# No synthesis needed, we are alright already

Martijn Goorden[1] and Martin Fabian[2]

***Abstract*— Supervisory control theory provides means to synthesize supervisors for cyber-physical systems based on models of the uncontrolled plant and models of the control requirements. In general, it has been shown that supervisory control synthesis is NP-hard, which is not beneficial for the applicability to industrial-sized systems. However, supervisory control synthesis seems to be easy for several industrial-sized systems compared to the theoretical worst-case complexity. In this paper, we propose properties to identify easy supervisory control problems. When a system satisfies these properties, we show that the plant models and the requirement models together are a controllable, nonblocking, and maximally permissive supervisor, i.e., no synthesis is needed to calculate a supervisor. Furthermore, these properties allow for local verification of each plant and requirement model separately.**

## I. INTRODUCTION

The design of supervisors for cyber-physical systems has become a challenge as these high-tech systems include more and more components to control and functions to fulfill, while at the same time market demands require verified safety, decreasing costs and decreasing time-to-market for these systems. Model-based systems engineering methodologies can help in overcoming these difficulties.

For the design of supervisors, the supervisory control theory of Ramadge-Wonham [1], [2] provides means to synthesize supervisors from a model of the uncontrolled plant and a model of the control requirements. Then synthesis guarantees by construction that the closed-loop behavior of the supervisor and the plant adheres to all requirements, is nonblocking, is controllable, and is maximally permissive.

Supervisors can be implemented on several different hardware platforms, of which the Programmable Logic Controller (PLC) is the one typically used [3]. Those hardware platforms have in common that the supervisor receives sensor signals through the input channels and sends actuator signals through the output channels.

Models on this input/output level are very well suitable for supervisory control theory as shown by [4]. The notion of controllable events match with (actuator) commands given by the supervisor to the plant, and the notion of uncontrollable events match with responses of the plant to these commands.

Recently, several models of industrial-size applications have been published that utilizes this input/output perspective, among them [5], [6], [7]. Analyzing the results of these cases, one discovers that the synthesized supervisors do not impose additional restrictions on the plant, i.e., the provided set of requirement models is sufficient to control the plant such that the closed-loop behavior is nonblocking, controllable, and maximally permissive. Therefore, time and computing resources have been wasted, as synthesis turned out to be unnecessary. If it was known beforehand that a synthesized supervisor would not impose additional restrictions, then this time and computing resources could be saved. Furthermore, if properties are defined that lead to a supervisor that does not impose any additional restrictions, one could try to model the plant in such a way that those properties are fulfilled, and thus immediately know that by construction it fulfills the requirements of a supervisor.

The main contribution of this paper is to provide a set of properties such that, if a set of plant models and requirement models satisfy these properties, no synthesis is needed. The proposed properties match the input/output perspective. Furthermore, verifying the properties does not suffer the notorious state-space explosion problem: the only global property concerns sharing of events, all other properties can be verified considering only a single plant model or a requirement model.

As far as the knowledge of the authors reach, no similar properties have been proposed before within the community of supervisory control theory. In [8] it was already noted that by observing real-world problems more closely one could discover instances of supervisory control synthesis that are no longer NP-hard. Unfortunately, the authors do not include any suggestion of what these instances might be or how to find these. Within the community of reactive synthesis, a class of LTL formulas exists, called Generalized reactivity(1), for which it is known that the synthesis problem can be solved in $N^3$ time, where $N$ is the size of the state space, instead of the theoretical double exponential lower bound for the general case [9]. Restricting this class even further can result in $N^2$ solutions [10]. Furthermore, the authors of [9] argue that the class of Generalized reactivity(1) is sufficiently expressive to provide complete specifications for many design problems suitable for reactive synthesis. Those readers interested in the similarities and differences between supervisory control synthesis and reactive synthesis are referred to [11]. Improved calculation complexity notwithstanding, those approaches still rely on synthesis, whereas properties presented in this paper do away with synthesis altogether; if the defined properties of the plant and the specification are met, the resulting model will be a correct supervisor, so no synthesis is needed.

The structure of this paper is as follows. In Section II the

*This work is supported by Rijkswaterstaat, part of the Ministry of Infrastructure and Water Management of the Government of the Netherlands

[1]Martijn Goorden is with Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands. m.a.goorden@tue.nl

[2]Martin Fabian is with the Department of Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden. fabian@chalmers.se

preliminaries of this paper are provided. The properties are presented in Section III. In Section IV examples are provided of models each satisfying all properties except one. For each example the intuition behind the property is explained by showing that synthesis is needed. In Section V it is proven that any set of plant models and requirement models satisfying the presented properties are by its own already nonblocking and controllable, and therefore also maximally permissive. Section VI concludes the paper.

## II. PRELIMINARIES

This section provides a brief introduction of languages, automata, and supervisory control theory. For a more in-depth introduction, the reader is referred to [12], [13].

### A. Languages

Let alphabet $\Sigma$ be a finite set of event labels and let $\Sigma^*$ be a set of all finite strings of elements in $\Sigma$, including the empty string $\varepsilon$. A string $u \in \Sigma^*$ is a prefix of $v \in \Sigma^*$ if there exists a string $s \in \Sigma^*$ such that $v = us$. The alphabet $\Sigma = \Sigma_c \cup \Sigma_u$ is partitioned into two disjoint sets containing the controllable events ($\Sigma_c$) and the uncontrollable events ($\Sigma_u$).

A language over $\Sigma$ is any subset of $\Sigma^*$. The empty language is denoted by $\emptyset$. The behavior of a discrete-event system (DES) can be modeled by language $L \subseteq \Sigma^*$.

The prefix closure of a language $L$ is given by $\overline{L} = \{u \in \Sigma^* | (\exists v \in \Sigma^*) uv \in L\}$. A language is called prefix closed if $\overline{L} = L$.

### B. Automata

An automaton is a five-tuple $G = (Q, \Sigma, \delta, q_0, Q_m)$, where $Q$ is the (finite) state set, $\Sigma$ is the alphabet of events, $\delta : Q \times \Sigma \to Q$ the partial function called the transition function, $q_0 \in Q$ the initial state, and $Q_m \subseteq Q$ the set of marked states.

We denote with $\delta(q, \sigma)!$ that there exists a transition from state $q \in Q$ labeled with event $\sigma$, i.e., $\delta(q, \sigma)$ is defined. The transition function can be extended in a natural way to strings as $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$ where $s \in \Sigma^*$, $\sigma \in \Sigma$, and $\delta(q, s\sigma)!$ if $\delta(q, s)! \wedge \delta(\delta(q, s), \sigma)!$. We define $\delta(q, \varepsilon) = q$ for the empty string. The language generated by the automaton $G$ is $\mathcal{L}(G) = \{s \in \Sigma^* \mid \delta(q_0, s)!\}$ and the language marked by the automaton $G$ is $\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) \mid \delta(q_0, s) \in Q_m\}$.

A path $p$ of an automaton is defined as a sequence of alternating states and events, i.e., $q_1\sigma_1 q_2\sigma_2 \dots \sigma_{n-1} q_n \sigma_n q_{n+1}$ such that for step $i = 1 \dots n$ it holds that $\delta(q_i, \sigma_i) = q_{i+1}$. A path can also be written in infix notation $q_1 \xrightarrow{\sigma_1} q_2 \xrightarrow{\sigma_2} \dots \xrightarrow{\sigma_{n-1}} q_n \xrightarrow{\sigma_n} q_{n+1}$.

A state $q$ of an automaton is called reachable if there is a string $s \in \Sigma^*$ with $\delta(q_0, s)!$ and $\delta(q_0, s) = q$. The automaton $G$ is called reachable if every state $q \in Q$ is reachable. A state $q$ is coreachable if there is a string $s \in \Sigma^*$ with $\delta(q, s)!$ and $\delta(q, s) \in Q_m$. The automaton $G$ is called coreachable if every state $q \in Q$ is coreachable. An automaton is called nonblocking if every reachable state is coreachable. An automaton is called trim if it is reachable and coreachable. Notice that a trim automaton is nonblocking, but a nonblocking automaton may not be trim, since it may have unreachable states.

An automaton is called a strongly connected automaton if from every state you can reach all other states, i.e., given a pair of states $q_1, q_2 \in Q$ there exists a string $s \in \Sigma^*$ such that $\delta(q_1, s) = q_2$ [14].

Two automata can be combined by synchronous composition.

*Definition 1:* Let $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$, $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$ be two automata. The synchronous composition of $G_1$ and $G_2$ is defined as

$$G_1 \parallel G_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1\|2}, (q_{0,1}, q_{0,2}), Q_{m,1} \times Q_{m,2})$$

where

$$\delta_{1\|2}((x_1, x_2), \sigma) = \begin{cases} (\delta_1(x_1, \sigma), \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2, \delta_1(x_1, \sigma)!, \\ & \text{and } \delta_2(x_2, \sigma)! \\ (\delta_1(x_1, \sigma), x_2) & \text{if } \sigma \in \Sigma_1 \setminus \Sigma_2 \text{ and } \delta_1(x_1, \sigma)! \\ (x_1, \delta_2(x_2, \sigma)) & \text{if } \sigma \in \Sigma_2 \setminus \Sigma_1 \text{ and } \delta_2(x_2, \sigma)! \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Synchronous composition is associative and commutative up to reordering of the state components in the composed state set.

A composed system $\mathcal{G}$ is a collection of automata, i.e., $\mathcal{G} = \{G_1, \dots, G_m\}$. The synchronous composition of a composed system $\parallel \mathcal{G}$ is defined as $\parallel \mathcal{G} = G_1 \parallel \dots \parallel G_m$. A composed system is called a product system if the alphabets are pairwise disjoint, i.e., $\Sigma_i \cap \Sigma_j = \emptyset$ for all $i, j \in [1, m], i \neq j$ [2].

Finally, let $G$ and $K$ be two automata with the same alphabet $\Sigma$. $K$ is said to be controllable with respect to $G$ if, for every string $s \in \Sigma^*$ and $u \in \Sigma_u$ such that $\delta_K(q_{0,K}, s)!$ and $\delta_G(q_{0,G}, su)!$, it holds that $\delta_K(q_{0,K}, su)!$.

### C. Supervisory control theory

The objective of supervisory control theory [1], [2], [12], [13] is to design an automaton called a supervisor that has the function to dynamically disable controllable events so that the closed loop system of the plant and the supervisor obeys some specified behavior. More formally, given a plant model $P$ and requirement model $R$, the goal is to synthesize supervisor $S$ that adheres the following control objectives.

- *Safety*: all possible behavior of the closed-loop system $P \parallel S$ should always satisfy the imposed requirements, i.e., $\mathcal{L}(P \parallel S) \subseteq \mathcal{L}(P \parallel R)$
- *Controllability*: uncontrollable events may never be disabled by the supervisor, i.e., $S$ is controllable with respect to $P$.
- *Nonblockingness*: the closed-loop system should be able to reach a marked state from every reachable state, i.e., $P \parallel S$ is nonblocking.

- *Maximal permissiveness*: the supervisor does not restrict more behavior than strictly necessary to enforce safety, controllability, and nonblockingness, i.e., for all other supervisors $S'$ it holds that $\mathcal{L}(P \parallel S') \subseteq \mathcal{L}(P \parallel S)$.

*Monolithic supervisory control synthesis* results in a single supervisor $S$ from a single plant model and a single requirement model [1]. We refer to $S$ as the automaton representation of the synthesized supervisor and we assume that $S = P \parallel S$, as only deterministic automata are considered. When the plant model and the requirement model are given as a composed system $P_s$ and $R_s$, respectively, the monolithic plant model $P$ and requirement model $R$ are obtained by performing the synchronous composition of the models in the respective composed system. Furthermore, $S$ can be obtained by calculating the supremal element of the set of controllable and nonblocking supervisors, i.e., $S = \sup CN(P, R)$.

For the purpose of supervisor synthesis, requirements can be modeled with automata and state-based expressions [15], [16]. The latter is useful in practice, as engineers tend to formulate requirements based on states of the plant. There are two forms of state-based expressions: state invariant expressions and state-event invariant expressions. To refer to states of the plant, we introduce the following notation and interpretation (inspired by [17]). Let $P$ be an automaton and $x \in Q$ the "current" state of automaton $P$, i.e., $x = \delta(q_0, s)$ after performing string $s$. The expression $P.q(x) \equiv P.q = x$ evaluates to true if $x$ equals state $q$ of automaton $P$, otherwise it evaluates to false. Therefore, state references can be combined with the Booleans literals **T** and **F** and logic connectives to create predicates. In the remainder of the paper, we use the notation $P.q$ as a short hand for $P.q(x)$.

A state invariant expression formulates state-based conditions that should always hold, i.e., a state invariant expression should evaluate to true for all reachable states in the closed-loop system. A state invariant expression $R$ can be converted to an automaton in the following way. Let $plants(R)$ denote the set of plant models mentioned in $R$. Then we create the automaton representation $R_a$ by taking the synchronous composition of all plants in $plants(R)$, removing the states that evaluate $R$ to false, and removing the transitions going to these removed states. Therefore, we define the synchronous composition of an automaton $P$ with a state invariant expression $R$, denoted with $P \parallel R$, as the synchronous composition of $P$ and the automaton representation $R_a$ of $R$, i.e., $P \parallel R = P \parallel R_a$.

A state-event invariant expression formulates conditions on the enablement of an event based on states of the plant, i.e., a state-event invariant expression should evaluate to true for the event to be enabled. A state-event invariant expression is of the form $\sigma$ **needs** $C$ where $\sigma$ is an event and $C$ a predicate stating the condition. Let $R$ be a state-event invariant expression, then $event(R)$ returns the event $\sigma$ used in $R$ and $cond(R)$ returns the condition predicate $C$. The synchronous composition of a plant $P$ with a state-event invariant expression $R$, denoted with $P \parallel R$, is defined by altering the transition function $\delta$.

*Definition 2:* Let $P = (Q, \Sigma, \delta, q_0, Q_m)$ and $R = \mu$ **needs** $C$. Then the synchronous composition of $P$ and $R$ is defined as

$$P \parallel R = (Q, \Sigma, \delta', q_0, Q_m)$$

where $\delta'(q, \sigma) = \delta(q, \sigma)$ unless $\sigma = \mu$ and $C_{|P.q} = \mathbf{F}$ where $C_{|P.q}$ indicates that all state references $P.q$ in $C$ are substituted by **T** and all state references $P.r, r \in Q, r \neq q$ in $C$ replaced by **F**.
This interpretation can be easily extended to a set of state-event invariant expressions $R_s = \{R_1, \ldots, R_n\}$.

Given a composed system representation of the plant $P_s = \{P_1, \ldots, P_m\}$ and a collection of requirements $R_s = \{R_1, \ldots, R_n\}$, we define the tuple $(P_s, R_s)$ as the control problem for which we want to synthesize a supervisor. We make the following (technical) assumptions about this control problem:

- $P_s \neq \emptyset$, while $R_s$ can be the empty set.
- All $P \in P_s$ agree on the controllability status of each shared event.
- For all $P \in P_s$, it holds that $P$ is an automaton where $Q_P$ and $\Sigma_P$ are nonempty.
- For all $R \in R_s$, it holds that
  - if $R$ is an automaton, then $Q_R$ and $\Sigma_R$ are nonempty, and $\Sigma_R \subseteq \Sigma_P$ where $\Sigma_P = \bigcup_{i=1}^{m} \Sigma_{P_i}$,
  - if $R$ is a state invariant expression, then for each state reference $P.q$ it holds that $P \in P_s$ and $q \in Q_P$,
  - if $R$ is a state-event invariant expression, then $event(R) \in \Sigma_P$, and for each state reference $P_i.q$ in $cond(R)$ it holds that $P_i \in P_s$ and $q \in Q_{P_i}$,

*Modular supervisory control synthesis* uses the fact that often the desired behavior is specified with a collection of requirements $R_s$ [18]. Instead of first transforming the collection of requirements into a single requirement, as monolithic synthesis does, modular synthesis calculates for each requirement a supervisor based on the plant model. In other words, given a control problem $(P_s, R_s)$ with $R_s = \{R_1, \ldots, R_n\}$, modular synthesis solves $n$ control problems $(P_s, \{R_1\}), \ldots, (P_s, \{R_n\})$. Each control problem $(P_s, \{R_i\})$ for $i \in [1, n]$ results in a safe, controllable, nonblocking, and maximally permissive supervisor $S_i$. Unfortunately, the collection of supervisors $S_s = \{S_1, \ldots, S_n\}$ can be conflicting, i.e., $S_1 \parallel \ldots \parallel S_n$ can be blocking. A nonconflicting check can verify whether $S_s$ is nonconflicting [19], [20]. In the case that $S_s$ is nonconflicting, $S_s$ is also safe, controllable, nonblocking, and maximally permissive for the original control problem $(P_s, R_s)$. In the case that $S_s$ is conflicting, an additional coordinator $C$ can be synthesized such that $S_s \cup \{C\}$ is safe, controllable, nonblocking, and maximally permissive for the original control problem $(P_s, R_s)$ [21].

## III. NONBLOCKING MODULAR SUPERVISORS

In this section we first describe several characteristics of industrial-sized applications where synthesis does not add

any restrictions beside those implied by the requirements. Then, we provide properties that together guarantee controllable and nonblocking modular supervisors that are together nonconflicting. Finally, we provide the formal proofs.

### A. Characteristics of models

First, as the supervisors synthesized for the industrial applications presented in [5], [6], [7] are intended to be implemented on control hardware, the input-output perspective of [4] is used. This entails that each sensor is modeled with uncontrollable events, while actuators are modeled with controllable events. This modeling paradigm results in a collection of numerous small plant component models that are loosely coupled, if at all, by shared events. Therefore, the plant model is a product system.

In the rest of this paper we call an automaton a sensor automaton if its alphabet only has uncontrollable events; an automaton is called an actuator automaton if the alphabet only contains controllable events.

Secondly, both sensors and actuators have cyclic behavior, resulting in a trim and strongly connected model. For example, all sensors and actuators in a production line are modeled in this way in [6]. Furthermore, unreachable states in an uncontrolled plant represent states that are physically impossible to reach and are often not modeled or removed from the model.

Finally, requirements for industrial-sized applications often originate from safety risk analysis [22]. States are identified in which some actuator actions would result in unsafe behavior. For example, the safety specifications of a waterway lock that need to be fulfilled by the supervisor are described in Section 4.191 of [23]. Each of the 16 requirements describes a state of the plant and the disablement of certain actuator actions for that state. It is shown in [5] that these textual specifications can easily be described with state-event invariant expressions.

### B. Properties

The following properties together guarantee that the control problem itself is a modular globally nonblocking and controllable supervisor.

**CNMSP** (Controllable and Nonblocking Modular Supervisors Properties)
A control problem $(P_s, R_s)$ satisfies **CNMSP** if it has the following properties:

1) $P_s$ is a product system
2) For all $P \in P_s$ it holds that
   a) $P$ is trim (which implies nonblocking)
   b) $P$ is a strongly connected automaton
3) For all $R \in R_s$ it holds that
   a) $R$ is a state-event invariant expression $e$ **needs** $C$
   b) There exists no other requirement for this event $e$
   c) $e \in \Sigma_c$
   d) $C = \bigvee \bigwedge X$, i.e., disjunctive normal form where $X$ is $P_i.l$ or $\neg P_i.l$
   e) $P_i$ is a sensor model
   f) Each conjunction contains at most one reference to each $P_i$
   g) When $P_i$ only has a single state, $\neg P_i.l$ is not allowed

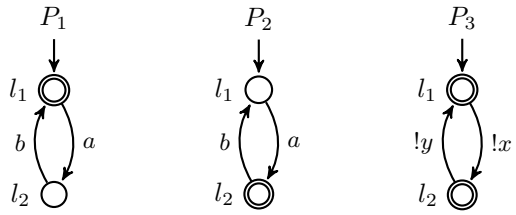



Fig. 1: Example violating Property 1 of **CNMSP**. In this and subsequent figures, marked states are indicated by concentric circles and uncontrollable events are prefixed with an exclamation mark.

The first workstation model of the FESTO production line as described in [6] satisfies these properties. Therefore, the provided plant and requirement models are sufficient to act as controllable and nonblocking modular supervisors, as formulated in the following theorem. In that case, the modular supervisor represented by the plant models and requirement models is by definition also maximally permissive.

*Theorem 1:* Let $P$ and $R$ be a set of plant models and requirement models satisfying **CNMSP**. Then no supervisor synthesis is required, i.e., $P \parallel R$ is controllable and nonblocking.

Before we prove this theorem, we will strengthen the intuition of the properties in the next section.

## IV. EXAMPLES OF BLOCKING MODULAR SUPERVISORS

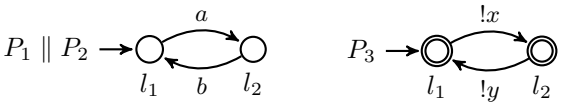
In this section, several examples of blocking modular supervisors are provided. In each example, only a single property of **CNMSP** is violated. These examples show the intuition behind the proposed properties. Furthermore, the examples also show that each **CNMSP** property is essential, removing any one (or more) of them no longer guarantees a nonblocking system without synthesizing a supervisor.

### A. Property 1 removed

If plant component models are allowed to share events, the control problem in Figure 1 would be allowed. $P = P_1 \parallel P_2 \parallel P_3$ is no longer a product system as $P_1$ and $P_2$ share events $a$ and $b$. Since $P_1 \parallel P_2$ becomes unmarked, a synthesized supervisor $S_1$ for this control problem is the null supervisor, hence it is not equal to $P \parallel R_1$.

### B. Property 2.a removed

The control problem in Figure 2 has a non-trim plant component model. This example solved the problem of Section IV-A (where $P$ was not a product system) by synchronizing the plant component automata that share events.




$R_1 : a$ **needs** $P_3.l_1$

Fig. 2: Example violating Property 2.a of **CNMSP**.


$R_1 : b$ **needs** $P_2.l_1$

Fig. 3: Example violating Property 2.b of **CNMSP**.

Unfortunately, now the property that each plant component automaton should be trim is violated, and again a synthesized supervisor $S_1$ would be the null supervisor, hence it is not equal to $P \parallel R_1$.

### C. Property 2.b removed

In the example of Figure 3, $P_2$ is trim, but not strongly connected, so condition $R_1$ is not fulfilled once $P_2$ left its initial state $P_2.l_1$, which blocks the plant when $P_1$ is in state $P_1.l_1$. As $x$ is uncontrollable, no supervisor can prevent $P_2$ from leaving $P_2.l_1$. Any supervisor must therefore disable event $a$ to prevent $P_1$ entering state $P_1.l_2$, which solves the blocking issue. As the maximally permissive supervisor must disable events, it does not hold that $S_1 = P \parallel R_1$.

### D. Property 3.a removed

If requirements other than state-event invariant expressions are allowed, the control problem in Figure 4 would be allowed. In this example, we used a state invariant expression excluding the globally marked state. Therefore, a synthesized nonblocking supervisor $S_1$ would be the null supervisor, hence it is not equal to $P \parallel R_1$.

### E. Property 3.b removed

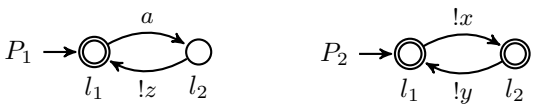
If more than one state-event invariant expression exists for the same event $e$, the control problem in Figure 5 would be allowed. Here, when $P_1$ reaches state $P_1.l_2$, the only transition to a marked state is labeled with event $b$. The condition of requirement $R_1$ can always eventually become true as state $P_2.l_1$ can always eventually be reached. Therefore, supervisor $S_1 = P \parallel R_1$. The same argument applies for requirement $R_2$, resulting in $S_2 = P \parallel R_2$. Unfortunately, $S_1$ and $S_2$ are conflicting as $P_2$ can only be either in $P_2.l_1$ *or* $P_2.l_2$, thus only the condition of $R_1$ *or* $R_2$ can be true at the same time.


$R_1 : \neg(P_1.l_2 \wedge P_2.l_2)$

Fig. 4: Example violating Property 3.a of **CNMSP**.


$R_1 : b$ **needs** $P_2.l_1$ $\quad R_2 : b$ **needs** $P_2.l_2$

Fig. 5: Example violating Property 3.b of **CNMSP**.




$R_1 : !z$ **needs** $P_2.l_1$

Fig. 6: Example violating Property 3.c of **CNMSP**.

### F. Property 3.c removed

If for the state-event invariant expression $e$ **needs** $C$ the event $e$ was allowed to be uncontrollable, the problem illustrated in Figure 6 could arise. As requirement $R_1$ is not controllable with respect to the plant component $P_1$, any supervisor would have to disable some controllable event, $a$ in Figure 6, and so a synthesized supervisor $S_1$ would not be equal to $P \parallel R_1$.

### G. Property 3.d removed

If the condition $C$ of a state-event invariant expression could be any Boolean expression, the control problem in Figure 7 is allowed. In this example, as $P_2$ has only 2 states, the expression $P_2.l_1 \vee P_2.l_2$ will always results in true, and thus $\neg(P_2.l_1 \vee P_2.l_2)$ will always results in false. Therefore, when $P_1$ is in state $P_1, l_2$, event $b$ is disabled and $P_1$ cannot reach a marked state. To resolve this, any synthesized supervisor $S_1$ needs to disable event $a$ to prevent $P_1$ from entering state $P_1.l_1$, hence it is not equal to $P \parallel R_1$.

### H. Property 3.e removed

In the example of Figure 8, a state of a sensor model is used in the condition $C$ of the state-event invariant expression. Here, it still holds for each individual supervisor that $S_j = P \parallel R_j$, but the collection of supervisors $S$ is blocking, as when the plant is in state $(P_1.l_2, P_2.l_2)$, no transition is possible. To be able to perform the transition labeled with $b$, plant component $P_2$ needs to be in state $P_2.l_1$. But to get $P_2$ to this state, the transition labeled with event $d$ needs to be performed. Unfortunately, event $d$ is only enabled when plant component $P_1$ is in state $P_1.l_1$. But to get $P_1$ to this state, the transition labeled with event $b$ needs to be performed. Now we have cycled back in our


$R_1 : b$ **needs** $\neg(P_2.l_1 \vee P_2.l_2)$

Fig. 7: Example violating Property 3.d of **CNMSP**.


$R_1 : b$ **needs** $P_2.l_1$ $\quad$ $R_2 : d$ **needs** $P_1.l_1$

Fig. 8: Example violating Property 3.e of **CNMSP**.


$R_1 : b$ **needs** $(P_2.l_1 \wedge P_2.l_2)$

Fig. 9: Example violating Property 3.f of **CNMSP**.

argumentation, concluding that there does not exist a path to a globally marked state. Thus $S$ is blocking.

### *I. Property 3.f removed*

If a conjunction could have more than one reference to the same plant component model, the control problem in Figure 9 is allowed. As an automaton can only be in one state at a time, $P_2$ is either in $P_2.l_1$ *or* $P_2.l_2$. Therefore, the condition $(P_2.l_1 \wedge P_2.l_2)$ evaluates in this example always to false. Thus, when $P_1$ reaches state $P_1.l_2$, no path exists to a marked state. To resolve this, any synthesized supervisor $S_1$ needs to disable event $a$, hence it is not equal to $P \parallel R_1$.

### *J. Property 3.g removed*

If the negation can be used for referring to a single-state plant component, the problem illustrated in Figure 10 could arise. In this example, as $P_2$ has only a single state, the predicate $\neg P_2.l_1$ evaluates always to false. Thus, when $P_1$ reaches state $P_1.l_2$, no path exists to a marked state. To resolve this, any synthesized supervisor $S_1$ needs to disable event $a$, hence it is not equal to $P \parallel R_1$.

## V. PROOFS

In order to prove that a control problem satisfying **CNMSP** does not require synthesis (Theorem 1), we start by proving the following five lemmas.

The first two lemmas show that when a plant model is provided as a product system and each individual automaton is trim or strongly connected, then the synchronous composition of these automata is also trim or strongly connected, respectively.

*Lemma 1:* Let $P_s = \{P_1, \ldots, P_m\}$ be a product system where each individual $P_i \in P_s$ is trim. Then $P_1 \parallel \ldots \parallel P_m$ is trim.


$R_1 : b$ **needs** $\neg P_2.l_1$

Fig. 10: Example violating Property 3.g of **CNMSP**.

*Proof:* Denote $P = P_1 \parallel \ldots \parallel P_n$, $P = (Q, \Sigma, \delta, q_0, Q_m)$, and $P_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$. We show that $P$ is reachable and coreachable.

Firstly, assume that $q = (q_1, \ldots, q_n)$ is a state in $P$. As each individual $P_i$ is trim, it follows that there exists a string $s_i \in \Sigma_i^*$ such that $\delta_i(q_{0,i}, s_i) = q_i$. From the definition of synchronous composition and the fact that $P_s$ is a product system, it follows that $\delta((r_1, \ldots, q_{0,i}, \ldots, r_m), s_i) = (r_1, \ldots, q_i, \ldots, r_m)$ for any state $r_j \in Q_j, j \neq i$. Therefore, it holds that $\delta((q_{0,1}, \ldots, q_{0,n}), s_1 s_2 \ldots s_n) = q$ in $P$. As the state $q$ is chosen arbitrarily, it follows that $P$ is reachable.

Secondly, assume again that $q = (q_1, \ldots, q_n)$ is a state in $P$. As each individual $P_i$ is trim, it follows that there exists a string $s_i \in \Sigma_i^*$ such that $\delta_i(q_i, s_i) = q_{i,k} \in Q_{m,i}$. From the definition of synchronous composition and the fact that $P_s$ is a product system, it follows that $\delta((r_1, \ldots, q_i, \ldots, r_m), s_i) = (r_1, \ldots, q_{i,k}, \ldots, r_m)$ where $q_{i,k}$ for any state $r_j \in Q_j, j \neq i$. Therefore, it holds that $\delta(q, s_1 s_2 \ldots s_n) \in Q_m$ in $P$. As state $q$ is chosen arbitrarily, it follows that $P$ is coreachable. ■

*Lemma 2:* Let $P_s = \{P_1, \ldots, P_m\}$ be a product system where each individual $P_i \in P_s$ is a strongly connected automaton. Then $P_1 \parallel \ldots \parallel P_m$ is a strongly connected automaton.

*Proof:* Denote $P = P_1 \parallel \ldots \parallel P_n$, $P = (Q, \Sigma, \delta, q_0, Q_m)$, and $P_i = (Q_i, \Sigma_i, \delta_i, q_{0,i}, Q_{m,i})$. We show that for any pair of two states $x = (x_1, \ldots, x_m) \in Q, y = (y_1, \ldots, y_m) \in Q$ there exists a string $s \in \Sigma^*$ such that $\delta(x, s) = y$.

As each individual $P_i$ is strongly connected, it follows that there exists a string $s_i \in \Sigma_i^*$ such that $\delta_i(x_i, s_i) = y_i$. From the definition of synchronous composition and the fact that $P_s$ is a product system, it follows that $\delta((r_1, \ldots, x_i, \ldots, r_m), s_i) = (r_1, \ldots, y_i, \ldots, r_m)$ for any state $r_j \in Q_j, j \neq i$. Therefore, it holds that $\delta(x, s_1 s_2 \ldots s_n) = y$ in $P$. As states $x$ and $y$ are chosen arbitrarily, it follows that $P$ is a strongly connected automaton. ■

The following lemma expresses that when a control problem with a single requirement satisfies **CNMSP**, then we can always eventually reach a state such that the condition of this requirement evaluates to true, thus enabling the guarded event.

*Lemma 3:* Let $(P_s, \{R\})$ be a control problem with a single requirement satisfying **CNMSP**. Denote $R = e$ **needs** $C$. Then, from any state $q \in Q$, there exists a string $s \in \Sigma^*$ such that a state $r$ is reached and $C(r) = \mathbf{T}$.

*Proof:* As $P_s$ is a product system (Property 1), there is only a single plant component $P_k$ such that $e \in \Sigma_k$. From the combination of properties 3.c-e, it follows that plant component $P_k$ is not used in condition $C$, as it has to be an actuator model. Therefore, the state of $P_k$ does not matter.

Furthermore, observe that $P_s \setminus \{P_k\} = (P_s \setminus \{P_k\}) \parallel R$. From Property 2.b and Lemma 2 it follows that $P_s \setminus \{P_k\}$ is a strongly connected automaton, thus $P_s \setminus \{P_k\} \parallel R$ is also a strongly connected automata. Therefore, if there exists a state $r$ that satisfies $C$, i.e., $C(r) = \mathbf{T}$, then there also exists

a string $s \in \Sigma^*$ such that $\delta(q, s) = r$. So it remains to be proven that such a state $r$ exists.

As $C$ is in disjunctive normal form (Property 3.d), it follows that if $r$ satisfies $C$, it satisfies one of the conjunctions. From properties 3.e and 3.f we know that there is at most one reference to each $P_i \in P_s \setminus \{P_k\}$ in each conjunction. If there is no reference to $P_i$, then all states of this automaton satisfy this conjunction. If $P_i$ is mentioned in this conjunction, then, from properties 3.d and 3.g, there exists at least one state $q_i \in Q_i$ that satisfies this conjunction. Thus there exists a state $r$ such that $C$ is satisfied. ∎

Now we prove the following two lemmas: the first one shows that under the given conditions, we do not have to do synthesis locally, and the second one shows that under the given conditions the supervisors are globally nonblocking.

*Lemma 4:* Let $(P_s, R_s)$ be a control problem satisfying **CNMSP**. Construct the set of modular supervisors $S = \{S_1, \ldots, S_n\}$ such that each supervisor $S_j = \sup CN(P, R_j)$ is the maximally permissive controllable and nonblocking supervisor for plant $P =\| P_s$ and requirement $R_j \in R$. Then $S_j = P \parallel R_j$.

*Proof:* In the case that $R_s = \emptyset$, no supervisor is synthesized. It follows from properties 1 and 2.a and Lemma 1 that $P$ is trim, so there is indeed no need for a supervisor. In the remainder of the proof we assume that $R_s \neq \emptyset$.

For each individual supervisor $S_j$ we show that $R_j$ is controllable with respect to plant $P$ and that $P \parallel R_j$ is nonblocking. The fact that $R_j$ is controllable follows directly from Property 3.c. It remains to be proven that $P \parallel R_j$ is nonblocking. From Property 3.a we have an event $e_j = event(R_j)$ associated with this requirement $R_j$. As $P_s$ is a product system (Property 1), there is only a single plant component $P_k$ such that $e_j \in \Sigma_k$. Now we partition the set of plant component models into $\{P_k\}$, $P_{sm} = \{P_i \in P_s \mid P_i \text{ is a sensor model}\}$, and $P_o = P_s \setminus (\{P_k\} \cup P_{sm})$. Observe that the behavior of the plant components in $P_{sm}$ and $P_o$ are not altered by requirement $R_j$, so lemmas 1 and 2 apply on the sets $P_{sm}$, $P_o$, and $P_{sm} \cup P_o$, i.e, $P_{sm} \parallel R_j$, $P_o \parallel R_j$, and $(P_{sm} \cup P_o) \parallel R_j$ are all trim and strongly connected automata.

To show that $P \parallel R_j$ is nonblocking, we show that for each reachable state $q$ there exists a string $s \in \Sigma^*$ such that a marked state $q_m \in Q_m$ can be reached. Consider automaton $P_k$ with current state $q_k$. As automaton $P_k$ is trim (Property 2.a), there exists a path labeled with string $s_k \in \Sigma_k^*$ such that a state $q_{m,k} \in Q_{m,k}$ can be reached from state $q_k$. We will show that this path is still possible under the influence of requirement $R_j$, i.e., it is still a path in $P_k \parallel R_j$. Consider two cases for this path.

- If $s_k$ does not contain event $e_j$, then the path labeled with $s_k$ is trivially possible in $P_k \parallel R_j$.
- If $s_k$ contains event $e_j$, then requirement $R_j$ may remove event $e_j$ from the enabled event sets and preventing $P_k \parallel R_j$ from reaching a marked state. For each transition labeled with event $e_j$, we know from Lemma 3 that there exists a path in $P$ reaching a state $r$ such that $C(r) = \mathbf{T}$. Therefore, there always exists a path in $P$ such that $e_j$ is enabled. Thus, the path labeled with $s_k$ is still possible in $P_k \parallel R_j$.

Combining the above observation for $s_k$ and the fact that $(P_{sm} \cup P_o) \parallel R_j$ is trim, we know that string $s$ exists such that a marked state $q_m$ is reached from state $q$. As $q$ is arbitrarily chosen, it follows that $P \parallel R_j$ is nonblocking. ∎

*Lemma 5:* Let $(P_s, R_s)$ be a control problem satisfying **CNMSP**. Construct the set of modular supervisors $S = \{S_1, \ldots, S_n\}$ such that each supervisor $S_j = \sup CN(P, R_j)$ is the maximally permissive controllable and nonblocking supervisor for plant $P = P_1 \parallel \ldots \parallel P_m$ and requirement $R_j \in R$. Then $S$ is nonconflicting.

*Proof:* For $S$ to be nonconflicting, it should hold that $S_1 \parallel \ldots \parallel S_n$ is nonblocking. From Lemma 4 it follows that each $S_j = P \parallel R_j$. Therefore, $S_1 \parallel \ldots \parallel S_n = (P \parallel R_1) \parallel \ldots \parallel (P \parallel R_n) = P \parallel R_1 \parallel \ldots \parallel R_n$. Partition the set of plant models $P_s$ into the set of sensor models $P_{sm} = \{P_i \in P_s \mid P_i \text{ is a sensor model}\}$, the set of restricted models $P_r = \{P_i \in P_s \mid \exists R_j \in R_s \text{ s.t. } event(R_j) \in \Sigma_i\}$, and the other plant models $P_o = P_s \setminus (P_{sm} \cup P_r)$.

Clearly, no plant model in $P_o$ is affected by the requirements, so lemmas 1 and 2 apply, i.e., $P_o \parallel R_s$ is a trim and strongly connected automaton. Furthermore, from Property 3.c and the definition of a sensor model it follows that also no plant model in $P_{sm}$ is affected by the requirements, thus by lemmas 1 and 2 it follows that $P_{sm} \parallel R_s$ is a trim and strongly connected automaton. Again using lemmas 1 and 2 results that $P_o \parallel P_{sm} \parallel R_s$ is a trim and strongly connected automaton.

For $P_o \parallel P_{sm} \parallel P_r \parallel R_s$ to be nonblocking, it should hold that from every reachable state $q \in Q$ there exists a string $s \in \Sigma^*$ such that $\delta(q, s) \in Q_m$. As $P_r$ is trim (Lemma 1) it follows that there exists a string $s_r \in \Sigma_r^*$ such that $\delta(q_r, s_r) \in Q_m$ in $P_r$. From the definition of synchronous composition with a state-event requirement, it follows that $\theta(q_r, s_r) \in Q_m$ in $P_r \parallel R_s$. For $\delta(q_r, s_r) \in Q_m$ in $P_r \parallel R_s$ to hold, each event in $s_r$ should be enabled along its path. There are two cases for each event $\sigma$ in string $s_r$.

- If there does not exist a requirement $R_j \in R_s$ such that $event(R_j) = \sigma$, then $\sigma$ is enabled.
- If there does exist a requirement $R_j \in R_s$ such that $event(R_j) = \sigma$, then $R_j$ is also the only requirement in $R_s$ such that $event(R_j) = \sigma$ (Property 3.b). As the condition $C_j = cond(R_j)$ only depends on plant components from $P_{sm}$ and not plant components from $P_r$ or $P_o$ (Property 3.e), it follows from Lemma 4 that there exists a string in $P_{sm}$ such that the reached state $r$ satisfies $C_j$. No transition in plant components from $P_r$ and $P_o$ are needed as all states from these plant components are irrelevant in satisfying the condition $C_j$. Therefore, there exists a path in $P$ such that $\sigma$ is enabled.

From the above observation, we conclude that we can always find a string (including the empty string) such that $\sigma$ is enabled. As $\sigma$ is chosen arbitrarily along the path in $P_r$

labeled with $s_r$, it follows that $\delta(q_r, s_r) \in Q_{m,r}$. Finally, combining this with the fact that $q_r$ is chosen arbitrarily and that $P_o \parallel P_{sm} \parallel R_s$ is trim, it follows that $P_o \parallel P_{sm} \parallel P_r \parallel R_s$ is nonblocking. ■

Now we are ready to prove the main theorem of the paper.

*Theorem 1:* Let $P$ and $R$ be a set of plant models and requirement models satisfying **CNMSP**. Then no supervisor synthesis is required, i.e., $P \parallel R$ is controllable and nonblocking.

*Proof:* From lemmas 4 and 5 it follows that we can construct a set of supervisors $S = \{S_1, \ldots, S_n\}$ such that $S_j = \sup CN(P, R_j) = P \parallel R_j$ and $S$ is nonconflicting. The antecedent follows directly from combining these last two facts. ■

## VI. CONCLUSION

In this paper, we presented properties such that a control problem satisfying these properties is already controllable and nonblocking without synthesizing a supervisor. Therefore, the control problem itself represents a safe, controllable, nonblocking, and maximally permissive supervisor. The properties match with the input/output perspective often needed for supervisor implementations. Furthermore, the properties can be verified easily, avoiding the notorious state-space explosion problem of synthesis. Examples show that violating any one of these properties may result in the need of synthesis.

Further research is needed in the relaxation of the proposed properties. Sometimes it is desired to model the physical relation between actuators and sensors [24]. Otherwise, a supervisor that is proven to be nonblocking may block after implementation. Adding shared events to model the interactions will violate Property 1, as it is no longer a product system. Transforming this new model into a product system representation, the actuator and sensor models are combined into one. Therefore, requirements no longer refer only to states of sensor models (violating Property 3.e). Furthermore, sometimes a requirement needs to express explicitly that, for example, an actuator needs to be at rest to guarantee safety of the plant. This type of requirements also violates Property 3.e.

## ACKNOWLEDGMENT

The authors thank Michel Reniers, Joanna van de Mortel-Fronczak, and Jacobus Rooda for several fruitful discussions on the proposed set of properties and its applicability to industrial-based cases.

## REFERENCES

[1] P. J. G. 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, Jan. 1987.

[2] ——, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[3] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *Proceedings of the 37th IEEE Conference on Decision and Control*, vol. 3, 1998, pp. 3305–3310 vol.3.

[4] S. Balemi, "Control of Discrete Event Systems: Theory and Application," Ph.D. thesis, Swiss Federal Institue of Technology Zurich, Zurich, 1992.

[5] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory control synthesis for a waterway lock," in *IEEE Conference on Control Technology and Applications*, Aug. 2017, pp. 1562–1568.

[6] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, M. A. Reniers, and J. E. Rooda, "Application of dependency structure matrices and multilevel synthesis to a production line," in *IEEE Conference on Control Technology and Applications*, Aug. 2018, pp. 458–464.

[7] F. F. H. Reijnen, M. A. Goorden, J. M. van de Mortel-Fronczak, and J. E. Rooda, "Supervisory control synthesis for a lock-bridge combination," *submitted to Discrete Event Dynamic Systems*, 2019.

[8] P. Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 30, no. 5, pp. 643–652, Oct. 2000.

[9] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, Jan. 2006, pp. 364–380.

[10] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller synthesis for timed automata," *IFAC Proceedings Volumes*, vol. 31, no. 18, pp. 447–452, Jul. 1998.

[11] R. Ehlers, S. Lafortune, S. Tripakis, and M. Y. Vardi, "Supervisory control and reactive synthesis: a comparative introduction," *Discrete Event Dynamic Systems*, vol. 27, no. 2, pp. 209–260, Jun. 2017.

[12] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Boston: Springer, 2008.

[13] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*, 1st ed. Springer, 2018.

[14] M. Ito, "A representation of strongly connected automata and its applications," *Journal of Computer and System Sciences*, vol. 17, no. 1, pp. 65–80, Aug. 1978.

[15] C. Ma and W. M. Wonham, *Nonblocking Supervisory Control of State Tree Structures*, ser. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2005, no. 317.

[16] J. Markovski, K. G. M. Jacobs, D. A. van Beek, L. J. Somers, and J. E. Rooda, "Coordination of resources using generalized state-based requirements." in *10th International Workshop on Discrete Event Systems*, 2010, pp. 300–305.

[17] S. Miremadi, K. Åkesson, and B. Lennartson, "Extraction and representation of a supervisor using guards in extended finite automata," in *9th International Workshop on Discrete Event Systems*, May 2008, pp. 193–199.

[18] W. M. Wonham and P. J. G. Ramadge, "Modular supervisory control of discrete-event systems," *Mathematics of Control, Signals and Systems*, vol. 1, no. 1, pp. 13–30, Feb. 1988.

[19] S. Mohajerani, R. Malik, and M. Fabian, "A framework for compositional nonblocking verification of extended finite-state machines," *Discrete Event Dynamic Systems*, vol. 26, no. 1, pp. 33–84, Mar. 2016.

[20] R. Su, J. H. van Schuppen, J. E. Rooda, and A. T. Hofkamp, "Nonconflict check by using sequential automaton abstractions based on weak observation equivalence," *Automatica*, vol. 46, no. 6, pp. 968–978, Jun. 2010.

[21] R. Su, J. H. van Schuppen, and J. E. Rooda, "Synthesize nonblocking distributed supervisors with coordinators," in *17th Mediterranean Conference on Control and Automation*, Jun. 2009, pp. 1108–1113.

[22] M. Modarres, *Risk Analysis in Engineering : Techniques, Tools, and Trends*. CRC Press, Apr. 2016.

[23] Rijkswaterstaat, "Landelijke brug- en sluisstandaard, vraagspecificatie eisen v4.0," Dutch Ministry of Infrastructure and the Environment, Standard, October 2015, in Dutch.

[24] J. Zaytoon and V. Carre-Meneatrier, "Synthesis of control implementation for discrete manufacturing systems," *International Journal of Production Research*, vol. 39, no. 2, pp. 329–345, Jan. 2001.