \Sigma_f \neq \emptyset$, this V is indeed a supervisory control as defined in Equation (1). This supervisory control V will force a forcible event f after string s only when $f \in E_F(s)$ and there is an uncontrollable event u s.t. $su \in L(P) \setminus \overline{F}$ (i.e. controllability fails).

(Sufficiency) We claim that with the above V in (2),

$$L(V/P) = \overline{F}.$$

Before proving this claim, we point out that once this claim is established, the desired conclusion follows immediately:

$$\overline{L_m(V/P)} = \overline{F} = L(V/P) \quad (V/P \text{ is nonblocking})$$
$$L_m(V/P) = L(V/P) \cap F = \overline{F} \cap F = F.$$

Now we prove the claim, by induction on the length of strings. For the base case, since the empty string ϵ belongs to both $L(V/P)$ (by definition) and $\overline{F}$ (since $F \neq \varnothing$), the conclusion holds. Now let $s \in \Sigma^*$ and suppose that

$$s \in L(V/P) \Leftrightarrow s \in \overline{F}.$$

Let $\sigma \in \Sigma$. First suppose $s\sigma \in L(V/P)$; it will be shown $s\sigma \in \overline{F}$. By the definition of $L(V/P)$, we have

$$s \in L(V/P),\ \sigma \in V(s),\ \sigma \in E_P(s).$$

We also have by hypothesis that $s \in \overline{F}$. Now we consider two cases, according to the supervisory control V in Equation (2).

Case 1: $E_F(s) \cap \Sigma_f = \varnothing \vee (\forall \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \Rightarrow s\sigma \in \overline{F}]$.

In this case, if $\sigma \in \Sigma_u$ and $E_F(s) \cap \Sigma_f = \varnothing$, since F is forcibly controllable and $\sigma \in E_P(s)$, we have $s\sigma \in \overline{F}$ (first logic formula in Definition 3). If $\sigma \in \Sigma_u$ and $(\forall \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \Rightarrow s\sigma \in \overline{F}]$, since $\sigma \in E_P(s)$, we again have $s\sigma \in \overline{F}$. Finally if $\sigma \in \Sigma_c$, since $\sigma \in V(s)$, we have $s\sigma \in \overline{F}$ (by $V(s)$ in Equation (2)).

Case 2: $E_F(s) \cap \Sigma_f \neq \varnothing \wedge (\exists \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \wedge s\sigma \notin \overline{F}]$.

In this case, since $\sigma \in V(s)$, we have $\sigma \in \Sigma_f$ and $s\sigma \in \overline{F}$. Therefore in both cases above, we have established $s\sigma \in \overline{F}$.

Conversely suppose $s\sigma \in \overline{F}$; we show $s\sigma \in L(V/P)$. It follows from $s\sigma \in \overline{F}$ that

$$s\sigma \in L(P) \quad (\overline{F} \subseteq L(P))$$
$$s \in \overline{F} \quad (\overline{F} \text{ is closed})$$
$$s \in L(V/P) \quad \text{(hypothesis)}.$$

Hence to show $s\sigma \in L(V/P)$, all we need to show is $\sigma \in V(s)$. Again we consider two cases like above.

Case 1: $E_F(s) \cap \Sigma_f = \varnothing \vee (\forall \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \Rightarrow s\sigma \in \overline{F}]$.

In this case, if $\sigma \in \Sigma_u$, then $\sigma \in V(s)$ (by $V(s)$ in Equation (2)). If $\sigma \in \Sigma_c$, since $s\sigma \in \overline{F}$, it again follows from $V(s)$ in Equation (2) that $\sigma \in V(s)$.

Case 2: $E_F(s) \cap \Sigma_f \neq \varnothing \wedge (\exists \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \wedge s\sigma \notin \overline{F}]$.

In this case, since $s\sigma \in \overline{F}$ and F is forcibly-controllable (in particular the second logic formula in Definition 3), we have $\sigma \in \Sigma_f$. Hence $\sigma \in V(s)$.

Therefore in both cases above, we have established $s\sigma \in V(s)$, and thereby $s\sigma \in L(V/P)$ is proved. This finishes the induction step, and the proof of sufficiency is now complete.

(Necessity) Suppose that $\overline{L_m(V/P)} = L(V/P)$ and $L_m(V/P) = F$. It will be shown that F is forcibly-controllable. By the above assumption, we have $L(V/P) = \overline{L_m(V/P)} = \overline{F}$. Let $s \in \overline{F}$. Then $s \in L(V/P)$. Below we consider two cases (as done in the sufficiency proof).

Case 1: $E_F(s) \cap \Sigma_f = \varnothing \vee (\forall \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \Rightarrow s\sigma \in \overline{F}]$.

If $E_P(s) \cap \Sigma_f = \varnothing$, let $\sigma \in \Sigma_u$ and $\sigma \in E_P(s)$. Then $\sigma \in V(s)$ and $s\sigma \in L(P)$. It follows from the definition of $L(V/P)$ that $s\sigma \in L(V/P)$. Hence $s\sigma \in \overline{F}$. This shows that F is controllable, and therefore is forcibly-controllable. On the other hand, if $(\forall \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \Rightarrow s\sigma \in \overline{F}]$, by definition F is forcibly-controllable.

Case 2: $E_F(s) \cap \Sigma_f \neq \varnothing \wedge (\exists \sigma \in \Sigma_u)\ [\sigma \in E_P(s) \wedge s\sigma \notin \overline{F}]$.

In this case, since $E_F(s) \cap \Sigma_f \neq \varnothing$, there exists $f \in E_P(s) \cap \Sigma_f$. This means that $(\exists f \in \Sigma_f) sf \in \overline{F}$. Moreover, let $\sigma \in \Sigma \setminus \Sigma_f$. Then by definition of $V(s)$ in (2), we have $\sigma \notin V(s)$. So $s\sigma \notin L(V/P) = \overline{F}$. This proves $(\forall \sigma \in \Sigma \setminus \Sigma_f) s\sigma \notin \overline{F}$. Therefore by Definition 3, F is forcibly-controllable. □

Theorem 1 provides a necessary and sufficient condition for the existence of a nonblocking supervisory control V that realizes an imposed specification F, as required in Problem 1. This supervisory control V may be implemented by a finite automaton S in the same way as [1, section 3.6], such that the synchronous product of plant P and S is nonblocking and represents the specification F. This automaton S is called a *supervisor*.

IV. Supremal Forcibly-Controllable Sublanguages

In the preceding section, we know from Theorem 1 that Problem 1 is solvable whenever the specification $F \subseteq L_m(P)$ is forcibly-controllable. In this section, we inquire: *what if F is not forcibly-controllable? In this case, does there exist an 'optimal' supervisor in a sense of maximal permissiveness?*

Let $F \subseteq L_m(P)$. Whether or not F is forcibly controllable, write the family

$$\mathcal{F}(F) := \{K \subseteq F \mid K \text{ is forcibly-controllable w.r.t. } P\}. \tag{3}$$

In words, $\mathcal{F}(F)$ is the collection of all subsets of F that are forcibly-controllable. Note that $\mathcal{F}(F)$ is nonempty because the empty language $\emptyset$ belongs to it (see **Fact 2** above). Moreover, the next result asserts that $\mathcal{F}(F)$ is closed under set unions.

Proposition 1. Consider $\mathcal{F}(F)$ in (3). If $K_1, K_2 \in \mathcal{F}(F)$, then $K_1 \cup K_2 \in \mathcal{F}(F)$.

Proof. Let K_1 and K_2 be forcibly-controllable sublanguages of F w.r.t. P. First $K_1 \cup K_2$ is evidently a sublanguage of F. Now let $s \in \overline{K_1 \cup K_2}$. We need to show that $(\forall \sigma \in \Sigma_u)\ [s\sigma \in L(P) \implies s\sigma \in \overline{K_1 \cup K_2}] \vee ((\exists f \in \Sigma_f)\ [sf \in \overline{K_1 \cup K_2}] \wedge (\forall \sigma \in \Sigma \setminus \Sigma_f)\ [s\sigma \notin \overline{K_1 \cup K_2}])$. This is equivalent to show that if

$$\neg(\forall \sigma \in \Sigma_u)\ [s\sigma \in L(P) \implies s\sigma \in \overline{K_1 \cup K_2}] \tag{4}$$

then $(\exists f \in \Sigma_f)\ [sf \in \overline{K_1 \cup K_2}] \wedge (\forall \sigma \in \Sigma \setminus \Sigma_f)\ [s\sigma \notin \overline{K_1 \cup K_2}]$.

Suppose that (4) holds. Since $s \in \overline{K_1 \cup K_2}$, we have $s \in \overline{K_1}$ or $s \in \overline{K_2}$. In the following we consider the case $s \in \overline{K_1}$; the other case $s \in \overline{K_2}$ is analogous. Since $s \in \overline{K_1}$ and K_1 is forcibly-controllable, we have

- $(\forall \sigma \in \Sigma_u)\ [s\sigma \in L(P) \implies s\sigma \in \overline{K_1}]$, or
- $(\exists f \in \Sigma_f)\ [sf \in \overline{K_1}] \wedge (\forall \sigma \in \Sigma \setminus \Sigma_f)\ [s\sigma \notin \overline{K_1}]$.

The first case is impossible due to (4). Thus we only need to consider the second case. It follows from $(\exists f \in \Sigma_f)\ [sf \in \overline{K_1}]$ that $(\exists f \in \Sigma_f)\ [sf \in \overline{K_1 \cup K_2}]$. Now let $\sigma \in \Sigma \setminus \Sigma_f$. Thus $s\sigma \notin \overline{K_1}$. To show that $s\sigma \notin \overline{K_1 \cup K_2}$, we must prove $s\sigma \notin \overline{K_2}$. Suppose on the contrary that $s\sigma \in \overline{K_2}$; then $s \in \overline{K_2}$. Since K_2 is also forcibly-controllable, we have

- $(\forall \sigma \in \Sigma_u)\ [s\sigma \in L(P) \implies s\sigma \in \overline{K_2}]$, or
- $(\exists f \in \Sigma_f)\ [sf \in \overline{K_2}] \wedge (\forall \sigma \in \Sigma \setminus \Sigma_f)\ [s\sigma \notin \overline{K_2}]$.

The first case is again impossible due to (4). From the second case, we derive $s\sigma \notin \overline{K_2}$, which directly contradicts the assumption that $s\sigma \in \overline{K_2}$. Therefore $s\sigma \notin \overline{K_2}$ after all, and consequently $s\sigma \notin \overline{K_1 \cup K_2}$. This establishes that $K_1 \cup K_2$ is forcibly-controllable. □

Based on Proposition 1, $\mathcal{F}(F)$ in (3) is closed under set unions, and therefore contains a unique supremal element which is the union of all members in $\mathcal{F}(F)$:

$$\sup \mathcal{F}(F) := \bigcup \{K \mid K \in \mathcal{F}(F)\}. \tag{5}$$

This $\sup \mathcal{F}(F)$ is the largest forcibly-controllable sublanguage of $F \subseteq L_m(P)$. If F was already forcibly-controllable, then $\sup \mathcal{F}(F) = F$.

Theorem 2. Consider a plant P and a specification $\varnothing \neq F \subseteq L_m(P)$. Let $F_{\sup} := \sup \mathcal{F}(F)$. If $F_{\sup} \neq \emptyset$, then there exists a supervisory control $V_{\sup} : L(P) \to Pwr(\Sigma)$ such that $V_{\sup}/P$ is nonblocking and $L_m(V_{\sup}/P) = F$.

Proof. Since $F_{\sup}$ is forcibly-controllable and nonempty, the conclusion follows from Theorem 1. □

Theorem 2 generalizes the classical result in [17] with the forcing mechanism incorporated. The supervisory control $V_{\sup}$ in Theorem 2 realizing the supremal forcibly-controllable sublanguage of F may also be implemented by a finite automaton, or a supervisor (cf. discussion at the end of Section III). Owing to 'supremum' of the realized language $F_{\sup}$, we call this supervisor the *maximally permissive supervisor*. In the next section, we present an algorithm to synthesize such maximally permissive supervisors.

Remark 1. We underline an important point about 'maximal permissiveness'. The 'maximally permissive supervisor' we just mentioned has maximal permissiveness in two meanings. For the first (obvious) one, the language realized by this supervisor is the supremal forcibly-controllable sublanguage $F_{\sup}$. The second meaning of maximal permissiveness is that this supervisor permits 'maximal freedom' of choosing forcible events for preempting. Indeed, whenever preempting is needed, all forcible events that can keep the supremal $F_{\sup}$ invariant are available in this suprevisor to be chosen. Having

said the above, we point out that in implementation of forcing control, eventually only one forcible event (possibly among multiple valid choices) is chosen. In other words, one can only force one forcible event at a time, and in this sense 'maximal permissiveness' is not possessed by our supervisor. In fact in this case, a maximally permissive supervisor generally does not exist. For theoretic soundness and correspondence with the conventional supervisory control, we adopt the term 'maximally permissive supervisor' as that realizes the supremal forcibly-controllable sublanguage $F_{\sup}$.

Before ending this section, we collect several facts on forcibly-controllable and forcible sublanguages. Not all of them are needed in the sequel, but they are interesting in their own right.

Fact 3. Let plant P be a finite automaton over $\Sigma = \Sigma_c \cup \Sigma_u$ and a forcible event set $\Sigma_f \subseteq \Sigma$. Also let $F \subseteq L_m(P)$.

1) The union of forcibly-controllable sublanguages of F is also a forcibly-controllable sublanguage of F.
2) The union of forcible sublanguages of F is also a forcible sublanguage of F.
3) The intersection of forcibly-controllable sublanguages of F is not necessarily a forcibly-controllable sublanguage of F.
4) The intersection of forcible sublanguages of F is not necessarily a forcible sublanguage of F.

The first fact above is Proposition 1, while the second about forcible sublanguages is a straightforward corollary of Proposition 1. The third and fourth facts are illustrated by Example 1 below. These facts together indicate (in terms of algebraic structures) that the set of all forcibly-controllable (or forcible) sublanguages of a given language F is a *complete upper semilattice* with the set union operation of the complete lattice of all sublanguages of F.

Example 1. Consider plant P with $L(P) = L_m(P) = \{\varepsilon, f_1, f_2, u\}$ and $F = \{\varepsilon, f_1, f_2\}$; here f_1, f_2 are forcible and u is uncontrollable. Also consider two sublanguages $K_1 = \{\varepsilon, f_1\}$ and $K_2 = \{\varepsilon, f_2\}$ of F. Now, it can be verified that both K_1 and K_2 are forcibly-controllable and forcible (without controllable events, forcible-controllability and forcibility are equivalent: **Fact 2**). However, $K_1 \cap K_2 = \{\varepsilon\}$ is neither forcibly-controllable nor forcible (due to the absence of forcible events and the presence of the uncontrollable event u enabled after ε in the plant).

V. Synthesizing the maximally permissive forcibly-controllable nonblocking supervisor

In this section we present an algorithm to compute maximally permissive supervisors introduced in the preceding section (see in particular Theorem 2 and the paragraph following Theorem 2).

So far we have considered the setting where a plant automaton P and a specification language $F \subseteq L_m(P)$ are given. Now assume that F is a regular sublanguage, so F can be represented by a finite automaton. It is well-known that the conventional synthesis problem with a regular specification language is easily transformed into a synthesis problem where only a plant automaton is considered by applying a so-called *plantification transformation* on the finite automaton representing the specification [18]. This leads to the simplified (but equivalent) problem of supervisory control synthesis below.

Problem 2 (Synthesis of maximally permissive forcibly-controllable nonblocking supervisor)**.** *Given a plant automaton $P = (Q, \Sigma, \delta, q_0, Q_m)$ with sets $\Sigma_u \subseteq \Sigma$ of uncontrollable events and $\Sigma_f \subseteq \Sigma$ of forcible events, synthesize a maximally permissive, forcibly-controllable, nonblocking supervisor S for P.*

In Algorithm 1 below, a maximally permissive, forcibly-controllabe, nonblocking supervisor is computed starting from the plant automaton (as is common in SCT). Its structure is based on the well-known algorithm for synthesis of maximally permissive supervisors for EFA (without using variables though) [19]. Besides sets of nonblocking and bad states, in this case also sets of states F_k and F_k^j where forcing is applied are maintained.

Controllable sublanguages have the transitivity property, i.e., whenever F_1 is a controllable sublanguage of F_2 and F_2 is a controllable sublanguage of F_3, then also F_1 is a controllable sublanguage of F_3. A similar useful property does not hold for forcibly-controllable sublanguages. Consider the languages $F_1 = \{\varepsilon\}$, $F_2 = \{f\}$, and $F_3 = \{f, u\}$ where $f \in \Sigma_f$ and $u \in \Sigma_u$. It is easily verified that F_1 is a forcibly-controllable sublanguage of F_2 and F_2 is a forcibly-controllable sublanguage of F_3. However F_1 is not a forcibly-controllable sublanguage of F_3.

The reason for the absence of this property is the situation that a forcible event (f) that has been used to preempt an uncontrollable event (u) is later disabled (which means that the uncontrollable event in hindsight cannot be preempted and thus has to be maintained.

This difference between traditional SCT and SCT with forcible events results in the need to maintain a set of states from which forcing (by a forcible event) has been used to preempt an uncontrollable transition to a bad state. As soon as all forcible events from such a forcing state need to be disabled (because they all lead to a bad state) the forcing state itself becomes bad (Line 17).

Theorem 3 (Termination)**.** Algorithm 1 terminates.

Proof. First, consider the iteration represented by the repeat-until in Lines 8 - 11. In each execution of the body of this repeat-until either a state is added to a set NB_{k+1}^{l+1} (compared to the set of nonblocking states computed in the previous execution of that body NB_{k+1}^{l}), or the set NB_{k+1}^{l+1} is the same as the set NB_{k+1}^{l}. The latter leads to termination, and the former can only take place finitely often since $NB_{k+1}^{l} \subseteq Q_k \subseteq Q$ for all k and l and the set of states Q is finite.

Second, consider the iteration represented by the repeat-until in Lines 16 - 20. Again, as in the previous case, each execution of the body of the repeat-until results in adding at least one state to the set B_{k+1}^{j+1} (compared to the 'previous' set B_{k+1}^{j}) or results in the same set of bad states. The former

Algorithm 1 Supervisory controller synthesis for maximally permissive, forcibly-controllable, nonblocking supervisor

Input: plant $P = (Q, \Sigma, \delta, q_0, Q_m)$, uncontrollable events Σ_u, forcible events Σ_f

Output: S is maximally permissive, forcibly-controllable, nonblocking supervisor for P if it exists, and empty supervisor otherwise

1: $Q_0 \leftarrow Q$
2: $F_0 \leftarrow \varnothing$
3: $\delta_0 \leftarrow \delta$ $\quad S_0 \leftarrow (Q_0, \Sigma, \delta_0, q_0, Q_0 \cap Q_m)$
4: $k \leftarrow 0$
5: **repeat**
6: $\quad NB^0_{k+1} \leftarrow Q_m \cap Q_k$
7: $\quad l \leftarrow 0$
8: $\quad$ **repeat**
9: $\quad\quad NB^{l+1}_{k+1} \leftarrow NB^l_{k+1} \cup \{q \in Q_k \mid (\exists \sigma \in \Sigma)\ [\delta_k(q, \sigma) \in NB^l_{k+1}]\}$
10: $\quad\quad l \leftarrow l + 1$
11: $\quad$ **until** $NB^l_{k+1} = NB^{l-1}_{k+1}$
12: $\quad NB_{k+1} \leftarrow NB^l_{k+1}$
13: $\quad B^0_{k+1} \leftarrow Q_k \setminus NB_{k+1}$
14: $\quad F^0_{k+1} \leftarrow F_k \cap Q_k$
15: $\quad j \leftarrow 0$
16: $\quad$ **repeat**
17: $\quad\quad B^{j+1}_{k+1} \leftarrow B^j_{k+1} \cup \left\{ q \in Q_k \middle| \begin{array}{l} (\exists u \in \Sigma_u)\ [\delta_k(q, u) \in B^j_{k+1}] \\ \wedge (\forall f \in \Sigma_f)\ [\delta_k(q, f) \in B^j_{k+1}] \end{array} \right\}$
18: $\quad\quad F^{j+1}_{k+1} \leftarrow F^j_{k+1} \cup \left\{ q \in Q_k \middle| \begin{array}{l} (\exists u \in \Sigma_u)\ [\delta_k(q, u) \in B^j_{k+1}] \wedge \\ (\exists f \in \Sigma_f)\ [\delta_k(q, f) \notin B^j_{k+1}] \end{array} \right\}$
19: $\quad\quad j \leftarrow j + 1$
20: $\quad$ **until** $B^j_{k+1} = B^{j-1}_{k+1} \ \wedge\ (\forall q \in F^j_{k+1} \setminus B^j_{k+1})\ [(\exists f \in \Sigma_f)\ [\delta_k(q, f) \notin B^j_{k+1}]]$
21: $\quad B_{k+1} \leftarrow B^j_{k+1}$
22: $\quad F_{k+1} \leftarrow F^j_{k+1}$
23: $\quad Q_{k+1} \leftarrow Q_k \setminus B_{k+1}$
24: $\quad \delta_{k+1} \leftarrow (\delta_k \cap (Q_{k+1} \times \Sigma \times Q_{k+1})) \setminus (F_{k+1} \times \Sigma \setminus \Sigma_f \times Q_{k+1})$ $\quad S_{k+1} \leftarrow (Q_{k+1}, \Sigma, \delta_{k+1}, q_0, Q_m \cap Q_{k+1})$
25: $\quad k \leftarrow k + 1$
26: **until** $Q_k = Q_{k-1} \wedge \delta_k = \delta_{k-1}$

situation can only take place finitely often as $B^j_{k+1} \subseteq Q_k \subseteq Q$ for all k and j. The latter situation leads to termination in case the other condition of the guard in Line 20 is satisfied. If that condition is not satisfied, then $(\exists q \in F^j_{k+1} \setminus B^j_{k+1})\ [(\forall f \in \Sigma_f)\ [\delta_k(q, f) \in B^j_{k+1}]]$. But then, in the next execution of the body of the repeat-until, at least one state is added to the set of bad states of that iteration.

Finally, consider the outermost repeat-until construction (Lines 5 - 26). Invariantly it is the case that $Q_{k+1} \subseteq Q_k$ and $\delta_{k+1} \subseteq \delta_k$ for all k. Now, given the computations of Q_{k+1} in Line 23 and δ_{k+1} in Line 24, there are three cases:

- $Q_{k+1} \subset Q_k$
- $\delta_{k+1} \subset \delta_k$
- $Q_{k+1} = Q_k$ and $\delta_{k+1} = \delta_k$.

Therefore, either the repeat-until terminates because the first two cases can only be repeated finitely often (there is only a finite number of states and a finite number of transitions in the input plant), or the termination condition of the repeat-until becomes true and the repeat-until terminates. □

Theorem 4 (Correctness)**.** Algorithm 1 computes a maximally permissive, forcibly-controllable, nonblocking supervisor.

Proof. For presenting the proof (of forcible-controllability) we introduce thought variables $S_k = (Q_k, \Sigma, \delta_k, q_0, Q_k \cap Q_m)$ which represent the subautomata of P derived after iteration k of the outer iteration. See the green lines of code and note that the variables S_k are never used to update any other variable, nor to evaluate a condition in the control flow of the pseudo code. Consequently, these lines have no impact on the outcome of the algorithm and are only used for facilitating the proof.

Nonblocking: Upon termination of Algorithm 1, say after K iterations, we have $Q_K = Q_{K-1}$ and $\delta_K = \delta_{K-1}$. This can only be the case if $B_K = \varnothing$ and therefore only if $NB_K = Q_K$. So, if we prove that $NB_K = \{q \in Q_{K-1} \mid q \text{ is nonblocking } \}$ then we are done, since $Q_{K-1} = Q_K$. The pseudo code computing the sets of nonblocking states is fairly standard and it has been shown before that indeed the set of nonblocking states is computed. See for example [19] or [20] for a statement to this effect.

Forcible-controllability: We will prove that the output of the algorithm is forcibly-controllable w.r.t. the input plant P. First, we prove the properties presented in Equations 6 and 7

below.

$$\begin{aligned}(\forall 0 \leq k \leq K,\ \forall q \in F_k \setminus B_k)\\ [(\exists f \in \Sigma_f)\ [\delta_k(q,f) \in Q_k]\\ \wedge\\ (\forall n \in \Sigma \setminus \Sigma_f)\ [\neg\delta_k(q,n)!]].\end{aligned} \tag{6}$$

The proof of the property in Equation 6 is by induction on k. For the base case $k = 0$ this follows immediately from the fact that $F_0 = \varnothing$ (Line 2). For the induction case, assume that $(\forall q \in F_k \setminus B_k)\ [(\exists f \in \Sigma_f)\ [\delta_k(q,f) \in Q_k] \wedge (\forall n \in \Sigma \setminus \Sigma_f)\ [\neg\delta_k(q,n)!]]$. Let $q \in F_{k+1} \setminus B_{k+1}$. Then, $q \in F^J_{k+1}$ and $q \notin B^J_{k+1}$ for the J that corresponds with the value of j upon termination of this iteration of the repeat-until statement in Lines 16 - 20. Then, also $(\forall q \in F^J_{k+1} \setminus B^J_{k+1})\ [(\exists f \in \Sigma_f)\ [\delta_k(q,f) \notin B^J_{k+1}]]$ as this is part of the termination condition of the repeat-until. Then, as $q \in F^J_{k+1} \setminus B^J_{k+1}$, we have $(\exists f \in \Sigma_f)\ [\delta_k(q,f) \notin B^J_{k+1}]$. Then also, following Line 23, $(\exists f \in \Sigma_f)\ [\delta_k(q,f) \in Q_{k+1}]$. Let $n \in \Sigma \setminus \Sigma_f$. As $q \in F_{k+1}$, from Line 24, it follows that $\neg\delta_{k+1}(q,n)!$. Consequently, Equation 6 holds.

$$\begin{aligned}(\forall 0 \leq k \leq K,\ \forall q \in Q_k \setminus F_k,\ \forall u \in \Sigma_u)\\ [\delta_0(q,u) \in Q_0 \implies \delta_k(q,u) \in Q_k]\end{aligned} \tag{7}$$

The proof of this property is by induction on k. The base case $k = 0$ follows trivially. For the induction step, assume that $(\forall q \in Q_k \setminus F_k,\ \forall u \in \Sigma_u)\ [\delta_0(q,u) \in Q_0 \implies \delta_k(q,u) \in Q_k]$. Let $q \in Q_{k+1} \setminus F_{k+1}$ and $u \in \Sigma_u$. Assume that $\delta_0(q,u) \in Q_0$. As $q \notin F_{k+1}$, it follows that $\delta_{k+1}(q,u) = \delta_k(q,u)$ provided that $\delta_{k+1}(q,u) \in Q_{k+1}$. Towards a contradiction assume that $\delta_k(q,u) \in Q_k \setminus Q_{k+1}$. Then $\delta_k(q,u) \in B_{k+1}$. Then $\delta_k(q,u) \in B^J_{k+1}$ and because of the termination condition of the repeat-until, also $\delta_k(q,u) \in B^{J-1}_{k+1}$. But then, either $q \in B^J_{k+1}$ or $q \in F^J_{k+1}$. This contradicts the assumption that $q \in Q_{k+1} \setminus F_{k+1}$.

Together the properties in Equation 6 and Equation 7, show that each S_k as produced during the execution of the algorithm is forcibly-controllable w.r.t. P. Each of the states of S_k, i.e., the set Q_k, either respects controllability (Equation 7 holds for the states from $Q_k \setminus F_k$) or is forcing and respects Equation 6 (for states from F_k that are in Q_k).

Maximally permissive: Suppose there is a larger nonblocking, forcibly-controllable subautomaton S' for P. Then, it contains a state that S does not, or, if the former does not hold (and both S' and S have the same set of states), it contains a transition between states they both have that S does not have. Maximal permissiveness is achieved by construction. In the algorithm, states or transitions are only removed if there is necessity for that.

Let us consider the former case. S' contains a state that is not present in S. Then this state is removed (in the computation of S) in some invocation of Line 23. Thus this state is in B_{k+1}, i.e., a bad state. We need to show that any addition of a state to the set of bad states cannot be avoided since otherwise the resulting supervisor violates nonblockingness or forcible-controllability. Proof of this follows the same reasoning as in the underlying algorithm used for synthesis of EFA [19].

Now, let us consider the latter case. A transition is present in S' but not in S. This has to be due to Line 24. Therefor this concerns a transition from a forcing state labelled by an nonforcible event. Since the source state is forcing (i.e. if it does not exercise its forcing action it is deemed to be bad) we know that this transition would violate forcible-controllability. □

Theorem 5 (Complexity)**.** Algorithm 1 has worst-case time complexity $\mathcal{O}(|Q|^2\ |\Sigma|)$.

Proof. The black part of the algorithm is basically the standard algorithm for supervisory control synthesis without event forcing and it has been established to have a worst-case time complexity of $\mathcal{O}(|Q|^2\ |\Sigma|)$ in [21]. Obviously, the red additions (for the purpose of forcing supervisory control) do not negatively, nor positively, affect this worst-case time complexity. □

Remark 2. When Algorithm 1 is applied with $\Sigma_f = \varnothing$, it in fact computes the maximally permissive, controllable, nonblocking supervisor. All sets of states where forcing is applied are empty and hence the red additions to the bad state computation do not add anything. Moreover, there is no need to change the transition relation (and the subscript can be removed).

Remark 3. Replacing all occurrences of the set Σ_u in Algorithm 1 (in Lines 17 and 18) by Σ results in an algorithm for the maximally permissive, forcible, nonblocking supervisor.

VI. Case study - small manufacturing line

We consider the example of the small manufacturing line that was also used in [1, section 3.8]. There are two machines (M_1 and M_2) and one (plantified) specification (namely that finishing processing on machine M_1 and starting processing on machine M_2 alternate). See Fig. 3 for the graphically represented automata. The specification has been formulated in such a way that it is controllable (and hence forcibly-controllable as well). For now, we consider only the event $start\_M_2$ representing the start of processing on machine M_2 as forcible.

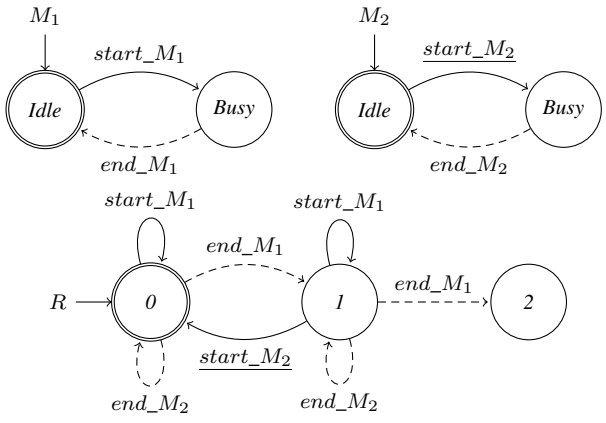


Fig. 3. Automata for the machines and the specification in the small manufacturing line.

Remark 4. The self-loops in the specification automaton are not necessary. We have added them in this case, because as a consequence the number of states in the synchronous product is smaller and that facilitates graphical representation of that synchronous product.

Fig. 4 presents a number of relevant automata:

- the synchronous product of the three automata from Fig. 3 by considering all states and transitions regardless their color. This automaton is the input for the synthesis procedure.
- the traditional supervisor (no forcing at all), is obtained by considering only the black colored states and transitions. The states $BI1$ and $BB1$ have been made unreachable by disabling the incoming $start\_M_1$ events.
- the forcing supervisor in the case that only event $start\_M_2$ is forcible is obtained by considering the black and blue states and transitions. The red and cyan states and transitions are omitted by synthesis. Note that this is the same result as predicted in [1, Section 3.8].
- the forcing supervisor in the case that also event end_M_2 is forcible is given by all but the red states and transitions.

Both the synchronous product and the traditional supervisor have been obtained by applying the Compositional Interchange Format (CIF) [22], [23]. CIF is part of the Eclipse Supervisory Control Engineering Toolkit (ESCET™) project.[1] Both forcing supervisors have been computed manually following Algorithm 1.

This example suggests, and our intuitions support this observation, that using additional forcible events generally results in more permissive supervisors.

VII. Case Study - Small Factory

In this section, both the traditional supervisor and forcing supervisor (for all controllable events) are provided for the small factory as discussed in [1]. The plant consists of two machines, namely M_1 and M_2 (see Fig. 5 for the automata representing the plant components). These machines can start and end processing (by means of controllable events $start\_M_i$, for $i = 1, 2$), but can also break down (uncontrollably with events $break\_M_i$, for $i = 1, 2$) when in the working state (W). In case of a breakdown, a repair can be achieved (controllable events $repair\_M_i$, for $i = 1, 2$) upon which the system starts again in the idle state (I). The supervisors developed are supposed to achieve the requirements given by means of the controllable automata specifications[2] given in Fig. 6[3]. The first requirement captures that the two machines are placed in line with a single-space buffer in between. Note that this buffer is not modeled explicitly. The second requirement states that in case both machines break down, priority has to be given to repairing machine M_2.

[1]See http://eclipse.org/escet. 'Eclipse', 'Eclipse ESCET' and 'ESCET' are trademarks of Eclipse Foundation, Inc.

[2]A controllable automaton is one whose marked language is controllable w.r.t. the plant.

[3]The models for the plant components and requirements are based on those provided in [1] with easier to interpret names for events and where the automata specifications are made controllable (where needed).

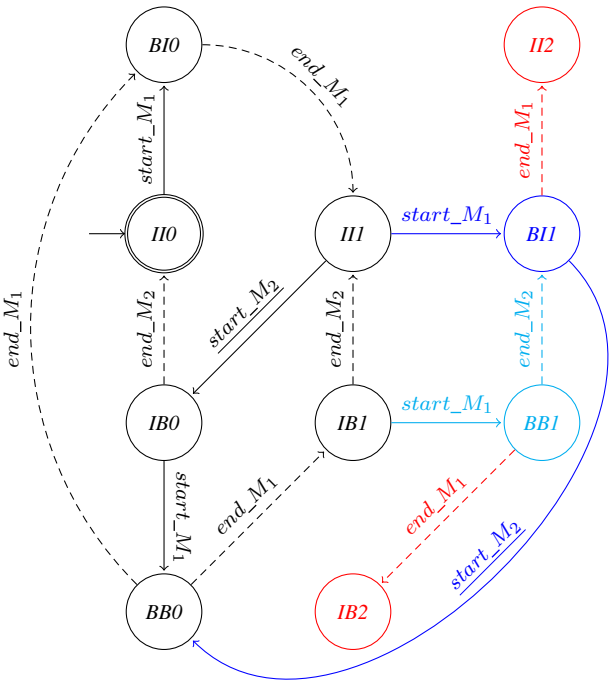


Fig. 4. Automaton representing the synchronous product and, using colors as described in the text, three different supervisors.

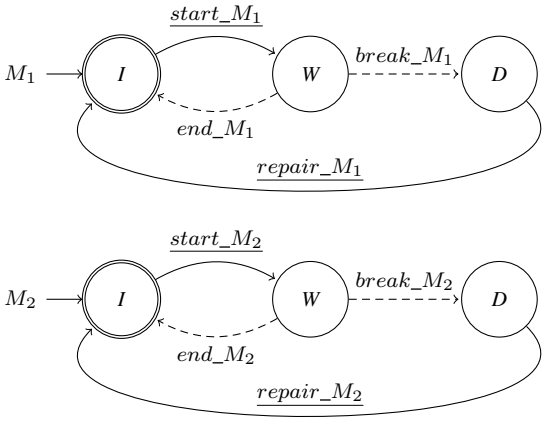


Fig. 5. Automata representing the machines M_1 and M_2 in the small factory.

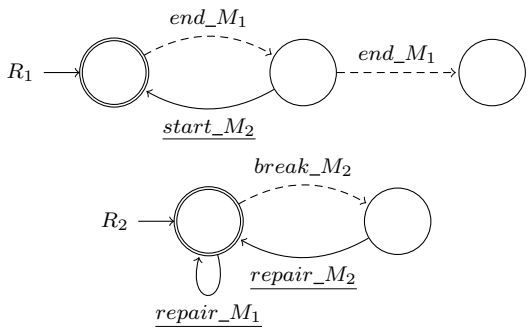


Fig. 6. Controllable specification automata for the small factory.

Application of supervisory control synthesis Algorithm 1 to this system (consisting of plant components and controllable requirements) where all (and only) the controllable events are considered forcible) results in a supervisor that can be represented by the automaton in Fig. 7.

In the figure, the green states are those where forcing of an event is used to prevent reaching a state that is unsafe via an uncontrollable event. In the conventional supervisor those states are made unreachable by preventing their incoming controllable events (the red transitions labelled by the event $start\_M_1$). So in the nonforcing supervisor, the green state and the red transitions are absent. Thus the forcing supervisor indeed allows behaviors that are prohibited by the nonforcing supervisor.

As can be seen from the figure, the forcing is achieved by using the forcible evens $start\_M_2$ and $repair\_M_2$. Therefore, in case the controllable events involving machine M_1 are not forcible, the same forcing supervisor results.

VIII. Conclusions

In this paper, the setting of supervisory control theory has been enriched with the possibility for a supervisor to force events in order to preempt uncontrollable events from leading the plant into undesired states. This has required an adaptation of the notion of controllability to forcible-controllability. This notion and its properties have been studied in detail and a supervisory control problem for obtaining a maximally permissive, foricbly-controllable, nonblocking supervisory controller has been posed and solved. The approach has been illustrated with two case studies from literature.

Our future work is twofold. In one direction, we aim to extend this framework with forcing to tackle other fundamental problems in supervisory control; these include partial observation [24], timed systems [7], and supervisor localization [25]. In the other direction, with the forcing mechanism which is the principal control mechanism used in continuous-time control theory, we are interested in establishing connections as well as contrasts between the two types of control theory; special attention will be given to several basic control-theoretic ideas including reachability, stability, and control-barriers [26].

References

[1] W. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*. Springer, 2019.

[2] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[3] P. Malik, "Generating controllers from discrete-event models," in *Proceedings of the Summer School in Modelling and Verification of Parallel processes*, 2002, pp. 337–242.

[4] F. Reijnen, A. Hofkamp, J. van de Mortel-Fronczak, M. Reniers, and J. Rooda, "Finite response and confluence of state-based supervisory controllers," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 2019, pp. 509–516.

[5] F. F. H. Reijnen, T. R. Erens, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory controller synthesis and implementation for safety plcs," *Discrete Event Dynamic Systems*, vol. 32, p. 115–141, 2022).

[6] C. H. Golaszewski and P. J. Ramadge, "Control of discrete event processes with forced events," *26th IEEE Conference on Decision and Control*, vol. 26, pp. 247–251, 1987.

[7] B. A. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Transactions on Automatic Control*, vol. 39, no. 2, pp. 329–342, 1994.

[8] R. Zhang, K. Cai, Y. Gan, Z. Wang, and W. Wonham, "Supervision localization of timed discrete-event systems," *Automatica*, vol. 49, no. 9, pp. 2786–2794, 2013.

[9] S. Takai and T. Ushio, "A new class of supervisors for timed discrete event systems," *Discrete Event Dynamic Systems*, vol. 16, no. 2, pp. 257–278, 2006.

[10] A. Rashidinejad, M. Reniers, and L. Feng, "Supervisory control of timed discrete-event systems subject to communication delays and non-fifo observations," *IFAC-PapersOnLine*, vol. 51, no. 7, pp. 456–463, 2018.

[11] A. Rashidinejad, P. van der Graaf, and M. Reniers, "Nonblocking supervisory control synthesis of timed automata using abstractions and forcible events," in *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2020, pp. 1–8.

[12] A. Rashidinejad, P. van der Graaf, M. Reniers, and M. Fabian, "Non-blocking supervisory control of timed automata using forcible events," *IFAC-PapersOnLine*, vol. 53, no. 4, pp. 356–362, 2020, 15th IFAC Workshop on Discrete Event Systems WODES 2020 — Rio de Janeiro, Brazil, 11-13 November 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896321000756

[13] T. Ushio and S. Takai, "Control-invariance of hybrid systems with forcible events," *Automatica*, vol. 41, no. 4, pp. 669–675, 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109804003097

[14] J. Huang and R. Kumar, "Directed control of discrete event systems for safety and nonblocking," *IEEE Transactions on Automation Science and Engineering*, vol. 5, no. 4, pp. 620–629, 2008.

[15] S. Balemi, G. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE Transactions on Automatic Control*, vol. 38, no. 7, pp. 1040–1059, 1993.

[16] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[17] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM Journal on Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.

[18] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discrete Event Dynamic Systems*, vol. 17, no. 4, pp. 475–504, 2007.

[19] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.

[20] S. Thuijsman and M. Reniers, "Transformational supervisor synthesis for evolving systems," *IFAC-PapersOnLine*, vol. 53, no. 4, pp. 309–316, 2020.

[21] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[22] D. van Beek, W. Fokkink, D. Hendriks, A. Hofkamp, J. Markovski, J. van de Mortel-Fronczak, and M. Reniers, "CIF 3: Model-based engineering of supervisory controllers," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 575–580.

[23] W. Fokkink, M. Goorden, D. Hendriks, B. van Beek, A. Hofkamp, F. Reijnen, P. Etman, L. Moormann, A. van de Mortel-Fronczak, M. Reniers, K. Rooda, B. van der Sanden, R. Schiffelers, S. Thuijsman, J. Verbakel, and H. Vogel, "Eclipse ESCET™: The Eclipse Supervisory Control Engineering Toolkit," in *TACAS 2023*, 2023).

[24] F. Lin and W. Wonham, "On observability of discrete-event systems," *Information Science*, vol. 44, no. 3, pp. 173–198, 1988.

[25] K. Cai and W. Wonham, *Supervisor Localization: A Top-Down Approach to Distributed Control of Discrete-Event Systems*. Springer, 2016.

[26] A. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: theory and applications," in *2019 18th European Control Conference*, 2019, pp. 3420–3431.

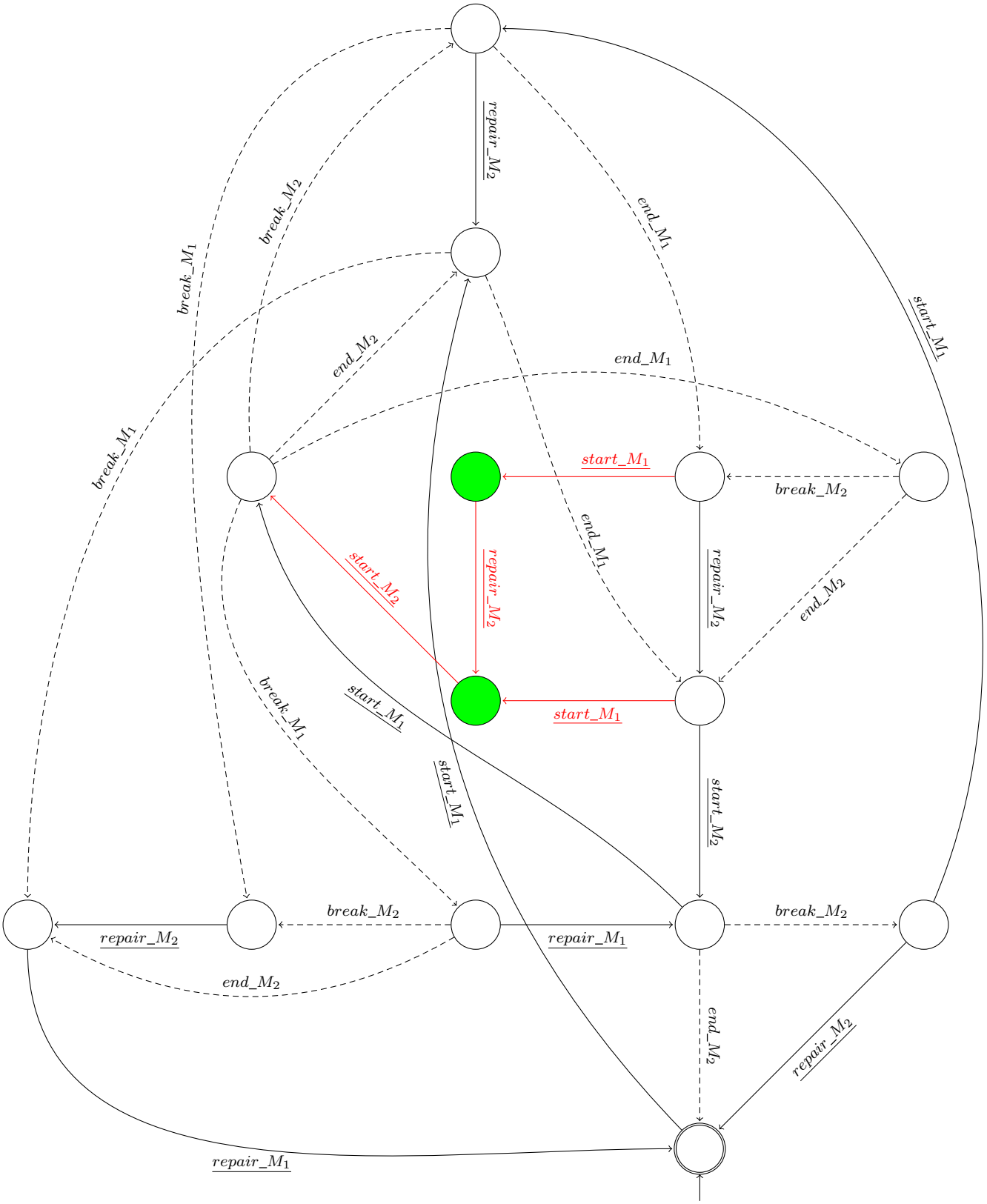
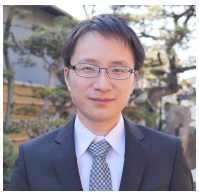


Fig. 7. Forcing supervisor for the small factory.

Michel Reniers (SM'17) is currently an Associate Professor in model-based engineering of supervisory control at the Department of Mechanical Engineering at TU/e. He received both his MSc and PhD degrees in computer science form Eindhoven University of Technology in 1995 and 1999 respectively. Previously he was a postdoc at CWI (National research institute for mathematics and computer science in the Netherlands), and an Assistant Professor at both the Department of Mathematics and Computer Science and the Department of Mechanical Engineering of Eindhoven University of Technology. Dr. Reniers is coauthor of the book *Process algebra : equational theories of communicating processes* published by Cambridge Univerity Press in 2010. He published over 150 journal and conference papers. His research portfolio ranges from model-based systems engineering and model-based validation and testing to novel approaches for supervisory control synthesis and discrete-event systems. Applications of this work are mostly in the areas of cyber-physical systems and manufacturing systems.

Dr. Reniers is currently acting as an Associate Editor for *Automata*, an Associate Editor for *Discrete Event Dynamic Systems* and an Associate Editor for the *IEEE Open Journal of Control Systems*. He is also member the *Technology Conference Editorial Board of the IEEE Control Systems Society*. He received the best paper award of ACSD 2018 and in 2022 the *Norbert Giambiasi Award for Conceptual Modeling Excellence*. (photograph by Angeline Swinkels)



Kai Cai (S'08-M'12-SM'17) received the B.Eng. degree in Electrical Engineering from Zhejiang University, Hangzhou, China, in 2006; the M.A.Sc. degree in Electrical and Computer Engineering from the University of Toronto, Toronto, ON, Canada, in 2008; and the Ph.D. degree in Systems Science from the Tokyo Institute of Technology, Tokyo, Japan, in 2011. He is currently a Full Professor at Osaka Metropolitan University. Previously, he was an Associate Professor at Osaka City University (2014–2020), an Assistant Professor at the University of Tokyo (2013–2014), and a Postdoctoral Fellow at the University of Toronto (2011–2013).

Dr. Cai's research interests include cooperative control of multi-agent systems, discrete-event systems, and cyber-physical systems. He is the co-author (with W.M. Wonham) of Supervisory Control of Discrete-Event Systems (Springer 2019) and Supervisor Localization (Springer 2016). He is serving as an Associate Editor for IEEE Transactions on Automatic Control and a Senior Editor for Nonlinear Analysis: Hybrid Systems. He was the Chair for IEEE CSS Technical Committee on Discrete Event Systems (2019–2022), and a member of IEEE CSS Conference Editorial Board (2017–2022). He received the Pioneer Award of SICE in 2021, the Best Paper Award of SICE in 2013, the Best Student Paper Award of IEEE Multi-Conference on Systems & Control in 2010, and the Young Author's Award of SICE in 2010.